Requirements Reformulation using Formal Specification: A Case Study

Luke Wildman

Software Verification Research Centre,
University of Queensland, Australia,
Email: luke@svrc.uq.edu.au

Abstract

This article describes our experience of using formal specification to reformulate the requirements of the Nulka Electronic Decoy. The Nulka Electronic Decoy is a hovering rocket that lures anti-ship missiles away from the ship. The requirements specification contained informal natural language requirements relating both to time-related performance requirements, and to other physical characteristics that were not time-related. 'Timed Interval Calculus' was used for the time-related performance requirements whereas simple mathematics was used for the others, thereby creating two different views of the Decoy. While no conflicting requirements or incorrect values were detected, 50% of the requirements were modified as a result of formalisation and consultation with domain experts. This article describes the techniques that were used, the changes that were made, reflects on lessons learned and discusses related work.

Keywords: Requirements Engineering, Formal Methods, Timed Interval Calculus

1 Introduction

Formal specification techniques apply formal mathematical notations to express system requirements. Formal mathematical notations enable precise expression and tool support that are not possible when using natural language (NL) exclusively.

This article describes a project where formal specification was used to improve the requirements given in the existing Prime Item Development Specification (PIDS) of the Nulka Electronic Decoy. In this case formal specification was applied cost-effectively: a dramatic improvement was achieved with a small cost. Costs were minimised by the use of appropriate formal notations and by using review as the sole form of analysis. This article focuses on the benefits provided by the use of an appropriate formal specification notation and review alone.

1.1 Overview of Paper

Section 2 describes the Electronic Decoy and summarises the analysis that was performed on the Decoy PIDS. Section 3 introduces the Timed Interval Calculus (TIC) (Fidge, Hayes, Martin & Wabenhorst 1998) and Section 4 then discusses how formal specification in TIC was applied to the Decoy performance requirements. Section 5 describes how simple mathematics was used to model the other physical characteristics. Section 6 discusses the results of the formal analysis. Some discussion of the possible benefits of tool support is included in Section 7. Related work is discussed in Section 8. Concluding remarks are made in Section 9.

Copyright ©2002, Australian Computer Society, Inc. This paper appeared at the Workshop on Formal Methods Applied to Defence Systems, Adelaide, Australia, June 2002. Conferences in Research and Practice in Information Technology, Vol. 12. L.M. Kristensen and J. Billington, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

2 Project Overview

The Nulka Electronic Decoy is a joint Australian/US project being developed to counter anti-ship missiles. The rocket-powered decoy is launched from its host ship and lures incoming missiles away from the ship. The Decoy PIDS establishes the performance, design, development and test requirements for the Decoy prime item of the Nulka ship launched Electronic Decoy. It has been prepared in accordance with MIL-STD-490A "Specification Practices" (Department of Defense 1985). This standard establishes uniform practice for the format and content of specifications prepared for the Department of Defence in order to ensure the inclusion of essential requirements, and to aid in the use and analysis of specification content.

2.1 PIDS Formalisation

The Nulka project was coming to the end of a long development phase motivating the development and reissue of the PIDS based on experience gained during the ten years of engineering development. As a first step, formal specification was used to capture the requirements contained in the existing PIDS. The formal model of the Decoy was then developed in consultation with Nulka project personnel with various areas of interest (Prime Contractor, Commonwealth, US) and expertise: flight dynamics, missile seduction, project management, to name a few.

The Timed Interval Calculus (Fidge, Hayes, Martin & Wabenhorst 1998) was used to formalise the performance requirements of the Decoy. Flight performance, being highly dynamic and time dependent, is notoriously difficult to specify; particularly where NL is used exclusively. However, this is where the application of formal specification had the greatest effect. Section 6 examines the result of formalisation.

Physical characteristics that were not time-related were formalised using simple mathematics involving predicates, functions and relations. Some minor revisions were made to the physical characteristics, mostly to do with unnecessary or incorrect requirements.

The formalised requirements were combined side-by-side with systematically translated NL requirements to generate the final Decoy PIDS. The NL requirements were understood to take precedence over the formal requirements. However, the formalised requirements were retained for clarification and as a basis for evolutionary requirements change.

2.2 Method

The project took place in two stages: detailed reformulation and final acceptance. The first stage involved 35 man-days taking place over a period of 6 months, during which the requirements were formalised in 5 iterations involving 4 workshops. The workshops were held with a variety of domain experts. This was for the purpose of reviewing progress and solving problems found within the original informal requirements text and its subsequent reformulations. Each iteration focussed on particular problem requirements. Attempted formalisation of these requirements raised questions which were then discussed at the workshops. Modifications were then integrated into the model.

The second stage occurred after a further year. This was in preparation for the final acceptance of the revised PIDS. The changes to the PIDS were documented and some final clarifications and corrections were made.

No tool support was used in the project. (Section 7 discusses why and considers possibilities for the incorporation of tools.) However, manual checking was possible because there were only 120 requirements. The formalised requirements were manually checked for the following properties.

- Consistency - that the formal notation used in the specification be free of syntax, type, and semantic errors. (This is not the formal definition of the word but suffices to describe what was checked.)
- Correctness - that the specification define behaviour equivalent to the informal requirements.
- Precision - that the specification be free of ambiguity.
- Abstraction - that the level of abstraction be suitable for defining a clear and concise specification.
- Completeness - that all necessary components of the system be modelled.
- Feasibility - that all requirements be implementable.
- Testability - that the requirements were able to be validated.
- Modularity - that the requirements were structured into independent, internally cohesive, components.
- Readability and Style - that naming conventions and structuring of definitions be acceptable and appropriate.

2.3 Views

The use of a second formalism constitutes a second formal *view* of the requirements. The use of multiple formal views is recommended where different aspects of a system are more easily expressed in different formalisms or frameworks (Jackson 1995). A simple *view invariant* links the views. Section 5.2 gives details of the view invariant.

2.4 Scope

While formalisation was attempted on all Decoy PIDS requirements, some requirements were not formalisable. For instance probabilistic requirements, i.e., those that are expressed probabilistically or statistically, were not formalised in this analysis because an appropriate (well understood) formalism was not available. (Section 8.3 discusses related work in this area.) Process requirements such as those to do with validation and documentation were also not addressed.

3 Overview of the formalism

The performance requirements were specified in a formalism based on the Timed Interval Calculus (TIC) (Fidge, Hayes, Martin & Wabenhorst 1998). Following a brief introduction to TIC in Section 3.1, Section 3.2 describes how interval predicates are used to model a system. Section 3.3 introduces some additional TIC operators and Section 3.4 discusses the use of differentiation of trace variables. Section 3.5 illustrates the expressiveness of TIC with some frequently used idioms.

3.1 Timed Interval Calculus

TIC is a simple set theoretic notation useful for concisely expressing time intervals and predicates over them. A system is specified by constraints on the non-empty time intervals over which properties hold. TIC is closely related to Duration Calculus (DC) (Chaochen, Hoare & Ravn 1991), featuring similar laws for interval based reasoning. However, where DC is defined in terms of temporal logic, TIC is defined in terms of set-theory. Further comparison of DC and TIC is presented elsewhere (Fidge, Hayes, Martin & Wabenhorst 1998).

Absolute time, $\mathbb{T}$, is modelled by real numbers. Here we assume non-negative real numbers.

Observable variables are modelled as traces (total functions from the time domain to a type representing the set of all values that the variable may have). For example, a variable indicating whether the Decoy is in the flight phase can be expressed as a function from time to Booleans ($\mathbb{B}$):

$\mathit{flight} : \mathbb{T} \rightarrow \mathbb{B}$	`//the Decoy is in flight phase.`

Similarly, the height of the Decoy can be expressed as a timed trace of Reals ($\mathbb{R}$).

$\mathit{height} : \mathbb{T} \rightarrow \mathbb{R}$	`//Decoy height`

Aside: We adhere to the box notation used in the formalised Decoy PIDS. Trace variable declarations and formalised requirements were added to the PIDS in a box embedded in the relevant PIDS paragraph. Comments were added to indicate the real-world interpretations of declared variables. *End Aside*

3.2 Interval Predicates

Sets of intervals can be specified using the interval brackets $\langle$ and $\rangle$. Here, a much simpler notation is used than the numerous brackets presented elsewhere (Fidge, Hayes, Martin & Wabenhorst 1998). The brackets $\langle$ and $\rangle$ represent sets of intervals where the endpoints are either open or closed. This is equivalent to $[\!\!-$ and $-\!\!]$ in Fidge *et al.* We forego the various other brackets (left-closed, right-closed etc) preferring to use the minimum of special notation.

For example, the set of all time intervals where the Decoy is in flight phase for the entire interval is specified as $\langle \mathit{flight} \rangle$. An interval I is in the set of intervals $\langle \mathit{flight} \rangle$ if and only if, for all times t in I, $\mathit{flight}(t)$ is true. Note that this includes all possible sub-intervals where flight is true. In general, the property in the brackets is any first-order predicate in which total functions from the time domain to some type X may be treated as values of type X. The elision of explicit references to the time domain of these functions results in specifications which are more concise and readable. For example, the set of all intervals where the Decoy is in flight phase and its height is greater than H metres for the whole interval is written as follows.

$$\langle \mathit{flight} \wedge \mathit{height} \geq H\mathrm{m} \rangle$$

The dimensions of constants (e.g. m, s, kg, m/s) have been included in the formal specification. We have added these annotations as a simple extension to the type system of TIC. Dimensions provide extra typing information useful when checking the consistency of the specification (Hayes & Mahony 1995). Dimensions are discussed further in Section 7.1.

Propositions are formed by combining sets of intervals using standard relations from set theory. For example, the property

```
During the flight phase the height of the
Decoy shall be not less than H metres.
```

can be specified as follows.

$$\boxed{\langle \mathit{flight} \rangle \subseteq \langle \mathit{height} \geq H\mathrm{m} \rangle}$$

That is, the set of all intervals where the Decoy is in flight phase is a subset of the set of intervals where the height of the Decoy in not less than H.

The set of all intervals of duration X seconds can be expressed as $\langle \delta = X\mathrm{s} \rangle$. The symbol δ is a reserved symbol representing the duration of an interval. Other reserved symbols are α and ω representing the *infimum* (start) and *supremum* (end) times of an interval respectively. These operators are related by $\delta = \omega - \alpha$, which alternately can be stated as an interval predicate in the following way:

$$\langle \delta = \omega - \alpha \rangle = \langle \mathit{true} \rangle$$

where $\langle \mathit{true} \rangle$ is the set of all intervals.

3.3 Interval Concatenation

More complex predicates can also be specified using the interval concatenation operator $(\_;\_)$ (chop). This operator forms a set of intervals by joining intervals from one set to those of the other whenever their end points adjoin.

$$\langle P \rangle = \langle R \rangle ; \langle S \rangle$$

In other words, an interval r from $\langle R \rangle$ above can be joined to an interval s from $\langle S \rangle$ to form a a new interval p in $\langle P \rangle$ if r occurs strictly before s and the two endpoints meet exactly, with no overlap or gap (Fidge, Hayes, Martin & Wabenhorst 1998). Importantly, the supremum of r must equal the infimum of s. However, so that no overlap occurs, this point may only lie in one of the intervals. That is, the joining interval endpoints must be either open-closed or closed-open. Here, where the interval brackets $\langle$ and $\rangle$ do not specify whether the interval endpoints are closed or open, the join can be formed from either open-closed or closed-open pairings of the adjoining endpoints.

For example, the requirement that the Decoy height must be no less than H after the Decoy has been in the flight Phase for at least X seconds can be specified as follows.

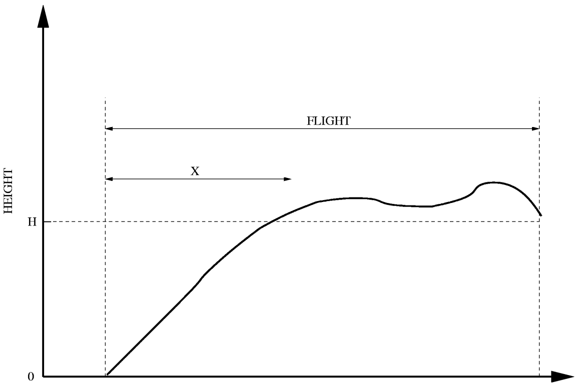

Figure 1: Flight Profile

$$\boxed{\begin{array}{l}\langle \mathit{flight} \wedge \delta > X\mathrm{s} \rangle \subseteq \\ \quad \langle \mathit{flight} \wedge \delta = X\mathrm{s} \rangle ; \langle \mathit{height} \geq H\mathrm{m} \rangle\end{array}}$$

This says that any interval of flight greater than X seconds is comprised of an interval of flight X seconds, where the height is unrestricted, followed by an interval where the height is no less than H metres. While this may seem slightly weaker than required (consider an $X + 1$ second sub-interval at the end of the flight phase - in isolation, this constraint says that during this sub-interval the Decoy need only be above H for the last second), it is sufficient because the constraint on the complete flight interval restricts the possible sub-intervals. That is, the $X + 1$ second sub-interval belongs to a larger $X + N$ second sub-interval where for the last N seconds the Decoy must be above H.

3.4 Differentiable functions

Functions modelling physical quantities generally map from the time domain to some contiguous subset of $\mathbb{R}$. In most cases such functions are differentiable (Fidge, Hayes & Mahony 1998). Derivatives were useful (where they existed) in the PIDS formalisation for modelling the dynamic behaviour of the Decoy.

For example, given the definition of *height* above, the derivative of *height*, denoted by $\overline{\mathit{height}}$, represents the vertical speed of the Decoy. Thus the requirement that the Decoy should continue to rise during the first X seconds of the flight phase before maintaining height H, can be formalised as follows.

$$\boxed{\begin{array}{l}(\langle \neg\, \mathit{flight} \rangle ; \langle \delta > X\mathrm{s} \wedge \mathit{flight} \rangle) \subseteq \\ \quad (\langle \neg\, \mathit{flight} \rangle ; \\ \quad\; \langle \delta = X\mathrm{s} \wedge \mathit{flight} \wedge \overline{\mathit{height}} \geq 0\mathrm{m/s} \rangle ; \\ \quad\; \langle \mathit{height} \geq H\mathrm{m} \rangle)\end{array}}$$

Here, the specification explicitly applies to the start of flight because it is only during the start of flight that the Decoy is required to rise. This specification is satisfied by the Decoy which flies according to the graph displayed in Figure 1. Such graphical representations of traces are a natural and useful way of understanding TIC specifications.

3.5 Idioms

A number of frequently occurring specification idioms have been extracted from the formalised PIDS and are presented in Table 1. These idioms are the basis of a systematic translation that was made from the formalised requirements back to the NL requirements. This was done to acheive some consistency between the form of the NL requirement and the form of the formalised requirement.

$\langle P \rangle = \langle Q \rangle$	P and Q always hold at the same time
$\langle P \rangle = \langle \mathit{true} \rangle$	P always holds
$\langle P \rangle \subseteq \langle Q \rangle$	When P holds on an interval, Q also holds
$\langle P \rangle \subseteq \langle \mathit{true} \rangle ; \langle Q \rangle$	When P holds on an interval, Q holds at the end
$\langle P \rangle ; \langle \neg\, P \rangle \subseteq \langle \neg\, Q \rangle ; \langle Q \rangle$	Immediately after P ceases to hold, Q is established.

Table 1: Specification Idioms

The last idiom may be understood by considering arbitrarily brief intervals of true followed by not true. Note also that intervals are non-empty.

4 Application to the Decoy

In this section some examples of the issues uncovered by formal specification are given.

4.1 Completeness

The initial review of the NL requirements revealed several lifecycle phases of the Decoy (say n phases). These provided a basic structure to the requirements and were the obvious place to start formalisation. Phases were formally represented as boolean variables, which in TIC are modelled as traces from time to boolean.

$$\mathit{phase}_1, \ldots, \mathit{phase}_n : \mathbb{T} \to \mathbb{B} \qquad \texttt{// Decoy phases}$$

However, further formalisation was hindered by the lack of information about the exact definitions of these phases. Various facts were able to be drawn from the requirements: some phases (but not all) were distinct, some phases occurred in order, the lengths of some of the phases were bounded. However no complete definition of the phases was possible. This 'loose specification' is not necessarily a problem. For instance, the timing of these phases may really have been unrestricted. However, other Decoy requirements expressed in terms of the phases were ambiguous because it was unclear when the Decoy went from one phase to another.

In response, the above properties were specified as follows and the issue was raised at the first PIDS workshop.

$$
\begin{array}{ll}
\mathit{xor}(\mathit{phase}_1, \ldots, \mathit{phase}_k) & \texttt{//}k\texttt{ phases are distinct} \\
\langle \mathit{phase}_1 \rangle ; \langle \neg\, \mathit{phase}_1 \rangle \subseteq \langle \neg\, \mathit{phase}_2 \rangle ; \langle \mathit{phase}_2 \rangle & \\
 & \texttt{//phase2 follows phase1} \\
\langle \mathit{phase}_2 \rangle ; \langle \neg\, \mathit{phase}_2 \rangle \subseteq \langle \neg\, \mathit{phase}_3 \rangle ; \langle \mathit{phase}_3 \rangle & \\
 & \texttt{//phase3 follows phase2} \\
\ldots & \\
\langle \mathit{phase}_1 \rangle \subseteq \langle \delta \leq T1\mathrm{s} \rangle & \texttt{//phase1 takes less than T1s} \\
\langle \mathit{phase}_2 \rangle \subseteq \langle \delta \leq T2\mathrm{s} \rangle & \texttt{//phase2 takes less than T2s} \\
\langle \mathit{phase}_3 \rangle \subseteq \langle \delta > T3\mathrm{s} \rangle & \texttt{//phase3 takes no less than T3s}
\end{array}
$$

Upon discussion of this issue in the workshop it became clear that there were various opinions as to the exact definitions of these phases and that such ambiguity had caused confusion when interpreting test results. At this stage a commitment was made to find exact definitions in terms of key Decoy events. After several iterations, some key event times $(E_1, \ldots, E_m)$ were identified. These events are either observable in the Decoy, or events known globally to the system. Identification of these events enabled the following much clearer (and complete) definition of formal requirements.

$$
\begin{array}{ll}
\langle \mathit{phase}_1 \rangle = \langle \alpha \geq E_1 \wedge \omega < E_2 \rangle & \\
 & \texttt{//phase1 goes from } E_1 \texttt{ to } E_2 \\
\langle \mathit{phase}_2 \rangle = \langle \alpha \geq E_2 \wedge \omega < E_3 \rangle & \\
 & \texttt{//phase2 goes from } E_2 \texttt{ to } E_3 \\
\ldots & \\
\langle \mathit{phase}_n \rangle = \langle \alpha \geq E_m \rangle & \texttt{//phase n starts at } E_m
\end{array}
$$

This event based definition also enabled a much simpler NL description. In addition, bounds on the lengths of the phases can be stated in terms of the event times, i.e., $E_2 - E_1 \leq X$s, and several other requirements are also more simply expressed.

As many Decoy requirements are phase related, a complete definition of the phases had a broad impact on the requirements. This example illustrates not only the benefit of formalisation for eliciting ambiguity, but also a follow-on effect: simplicity in the formal model can lead to more easily understood NL requirements. It was easy to verify that the original incomplete phase formulation is consistent with the new event-based reformulation. Thus, the event-based reformulation clarifies the original requirements.

4.2 Consistency and Correctness

No requirements were detected that were in conflict or had incorrect numerical values. While the errors found were sins of ommission rather than commission, minor inconsistencies arose because of the ambiguous specification of the phases mentioned previously. That is, providing a more complete specification solved problems due to inconsistent interpretation.

In addition, some requirements were corrected during the process of reformulation as a result of the involvement of the domain experts. Such changes were evolutionary in nature, yet had not been corrected in the Decoy PIDS. The process of reformulation provided the opportunity to make these corrections and introduce them into the formal specification. While not a direct product of formal analysis, the pursuit of completeness in the formal specification raised the questions which prompted these corrections to be made.

4.3 Feasibility

Absolute requirements with respect to timing, height, or other dynamic properties are infeasible because of the uncontrollable nature of the environment and limitations on the accuracy of sensors at the environment/system interface. Therefore all such requirements should be stated with allowable margins for error. Some absolute requirements were found in the Decoy PIDS and suitable error margins were sought at the workshops. These were then formalised in TIC as follows.

$$
\begin{array}{ll}
\mathit{demanded\_value} : \mathbb{T} \to \mathbb{R} & \texttt{// demanded value} \\
\mathit{actual\_value} : \mathbb{T} \to \mathbb{R} & \texttt{// actual value} \\
 & \\
\langle \mathit{phase}_i \rangle \subseteq & \\
\quad \langle \mathit{actual\_value} = \mathit{demanded\_value} \pm \mathit{margin} \rangle &
\end{array}
$$

That is, during phase i, the actual value shall be within *margin* of the demanded value. The error margin may either be constant or linearly or otherwise related to the demanded value, or it may be an independently evolving value. Formalising error margins in this way provides a basis for their justification in terms of assumptions about the properties of the environment, and the sampling rate/accuracy of the system/environment interface (Hayes 1993). However, this was not attempted in this project.

4.4 Testability

When reviewing the formalised requirements, the ability to validate the requirements was considered. While formal specification does not necessarily prevent one from writing untestable requirements, formalisation assisted in the identification of these requirements because they were generally very difficult to specify. For instance some requirements were ordered in terms of their priority. That is, certain requirements could be relaxed if meeting them meant that others of higher priority would be missed. Expressing this in TIC turned out to be quite difficult and motivated the authors to question the necessity of such a requirement. As it turned out, this was a case of over-specification. A more abstract requirement was identified which captured the over-riding intention. Thus, the detailed implementation in terms of priorities could be removed. This example also illustrates the way in which choosing a suitable level of abstraction can simplify requirement specification.

Requirements which were either not quantifiable or were too difficult to measure were either removed or moved to the *Advice Only* section of the PIDS.

4.5 Modularity

Modularity is a fundamental principle of large scale system development. Modularity allows system designers to control complexity by structuring a system into components with minimal interdependency and maximal cohesion (IEEE 1998). This principle was pursued in the Decoy PIDS in two ways:

- Reordering paragraphs so that like requirements occurred together, maximising the cohesiveness of PIDS sections while also minimising their interdependence.
- Restating requirements in terms of the Decoy frame of reference so that they applied to the Decoy itself rather than to the system in general.

Reordering removed overlapping and repeated requirements while also reducing the likelihood of inconsistency accidentally introduced through modifications of related requirements in different sections.

Phrasing the requirements in terms of objects in the Decoys frame of reference reduced the dependence on other requirements specification documents. This also served to increase the completeness of a requirement by identifying a precise value. An example of this is the identification of the events used to define the phases as described above. Part of the problem had been that some phases had been defined in terms of properties of the system which were not local to the Decoy frame of reference. Therefore, in attempting to precisely define the Decoy phase in terms of a system property, a large portion of the system had to be described in sufficient detail to state the requirement. However, the system property was sufficiently complex that a brief description was necessarily incomplete. Attempted formalisation of the system property could not be completed without formalising other Nulka system specifications. Fortunately, this was avoided because events local to the Decoy were identified that were suitable for completely defining the phases.

5 Physical Characteristics

Physical characteristics that were not time related were formally specified using simple predicate calculus, functions, and relations. This remainder of this section gives some details of their formal specification.

5.1 Componentry

The set of all physical Components is modelled by the set *Component*. Various components were identified.

$[Component]$ `//The set of all components`
$c_1, c_2, c_3, \ldots : Component$

Facts about the Decoy components were modelled by relations or functions. The requirements were then tested by trying to prove simple putative theorems. For instance, the *has_subcomponent* relation described the hierarchical structure of the Decoy components.

$has\_subcomponent : Component \leftrightarrow Component$
`//Decoy structure`
$c_1 \ has\_subcomponent \ c_2$
$\ldots$

A simple putative theorem given this structure is that it is a tree. That is,

- every component is either in the domain or range of this relation;
- the Decoy itself is the only component not in the range; and thirdly,
- no cycles should occur in this relation.

Such theorems were investigated by review.

Several times such analysis identified components that were out of the scope of the Decoy PIDS. This was the case when our putative theorems were too strong. For example, weight is a property of all components and was modelled by a total function.

$weight : Component \rightarrow \mathbb{R}$ `//Component weight`

$weight(c_1) = X\text{kg}$
$\ldots$

Here, a suitable obligation might be to show that every component was given a specified weight. However, it turned out that several requirements specified that the weight of a component should be such that when assembled with the other components, the assembly had the correct weight. That is, there was no specific requirement of the weight of some components, so long as the weight of certain key components were correct. In this case, the non-specific requirements were unnecessary: since the Decoy is already required to meet the specification, these requirements do not contribute any new constraints on the Decoy. As it turned out, several such components were specified by other requirements documents and were duplicated in the Decoy PIDS unnecessarily. Upon discussion with Nulka personnel, it was decided that they were out of the scope of the Decoy PIDS and tagged for removal. This was also done to keep the set of requirements as small as possible, thus simplifying requirement maintenance.

Incomplete	12
Missing	6
Terminology	10
Unnecessary	19

Table 2: Frequency of faults

5.2 Views

The formal specification contains two formal views of the Decoy. The first is a time-evolving dynamic view of Decoys expressed in TIC, where the second is a statement of properties that are true for Decoys, invariant over time, and expressed in simple mathematics. The use of multiple views is recommended where different aspects of a system are more easily expressed in different formalisms or frameworks (Jackson 1995). In such cases, use of a single view often results in an over-complicated specification because disparate aspects of the system have been forced into a common framework, unsuitable for expressing all of the various sorts of properties. The views are linked by the following simple view invariant. Any property P, invariant over time, can be expressed as a TIC property in the following way.

$$\langle P \rangle = \langle \mathit{true} \rangle$$

That is, all intervals have the property P. This approach requires that variables used to model physical characteristics be interpreted as traces.

6 Results

The final stage of the Decoy PIDS reformulation involved detailed analysis of the changes made to the PIDS. This included justifications of the reasons why requirements were changed, deleted, moved, or added. Table 2 summarises the frequency of the faults which lead to changes to the PIDS. The sorts of faults identified are

- *incompleteness*, leading to extra detail being added,
- *missing* requirements, leading to new requirements being added,
- incorrect *terminology*, leading to rephrasing of requirements, and
- *unnecessary* requirements, leading to their deletion.

The distinction between incomplete and missing requirements is that the former arises from ambiguity of the existing requirements whereas the later arises from the complete lack of a requirement.

The figures given in Table 2 result from an examination of the number of requirements modified as reported in the detailed comparison of the initial and final Decoy PIDS documents. A significant amount of reordering also occurred in the Decoy PIDS, and several requirements were both reordered and corrected of one of the given faults. However, a measurement of the amount of reordering has not been attempted.

In total, 120 individual requirements were formally specified. Of these 58 were modified in some way in the final Decoy PIDS (including reordering). Most of the modifications came about through formalisation of the performance requirements rather than the physical characteristics: 43 performance requirements were formalised resulting in 40 of them being modified whereas 77 physical characteristics were formalised resulting in 13 of them being modified (all removals). Besides the fact that the physical characteristics were not the prime focus of the work (no attention was given to static properties during the any of the four workshops) this is an indication of how much easier it is to get static properties correct.

6.1 Sources of faults

Many faults arose as some sort of ambiguity in the original PIDS. The faults were then identified by the process of trying to make the requirements more complete. For instance, ambiguity arose in three ways:

- genuine incompleteness,
- abstraction level too low - resulting in overly detailed but incomplete descriptions,
- inaccurate wording.

When genuine incompleteness arose more detail was added to make the requirement more complete. An inappropriate level of abstraction resulted either in a reduction of scope or the identification of a suitable parameter. In either case, some requirements were then unnecessary and deleted. Inaccurate wording was corrected by suitable rephrasing.

This perspective indicates that searching for incompleteness was the principal mechanism by which faults were identified and the Decoy PIDS improved. This view is widely held (IEEE 1998).

6.2 Effectiveness

The effectiveness of the formalisation is illustrated by the following qualities of the final PIDS.

- Smaller. Although the Decoy PIDS is more complete, it has less requirements largely due to the removal of requirements out of scope of the Decoy.
- Simpler. Being based on simple events and easily defined parameters, the Decoy PIDS is much simpler. This comes about because the modelling of system events has been replaced by references to interface parameters.
- More understandable. The Decoy PIDS can now be reviewed in a workshop taking a single day rather than several days. In addition, the Decoy PIDS was accepted at the final workshop as a baseline for further engineering change.
- Changeable. The new Decoy PIDS is much easier to keep up to date than the previous Decoy PIDS. This is mainly due to its simplicity and because it has been restructured into independent subunits.

7 Tool support

Tools where not used in this project due to time constraints. In the case of TIC, the only known support was provided by the Ergo theorem prover (Cereone 2001). However it was anticipated that the cost of setting up this tool and using it would be too high. In the case of the simple predicates used to model the physical characteristics, several tools would have been suitable, e.g., Sum (Nickson, Utting & Traynor 1996) or CADiZ (Toyn & McDermid 1995). However, with the prime focus of the work being the performance requirements, the benefit was not perceived to be high.

The cost-effectiveness of review based methods is appealing, however some would argue that tool support is required to validate and verify the application of the formalism (Rushby 1995). While the evidence of this article supports the view that formal specification with review provides significant value in itself, it is worth considering how tool support for the Timed Interval Calculus would improve the outcome.

7.1 Type checking

Definedness and type checking tools check that variables are declared before their use and that their use is consistent with their declared types. They may also check that every declared variable is used at least once. Such tools eliminate the most common inconsistencies found in a formal specification. However, in the case of TIC, most variables are declared to be of type $\mathbb{T} \to \mathbb{R}$, providing a fairly flat and homogeneous domain of objects when compared to languages exhibiting complex type structures such as ML and other functional languages. The lack of type structure diminishes the benefit of type checking. This situation is ameliorated to some extent by the use of dimensions such as *seconds*, *metres*, *metres/second* on physical quantities. This solution (Hayes & Mahony 1995), commonplace in the physical sciences, extends the reals with more type structure. The author is not aware of any automatic tool support for dimensions in any formal specification languages. However, within the presence of suitable type definitions and functions, it can be implemented with a minor extension to the type system of Z (Hayes & Mahony 1995).

7.2 Animation

The next level of tool support worth considering is animation. This is quite useful where external fidelity (checking that the specifications say what is intended) is of interest (Rushby 1995). External fidelity is most commonly checked by informal review by domain experts. However, this may require a tabular or diagrammatic form of specification where the domain experts are not familiar with mathematical specification languages. Animation of formal specifications allows the dynamic behaviour of the system being specified to be reviewed. Animation tools such as Possum (Hazel, Strooper & Traynor 1997) allow graphical representations to be used to interpret the states being generated by the tool. However, the author is not aware of any animation tools for TIC or other related interval-based specification languages. Some mechanism for drawing graphs such as that presented in Figure 1 would be worth pursuing.

7.3 Theorem provers

A variety of techniques are available for validating requirements where both external fidelity and completeness are key attributes of interest (Rushby 1995).

External fidelity may be investigated by:

- challenging the specification with a question/theorem it should be able to answer, or
- showing that the model describes simple desired situations by putative interpretation.

Completeness may be investigated by:

- proving that the disjunction of all cases is a tautology, or
- proving that the environmental assumptions and the requirements specification imply the required system constraints (Rushby 1995, pg 157).

In the formalisation of the Decoy PIDS, some of the above techniques were applied in an informal and *ad hoc* manner. Typically theorem provers provide no explicit support for performing the listed techniques, however, a sufficiently powerful theorem prover such as Isabelle (Paulson, Nipkov & Wenzel 2002) or PVS (Shankar, Owre, Rushby & Stringer-Calvert 2001) may allow them to be proved quickly. However, such provers typically require a significant amount of assistance from the user.

8 Related work

Interest in applying formal specification in real-world systems has increased in the systems and software engineering literature (Palshikar 2001, Lamsweerde 2000, Berry 2002). Some suggest that slow industrial acceptance of formal methods is due to the perceived difficulty of learning mathematical notations; costs of training; lack of tools, standards, and case studies. These problems are being addressed, for instance, numerous guidelines for the use of formal specification have been presented (Palshikar 2001, Bowen & Hinchey 1995) and case studies also appear describing how formal methods can be applied to requirements modelling on projects such as the International Space Station (Easterbrook, Lutz, Covington, Kelly, Ampo & Hamilton 1998) for example.

Tool development has become the focus of much formal methods research in general (Abrial 1996, Nickson et al. 1996). Support for DC has been implemented in PVS (Skakkebaek & Shankar 1994). DC has also been integrated with other formalisms such as SCR (Software Cost Reduction) (Heitmeyer, Kirby, Labaw & Bharadwaj 1998) (a formal method involving tabular presentations of function specifications) for the specification of the requirements of hybrid systems (Chaochen, Ravn & Hansen 1993).

8.1 Costs

The cost of applying formal methods is difficult to assess. Rules of thumb have been presented elsewhere (Berry 2002) which estimate that the use of formal specification without formal verification may drive up the cost of a project as much as twofold, whereas the use of formal verification may drive up the cost as much as tenfold. This is in agreement with our experience of applying formal specification to the Decoy PIDS; while we only applied formal specification to one of many requirements documents within a large project, formal verification would have increased the costs of our work dramatically.

8.2 Benefits

Similar results to ours are reported from a project which employed formal specification on subsystems of NASA's space shuttle software (Crow & Di Vito 1998). They conclude that formal specification confers benefits regardless of how extensively they are applied, and formal methods are most effective when they are judiciously tailored to the application. They emphasise the following benefits of formal specification by itself.

- Clarifies requirements: the concise and unambiguous formal statement of a requirement can cut through lengthy informal statements to expose fundamental issues that can be expressed simply.
- Articulates Implicit Assumptions: undocumented assumptions about inputs and state variables are made explicit when defined in the formal notation.

- Exposes flaws: NASA's project exposed a significant number of flaws in the Shuttle subsystem requirements.

Here, formalisation had a similar effect on the Decoy PIDS. For instance the description of the phases was dramatically simplified when events were chosen to complete their description in the formalism. Also, the implicit assumptions about margins for error on demanded values were exposed by pursuing feasible statements of the requirements. Several flaws were also exposed by formalisation.

8.3 Probabilistic requirements

Probabilistic requirements were not formalised in this project due to the lack of a sufficiently well understood formalism. However, much work exists in this area. For instance, the Probabilistic Duration Calculus (PDC) (Hung & Chaochen 1999) for dealing with dependability requirements (that the probability for undesirable but unavoidable behaviour of a system be below a certain limit) uses probabilistic automata to model imperfect implementations and then defines the probability that a system satisfies a DC formula. Should this approach be applicable to TIC, it is likely to provide a semantic basis for formalising the dependability requirements of the Decoy.

9 Conclusion

This article presents an overview of TIC and gives examples of the formalism in application. It also explains how formalisation motivated specific changes to the requirements. The improved readability and structure of the revised PIDS, combined with the minimal cost gained by using formal specification with review, suggest that the Decoy PIDS reformulation project was cost-effective. Opportunities for tool support are discussed and a brief survey of related work is presented.

Acknowledgements

The author gratefully acknowledges comments and suggestions received from Graeme Smith and Peter Lindsay (SVRC), Chris Edwards (Defence), Mal Crozier (Prime Contractor), Michal Severa (Defence), Tony Cant (DSTO), and the anonymous referees. This work has been partially funded by the Directorate of Systems Engineering and Software Acquisition Management, Australian Defence Force as part of the DefSafe project (`www.itee.uq.edu.au/~defsafe`) investigating improved methods for specifying and acquiring Safety Critical Systems and Australian Research Council Large Grant A00104650, Verified Compilation Strategies for Critical Computer Programs.

References

Abrial, J.-R. (1996), *The B Book: Assigning Programs to Meanings*, Tracts in Theoretical Computer Science, Cambridge University Press.

Berry, D. M. (2002), 'Formal methods: the very idea, some thoughts about why they work when they work', *Science of Computer Programming* **42**(1), 11–27.

Bowen, J. P. & Hinchey, M. G. (1995), 'Ten commandments of formal methods', *IEEE Computer* **28**(4), 56–63.

Cereone, A. (2001), Axiomatisation of an interval calculus for theorem proving, *in* 'Computing: The Australasian Theory Symposium, (CATS 2001)'.

Chaochen, Z., Hoare, C. A. R. & Ravn, A. P. (1991), 'A calculus of durations', *Information Processing Letters* **40**(5), 269–276.

Chaochen, Z., Ravn, A. P. & Hansen, M. R. (1993), An extended duration calculaus for hybrid real-time systems, *in* R. L. Grossman, A. Nerode, A. P. Ravnand & H. Rischel, eds, 'Hybrid Systems', Vol. 736 of *LNCS*, Springer-Verlag.

Crow, J. & Di Vito, B. (1998), 'Formalizing space shuttle software requirements: Four case studies', *ACM Transactions on Software Engineering and Methodology* **7**(3), 296–332.

Department of Defense (1985), *MIL-STD-490A, Specification Practices.*

Easterbrook, S., Lutz, R. R., Covington, R., Kelly, J., Ampo, Y. & Hamilton, D. (1998), 'Experiences using lightweight formal methods for requirements modeling', *Software Engineering* **24**(1), 4–14.

Fidge, C. J., Hayes, I. J. & Mahony, B. P. (1998), Defining differentiation and integration in Z, Technical Report 98-09, Software Verification Research Centre, School of Information Technology, The University of Queensland, Brisbane 4072, Australia.

Fidge, C. J., Hayes, I. J., Martin, A. P. & Wabenhorst, A. K. (1998), A set theoretic model for real-time specification and reasoning, *in* 'Mathematics of Program Construction (MPC 98)', Vol. 1422 of *LNCS*, Springer-Verlag, pp. 188–206.

Hayes, I. J. (1993), Specifying physical limitations: A case study of an oscilloscope, Technical Report 167, Department of Computer Science, University of Queensland, Australia, 4072.

Hayes, I. & Mahony, B. (1995), 'Using units of measurement in formal specifications', *Formal Aspects of Computing* **7**(3), 329–347. Also appears as SVRC Technical Report 94-31.

Hazel, D., Strooper, P. & Traynor, O. (1997), Possum: An animator for the Sum specification language, *in* 'Proceedings of the Asia-Pacific Software Engineering Conference and International Computer Science Conference'.

Heitmeyer, C. L., Kirby, J., Labaw, B. G. & Bharadwaj, R. (1998), SCR*: A toolset for specifying and analyzing software requirements, *in* 'Computer Aided Verification (CAV'98)'.

Hung, D. V. & Chaochen, Z. (1999), 'Probabilistic duration calculus for continuous time', *Formal Aspects of Computing* **11**, 21–44.

IEEE (1998), 'Guide for Developing System Requirements Specifications', IEEE Std 1233.

Jackson, D. (1995), 'Structuring Z specifications with Views', *ACM Transactions on Software Engineering and Methodology* **4**(4), 365–389.

Lamsweerde, A. (2000), Formal specification: a roadmap, *in* A. Finkelstein, ed., 'International Conference on Software Engineering (ICSE): The Future of Software Engineering', ACM Press, pp. 147–159.

Nickson, R., Utting, M. & Traynor, O. (1996), Cogito Ergo Sum - Providing structured theorem prover support for specification formalisms, *in* 'In Kotagiri Ramamohanarao, editor, Proceedings of the Nineteenth Australasian Computer Science Conference (ACSC'96)', Vol. 18.

Palshikar, G. K. (2001), 'Applying formal specification to real-world software development', *IEEE Software* **18**(6), 89–97.

Paulson, L. C., Nipkov, T. & Wenzel, M. (2002), *The Isabelle reference manual*, http://www.cl.cam.ac.uk/Research/HVG/Isabelle/dist/Isabelle2002/doc/ref.pdf.

Rushby, J. (1995), Formal methods and their role in digital systems validation for airborne systems, Technical Report NASA Contractor Report 4673, Prepared for Langley Research Center, NASA.

Shankar, N., Owre, S., Rushby, J. & Stringer-Calvert, D. W. J. (2001), *PVS Prover Guide*, http://pvs.csl.sri.com/doc/pvs-prover-guide.ps.gz.

Skakkebaek, J. U. & Shankar, N. (1994), Towards a duration calculus proof assistant in PVS, *in* 'Formal Techniques in Real-time and Fault-Tolerant Systems', Vol. 863 of *LNCS*, Springer-Verlag.

Toyn, I. & McDermid, J. A. (1995), 'CADiZ: An architecture for Z tools and its implementation', *Software - Practice and Experience* **25**(3), 305–330.