A Requirements Capture Method and its use in an Air Traffic Control Application *

T. L. McCluskey (`lee@zeus.hud.ac.uk`) **and J. M. Porteous** (`julie@zeus.hud.ac.uk`)

School of Computing and Mathematics,
The University of Huddersfield,
Huddersfield,
West Yorks HD1 3DH

Y. Naik (`yogesh@cs.city.ac.uk`) **and C. N. Taylor** (`christa@cs.city.ac.uk`)

Department of Computer Science,
The City University,
Northampton Square,
London EC1V OHB

S. Jones (`S.Jones@hertfordshire.ac.uk`)

Department of Computer Science,
University of Hertfordshire,
College Lane,
Hertfordshire

Keywords

requirements capture, formal specification, knowledge representation

*This research was sponsored by the NATS division of The Civil Aviation Authority

Summary

This paper describes our experience in capturing, using a formal specification language, a model of the knowledge-intensive domain of oceanic air traffic control. This model is intended to form part of the requirements specification for a decision support system for air traffic controllers. We give an overview of the methods we used in analysing the scope of the domain, choosing an appropriate formalism, developing a domain model, and validating the model in various ways. Central to the method was the development of a formal requirements engineering environment which provided automated tools for model validation and maintenance.

Introduction

The problems inherent in Software Engineering, especially those relating to the production of safety-critical systems, are nowadays all too apparent. Brown et al (in [1]), point out a number of these problems. We group them here into two classes:

- The problems of understanding the customer's initial requirements, and maintaining them as they change.
- The problems inherent in the complexity, malleability and invisibility of software itself. That is, as well as being deceptively complicated, software is easily changed, and is not a physical artifact unlike the output of other engineering processes.

In this paper we present an overview of a method for formally capturing requirements for knowledge-intensive systems that demand high integrity in their construction. We illustrate the method by its application in the development of a prototype decision support tool for air traffic control, involving separation of aircraft in the airspace over the North-East Atlantic. Using the method for this particular type of project, we believe that the problems in Software Engineering stated above can be addressed effectively.

This work provokes issues that span both software engineering and knowledge based systems, including knowledge capture, formalisation, correctness, validation and automated reasoning. The benefits of formally specifying requirements, to do with precision, removal of ambiguity, automated manipulation and so on, have been well argued with the appearance of appropriate formal languages (e.g. MAL [2]). Up to now, few real industrial applications have been reported, and it seems often to be the case that researchers concentrate on particular kinds of applications in order to promote a particular formalism. Our approach to requirements specification is based on a formalism-independent method, where the choice of formalism and appropriate engineering environment is a feature of that method. In outline the method encompasses the following steps:

- Scoping and Domain Analysis: determining the size and nature of the domain;

- Formalism Choice and Customisation: selecting and customising a language and environment for domain capture;

- Domain Model Capture: eliciting knowledge and capturing a model of the domain using the chosen formalism;

- Diverse Validation: a five point validation plan that includes dynamic testing, hand validation, and static analysis by formal reasoning.

The particular significance of this work lies in:

- the use of an expressive *formal specification language* (Many-Sorted First Order Logic) to capture part of the requirements specification for a real application;

- the construction and customisation of a formal requirements engineering environment (which is given the acronym "FREE" throughout the paper) that was used as a framework for the capture and validation of the model.

As well as carrying out all forms of syntactic checks, and allowing reasoning about the behaviour of the model to be carried out, the FREE translates the requirements model into:

- a "hand validation" form, for examination by domain experts who may be unfamiliar with formal logical notation;

- a prototype, for use within

  - a test harness to perform dynamic testing of the requirements, and

  - a simulator to allow users access to an animated version of the requirements, allowing "hands-on" validation by domain experts.

This approach addresses the two main problems mentioned at the start of the introduction. The software engineer's idea of what the users or customers want is precisely captured in a validated, maintainable formal model. It may be that the simulator can be further developed to satisfy the full user requirement (for example it could be optimised to satisfy response-time requirements). In this case the problems to do with software development are reduced to those encountered in the development of the simulator, and in constructing, acquiring and customising the tools that make up the model's environment. At the very least, this approach aims to deliver a well-validated formal specification with which to contract software developers, as well as a simulator with which to dynamically check final software.

In the project described here, we were concerned with the specification of requirements for software which would implement rules used in oceanic air traffic control, as we will explain below. Throughout the paper, we use the term "application domain" (or "domain") to mean that part of reality with which we are concerned, and the term "requirements model" (or "model") to refer to the specification of the domain. While a complete specification of requirements for the software would include definitions of "non-functional" requirements, such as those relating to its performance or user interface, the aim of our project was simply to capture "functional" requirements relating to the implementation of rules for aircraft separation. The part of the requirements model (or requirements "specification") we are concerned with here is thus equivalent to the model of the domain which we aim to capture.

An Overview of the FAROAS project

The formal requirements method was developed within the context of the FAROAS project, a research project funded by The Civil Aviation Authority. The general area of interest of the work was the separation of aircraft in oceanic airspace.

Air traffic in the airspace over the north eastern part of the Atlantic – The "Shanwick Oceanic Control Area" (Shanwick OCA) – must be separated in accordance with minima laid down by the International Civil Aviation Organisation. The separation distance that is applicable in any given situation depends on a large number of factors including the type of an aircraft, its navigational accuracy and whether it is climbing or descending. A structured, natural language description of these separation standards is contained in the Manual of Air Traffic Services, Part 2 (MATS-2) [3]. It is the responsibility of air traffic control officers to ensure that all aircraft within Shanwick OCA are separated by at least the required minima through the processes of *conflict prediction* and *conflict resolution.* Conflict prediction is the process of detecting potential separation violations by comparing the projected flight paths (flight profiles) of pairs of aircraft. Conflict resolution is the process of planning new conflict free flight profiles.

The air traffic control officers use an automated Flight Data Processing System (FDPS), a key component of which is the "conflict software", that provides assistance for the processes of conflict prediction and resolution. A new FDPS is currently being developed and we became involved with the capture of the requirements for the conflict software in the new FDPS. The long term goal to which the work of the FAROAS project contributed was to develop a formal specification of the requirements for conflict in oceanic airspace. The aim was to formalise and make complete the requirements of the Shanwick OCA aircraft separation standards with respect to the specific function of predicting and explaining separation violations (i.e. conflicts) in aircraft flight plans, in such a way that those requirements can be rigorously validated and maintained. Ultimately the formalisation might serve as an independent standard for the procurement of conflict software systems. Hence the role of this document would be comparable to the MATS-2 in operational ATC. Within this context the objectives of the FAROAS project were:

- to identify a formalism for requirements capture that could be validated by ATC experts
- to formally capture the functional requirements of conflict prediction
- to establish a method for validating (and re-validating when necessary) the formally

captured requirements.

The project commenced in May 1992 and was based at The City University in London. For the duration of the project, the project team consisted of of two full time research fellows, a project manager, a project consultant, a quality manager and a project manager from The Civil Aviation Authority. Also, a research fellow worked for 3 months on the use of automated proof assistants for maintaining the requirements specification. From the outset, project plans detailed the major deliverables and milestones against which the progress of the research was to be judged. Also implemented from the start of the project was a detailed quality plan which set down procedures for formal review of all project deliverables (reports, specifications and software). The project terminated successfully at the end of 1993 upon delivery of the formally captured requirements model which had been validated to the satisfaction of the client and sufficiently to demonstrate the method.

The Scope and Nature of the Domain

Criteria used in the domain analysis

The success of requirements capture depends greatly on establishing a clear scope for the project, and on a good choice of formalism. The scope of our application, which we will refer to as the *oceanic ATC domain*, was confined to conflict prediction, although we wished to keep open the possibility of adding a conflict resolution component at a future stage.

Once the scope of a project has been established, the domain should be analysed to identify

- the key sources of domain knowledge
- the groupings and the nature of various types of knowledge
- the size of the model to be constructed, and the aspects (e.g. time, agents) that need to be represented explicitly

In formal requirements capture, we have to deal with the vague concepts of "knowledge" and "knowledge representation". Since the field of knowledge representation has not yet arrived at generally agreed standard criteria or measures, the points above will instead be illustrated by example.

The key sources of knowledge

The key sources of knowledge will affect the *planning* of domain capture, for example if the major source is experts rather than documents, the setting up of interviews and document reviews will tend to lengthen the project and increase its cost. These key sources also play a crucial role in validation, and so are as important towards the end of a project as they are during the elicitation phase. For example, comparing the behaviour of a prototype that has been automatically generated from a requirements specification against the customer's existing software, is far easier than assembling groups of experts together to "hand validate" a set of formalised rules.

In the *oceanic ATC domain* the main knowledge source was:

- documentation – principally the separation standards encapsulated in the MATS-2 [3]. Additional sources were the design documentation of the current conflict prediction software [4, 5]

Additional knowledge sources were:

- people – principally air traffic control officers

- software – the current conflict prediction software

The groupings and the nature of various types of knowledge

Unless there is some strong reason to the contrary, such as the need for a *new* technical solution, then the captured model should reflect the existing groupings of knowledge. Each grouping should be analysed, to determine the types of knowledge it contains and its likely interface with other groupings.

The *oceanic ATC domain* contains two broad types of knowledge:

- *rule-based*: This includes a set of rules in natural language within the MATS-2 document that detail the minimum separation that must be maintained between aircraft in oceanic airspace. Also, there are rules that define a method of determining if two flight profiles will violate the minimum separation values that have been derived from the separation rules, in other words to detect potential conflicts.

  A simple rule defining the scope of a *segment* (a section of a flight profile) is paraphrased below:

  > "If a profile containing a segment is wholly or partly in Shanwick OCA then a segment of such a profile starts at or after the first recognised point for oceanic conflict prediction if and only if the entry time of that segment is at or later than the time of the first recognised 4D point for oceanic conflict prediction of the profile containing that segment."

- *object-based*: the objects, relationships and properties that underlie the rules. For example, “segment” is an important kind of object used in many of the rules. The rule above contains the following functional relationships:

“the entry time of that segment”
“the profile containing that segment”

The size of the model to be constructed

It is important to try to quantify the likely size of the model so that the project can be planned accordingly. This presumably extends work carried out during the feasibility study. One crude measure of model size is simply the number of axioms or rules that are predicted to appear in it.

For the FAROAS project, model size was estimated by considering the:

- textual size of the main knowledge sources, in terms of relevant assertions about the *oceanic ATC domain*; and the
- number and complexity of discernible types of objects in the domain.

In choosing a metric for size we had to assume a particular formalism (possible metrics include number of equations, operations, rules or axioms predicted to be in the model). Using these observations, and choosing a “number of axioms” metric, we examined the available documentation and correctly estimated the size as being in the region of several hundred axioms.

The nature of the explicit knowledge

Another important question regarding the model is: which aspects of the domain will require explicit representation? Some general aspects that might be considered here include:

- *agents*
- *time*
- *states of knowledge and belief*
- *actions*
- *probabilities*

- *permissions* and *obligations*

To some extent these aspects are independent of each other: some formalisms explicitly represent agents and belief, but none of the other aspects listed above, some explicitly represent actions, but none of the others and so on. Once it has been decided that a particular aspect needs to be represented explicitly, further questions arise as to how this should be done. For example, in the case of time, the model of time adopted may be either discrete or continuous; it may be bounded or unbounded (in either past or future directions); it may involve time points or time intervals (or both); and from a syntactic point of view, it may involve the use of modal operators, or alternatively the use of terms denoting times

The real domain of air traffic control clearly involves time and agents (e.g. pilots and air traffic controllers), who have beliefs and who perform actions (e.g. issuing clearances). However, since the focus of the FAROAS project was the rules governing oceanic aircraft separation and conflict prediction, we concluded that of the general aspects listed earlier, only time need be represented explicitly. The chosen model of time was discrete, and unbounded in the future. It was felt that times should be represented syntactically by explicit temporal terms, relative to a nominal zero time. The term "22 15 GMT day 2" is a typical example of such terms: the day number here is relative to an arbitrary "day 0", since there is no need to refer to the actual calendar date as far as the separation rules are concerned.

Formalism Choice and Domain Capture

Criteria for Formalism Choice

Once the scope, size and nature of the domain have been determined, the most important initial aspect of formal requirements capture is the *choice* of formalism. To some extent this will depend upon the experience and prejudices of the modellers, but there are also more objective criteria. In a similar study to this one (described briefly in [6]), an Airborne Collision Avoidance System (ACAS) was captured using the LOTOS specification language [7]. Describing the capture, Sowerbutts states that the main reason for the choice of LOTOS is the "natural mapping from ACAS onto LOTOS's processes". In other words LOTOS was chosen on the basis of how well it fitted the domain. Another factor was the availability of *tool support* for LOTOS (in particular an interpreter).

Considerations such as these lead us to the following criteria for evaluating candidate formalisms, the first being the most important:

- *natural fit* : does the formalism fit the domain? Does it allow domain knowledge to be represented at an appropriate level of abstraction, so that the model can mirror the domain? This question is analogous to consideration of the "semantic gap" in programming language selection: the narrower the gap between language features and application, the more natural the selection of that language is said to be. The chief advantage of a natural fit is that it eases knowledge elicitation and validation. For example, a model that mirrors the domain can be directly viewed by domain experts (as was intended in the FAROAS project) and facilitates the construction of a tool for translating parts of the specification into a *validation form* that can be easily read by the domain experts.

- *support environment* : are practical tools available for the formalism? The kinds of tools required for formal requirements capture are type checkers, parsers, translators, interpreters, proof checkers and inferencing mechanisms. These tools should be available in an integrated environment that can be customised for a particular project.

- *maintenance* : if the domain is subject to change, can the model be easily and consistently updated to reflect this? This question may in fact be combined with the one above - for example, are tools available to easily re-generate proof obligations to ensure model consistency?

- *expressiveness* and *extendibility* : will the formalism restrict natural expression in any way? If the scope or depth of requirements of the domain are increased, can the formalism be likewise extended?

- *formality* : does the language have a firm mathematical basis, where its meaning is clearly independent of and not tied to a program or interpreter? Is it possible to reason with the formalism in a precise and straightforward way, ideally with a tool such as a *proof assistant*?

- *experience* and *training* : are project staff initially familiar with the formalism, or will they require a period of learning or even formal training before they can use it effectively? If the latter is the case, considerable delay and additional cost may result. The same applies to *future* staff, who may have to maintain the system after the initial project team have left. A formalism that is uncommon or difficult to learn will thus introduce extra delays and costs throughout the whole life-cycle of the system.

Connecting all these criteria is the dominant issue of *validation*. The requirements model will have an interpretation that makes it a "model" of something real, making it analogous to an inductive scientific theory. Like a scientific theory, it cannot be formally proved absolutely correct or complete. However, its quality can be promoted by systematic validation, relying on diverse and largely automated validation processes. The link may be indirect, but a formalism that helps the validation process is also desirable.

Evaluation of Candidate Language Groups

At the start of the project we performed a survey and feasibility study into the use of various likely languages groupings [8], and evaluated them according to the criteria above. Here we will summarise that evaluation, taking each group in term.

Firstly, there are those languages traditionally used for knowledge representation in the areas of artificial intelligence and knowledged-based systems. These include rich, highly expressive languages such as frame-based representations [9], input languages for expert system shells [10] and variants of formal logic [11]. As early as the mid-1970's frame-based languages, which inspired the 1980's boom in object-oriented languages and methods, allowed one to capture knowledge as a collection of related objects, each object having internal structure comprising slots (or attributes) and procedural attachments [9]. This slant towards rich, very high level languages seems to be at the expense of semantic rigour. Frame-like representations and expert system shell languages can be criticised for having a meaning which lacks a formal mathematical basis, and is too tied to an interpreter or an implementation. Hence, whereas the machine-independent languages of logic are possible candidates, in general the languages in this group do not score well on the *formality* criterion.

We might also have considered using a specialised requirements specification language, such as RML [12] or MAL [2]. While languages such as these are being developed specifically for use in capturing and modelling the requirements for certain kinds of system, they were judged not to be appropriate for use in our project. Languages such as MAL are more expressive and more complex than was judged necessary: for example, although Structured MAL provides the engineer with the ability to specify agents and obligations in the domain of interest, we had already decided that the FAROAS project would not need to be concerned with the explicit modelling of such phenomena. Other languages, such as RML, have been targeted at use in developing information systems and therefore embody concerns different to those involved in the development of technical decision support systems. Still other requirements specification languages, such as that under investigation in the KAOS project [13] were judged to be at too early a stage in development to be used in specifying a real application. Support tools for such languages were also not readily available.

Other potential candidates were the established formal specification languages, which fall into two broad families. The first are those languages based on *equational algebra*, for example OBJ 3 [14], AXIS [15], and LOTOS [7]. A specification written in OBJ 3 is typically formed in a hierarchical structure of algebraic specifications of abstract data types. Specifications thus have an abstract, object-oriented flavour, supporting polymorphism and encapsulation, and their equational basis allows specifications to be prototyped using a re-write rule interpretation [16]. While it would be possible to build up definitions of the objects in the ATC domain in this way, the bulk of the domain (requiring a rich logical form) could not be represented naturally using equational expressions. In other words, the semantic gap between the rule based ATC knowledge and an equational-based specification language was too wide.

The *model-based* formal specification languages (chiefly VDM-SL [17] and Z [18]) are based on first order logic and set theory, and have the advantages of a growing user-base and tool support including parsers and type checkers (e.g. fUZZ for Z [19]). Specifications written in VDM-SL typically contain a mathematical model of a state involving composites of sets, sequences and mappings, as well as a collection of operations that specify state changes using pre- and post-conditions.

We initially used a model-based notation to represent some of the objects in the *oceanic ATC domain* and some of the functions on those objects. In an early project report on the domain analysis, flight profiles were represented as the following set [20] (the reader not familiar with this notation may safely ignore the details):

$$
\begin{aligned}
&\mathit{Flight\_profiles} = \{(a, f) : a \in \mathit{Aircraft} \wedge \\
&\quad f \in \mathrm{seq}[(\mathit{Flight\_positions} \times \mathit{Times}) \times \mathit{Aircraft\_speeds}] \wedge \\
&\quad ((((p_1, t_1), s_1), (p_2, t_2), s_2)) \in \mathit{adjacent}(f) \Rightarrow t_1 < t_2) \wedge \\
&\quad (\forall\, s \in \mathit{ran}(\mathit{ran}(f)).s \leq \mathit{max\_speed}(\mathit{type}(a))) \wedge \\
&\quad (\forall\, s \in \mathit{ran}(\mathit{dom}(\mathit{dom}(\mathit{ran}(f)))).s \leq \mathit{flight\_ceiling}(\mathit{type}(a))) \wedge \mathit{length}(f) \leq 2\}
\end{aligned}
$$

Our idea of an adequate representation for flight profiles incrementally shifted as our understanding deepened, however, so that effort creating (and typesetting) this initial definition had been largely wasted. We soon realised that at this early stage any commitment to a model of the objects in the domain was premature. Rather, we needed to construct the requirements model using a *loose* specification, one that allows us to make the *least commitment* to the structure and behaviour of the model (as explained on page 19 of reference [21]). When capturing requirements one does not have a deep enough knowledge of the domain to commit to a particular representation using abstract mathematical building blocks typified by the set. If one creates an inappropriate partial model in this form then throwing away the initial model and creating a new one wastes effort. In addition, our initial domain analysis reports containing set-based formulae, such as that shown above, were off putting to our client.

One final point against the use of a model-based formalism for this project is to do with their promotion of specifications using an abstract state. Our application could not easily be given an interpretation that involved operations on a state, as the bulk of the data represented (monotonic) knowledge which would be used to come to a binary decision about aircraft separation. Hence we would be unlikely to put these notations to full use.

These deliberations lead us to concentrate on more abstract languages based wholly on formal logic. A result of the domain analysis was that we did not need to represent uncertainties, beliefs, actions etc, which indicated that a straightforward first-order logic would be adequate, easing the problems of staff training and tool support. Also, the major part of the knowledge we were to capture was written used a logical phrasing (as the example paragraph on page 5 suggests) which could be captured at a natural level of abstraction by first order logic. To deal with objects in the application, classical logic can be enriched with *sorts* [22] defining classes of objects which share the same properties. Encapsulating primitive axioms in a sort definition also gives a natural structure to the specification, as explained in the next section.

The Domain Capture Formalism

Having decided on the type of formal language, a final decision was required between using an "off the shelf" formalism and customising our own. In the event we chose a formalism in the latter category, a customised version of Many-Sorted First Order Logic [22] which we refer to in the remainder of the paper as MSFOL. A strong candidate in the former category appeared to be Z [18], an alternative we will discuss in retrospect after an exposition of the use of MSFOL.

We defined a version of MSFOL to have a simple structure of disjoint sorts, with rigidly sort-restricted functions and predicates — with the sole exception of numerical sorts, predicates and functions. In this case, subsorts were allowed so that numerical operators could be overloaded in the usual way. For example, the symbol “$<$” could be used to compare two terms that were both of type natural numbers, integers or reals.

Atomic wffs were composed of mix-fix predicates and functions, allowing expressions to be written with maximum readability, for example:

```
(Segment  starts_at_or_after_first_recognised_pt_for_oceanic_cpr)
```

is an atomic wff, where Segment is a sort variable followed by a long but descriptive predicate name.

The syntax was defined using a “determinate clause grammar”, expressed in the Prolog grammar rule notation [23]. Such a grammar has a dual interpretation as a specification and a program, and hence doubles as a parser. This formed the “front-end” to the translation tools which were the central processes in the FREE, the requirements engineering environment which we constructed during the course of the project.

The Structure of the Conflict Prediction Specification

The model we constructed captured the functional requirements of the conflict prediction process within the *oceanic ATC domain*, and so in what follows we shall call it the “Conflict Prediction Specification” (the CPS). It should be clear from this section that parts of the model however, in particular the separation rules, can be re-used for other applications such as a specification of conflict resolution.

Many of the axioms in the CPS were non-recursive definitions of predicates or functions in terms of lower-level predicates and functions. For example, the rule stated in English on page 5 was represented by the following axiom:

```
(the_Profile_containing(Segment)  is_wholly_or_partly_in_shanwick_oca)
    =>
        [(Segment  starts_at_or_after_first_recognised_pt_for_oceanic_cpr)
        <=>
        (the_entry_Time_of(Segment)  is_at_or_later_than
            the_Time_of(the_first_recognised_4D_pt_for_oceanic_cpr_of(
                the_Profile_containing(Segment))))
        ]
```

This axiom amounts to a conditional definition (applicable only to segments belonging to profiles that are wholly or partly in the Shanwick OCA) of the predicate:

```
(Segment  starts_at_or_after_first_recognised_pt_for_oceanic_cpr)
```

in terms of the functions:

$the\_Profile\_containing(Segment)$
$the\_entry\_Time\_of(Segment)$
$the\_Time\_of(4D\_pt)$
$the\_first\_recognised\_4D\_pt\_for\_oceanic\_cpr\_of(Profile)$

and the predicates:

$(Profile\ \ is\_wholly\_or\_partly\_in\_shanwick\_oca)$
$(Time1\ \ is\_at\_or\_later\_thanTime2)$

The structure of the specification reflected the hierarchical structure of the conflict prediction domain, shown in figure 1. At the top-level are axioms specifically capturing the conflict prediction method, which involves pairwise comparisons of segments. For example, a recursive axiom (which we will refer to as the "box conflict axiom") describing the conditions under which conflict is said to exist within a time interval modelled as a set of discrete points, is as follows:

$$
\begin{array}{l}
[(Segment1\ \ and\ \ Segment2\ \ are\_subject\_to\_oceanic\_cpr) \\
\& \\
(Time1\ \ is\_in\_overlap\_time\_window\_for\ \ Segment1\ \ and\ \ Segment2) \\
\& \\
(Time2\ \ is\_in\_overlap\_time\_window\_for\ \ Segment1\ \ and\ \ Segment2) \\
\& \\
(Time2\ \ is\_at\_or\_later\_than\ \ Time1)] \\
=> \\
[(box\_conflict\_exists\_between\_linear\_tracks\_of\ \ Segment1\ \ and\ \ Segment2 \\
\quad at\_some\_time\_at\_or\_between \\
\quad\quad Time1\ \ and\ \ Time2) \\
\quad <=> \\
\quad [(box\_conflict\_exists\_between\_linear\_tracks\_of \\
\quad\quad Segment1\ \ and\ \ Segment2\ \ at\ \ Time1) \\
\quad or \\
\quad (box\_conflict\_exists\_between\_linear\_tracks\_of\ \ Segment1\ \ and\ \ Segment2 \\
\quad\quad at\_some\_time\_at\_or\_between \\
the\_next\_integer\_Time\_in\_mins\_after(Time1)\ \ and\ \ Time2) \\
\quad] \\
].
\end{array}
$$

The separation values for segments of a profile are captured by the "Separation Value Axioms". An example separation rule from the specification is as follows:

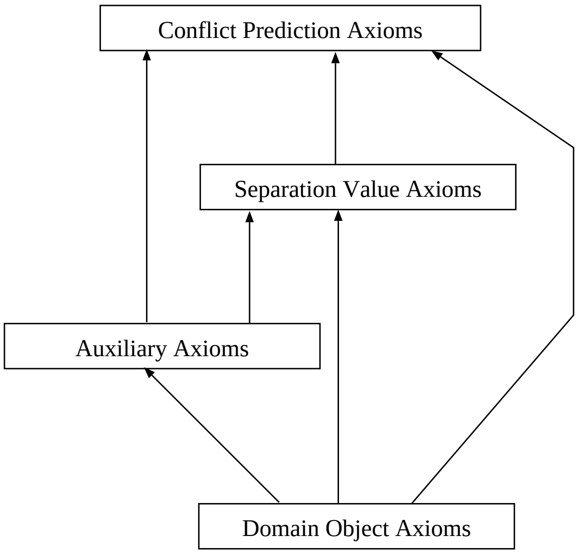

Figure 1: The Structure of the Specification

$$
\begin{array}{l}
[(\mathit{Segment1\ \ and\ \ Segment2\ \ are\_subject\_to\_oceanic\_cpr})\\
\&\\
(\mathit{Flight\_level1\ \ lies\_in\_flight\_level\_range\_of\ \ Segment1})\\
\&\\
(\mathit{Flight\_level2\ \ lies\_in\_flight\_level\_range\_of\ \ Segment2})]\\
\quad =>\\
\quad [(\mathit{the\_min\_vertical\_sep\_Val\_in\_feet\_required\_for}\\
\qquad \mathit{Flight\_level1\ \ of\ \ Segment1\ \ and\ \ Flight\_level2\ \ of\ \ Segment2}) \ = \ 1000\\
\quad <=>\\
\quad [(\mathit{both\ \ Segment1\ \ and\ \ Segment2\ \ are\_flown\_at\_subsonic\_speed})\\
\quad \&\\
\quad (\mathit{bothFlight\_level1\ \ and\ \ Flight\_level2\ \ are\_at\_or\_below\ \ FL}\ 290)]]
\end{array}
$$

This captures a rule which says that in certain situations there has to be a vertical separation of 1000 feet between aircraft. Again, note the use of mix-fix, readable predicates, contributing to the overall transparency of the model. Below the Separation Value Axioms in the hierarchy lie a larger group of Auxiliary Axioms, defining various auxiliary predicates and functions used in the Separation Value Axioms.

The higher levels of the specification are anchored by the "Domain Object Axioms", which constrain the meaning of the primitives associated with each sort. Sorts were textually encapsulated in definition modules, where the signature and axiomatic definition of operations (predicates and functions) of that sort resided. This gives the requirements model the object-centred flavour one expects to find in an algebraic specification, although the use of an object inheritance technique was not required. For example, an extract of the sort Segment is given in figure 2. This was the largest sort definition having 60 functions, 20 predicates and 50 axioms associated with it.

$\mathit{Sortname}: \ \mathit{Segments}$

$\mathit{Function\ \ names}:$

$\mathit{the\_Segment}(\mathit{Profile}, 4D\_pt1, 4D\_pt2, \mathit{Val})$

$\mathit{the\_Profile\_containing}(\mathit{Segment})$

$\mathit{the\_entry\_4D\_pt\_of}(\mathit{Segment})$

$\mathit{the\_exit\_4D\_pt\_of}(\mathit{Segment})$

$\mathit{the\_machno\_Val\_on}(\mathit{Segment})$

$\mathit{the\_cruise\_climb\_status\_Val\_of}(\mathit{Segment})$

$\ldots$

$\mathit{Predicate\ \ names}:$

$\mathit{Segment1} \ = \ \mathit{Segment2}$

$\mathit{Segment1} \ \backslash = \ \mathit{Segment2}$

$(\mathit{Int\_gte\_0} \ \ \mathit{is\_a\_min\_long\_sep\_value\_for} \ \ \mathit{Segment1} \ \ \mathit{and} \ \ \mathit{Segment2}$
$\quad \mathit{entered\_via\_the\_mst\_command})$

$(\mathit{time\_periods\_of} \ \ \mathit{Segment1} \ \ \mathit{and} \ \ \mathit{Segment2} \ \ \mathit{overlap})$

$(\mathit{flight\_level\_ranges\_ofSegment1} \ \ \mathit{and} \ \ \mathit{Segment2} \ \ \mathit{overlap})$

$(\mathit{Flight\_level} \ \ \mathit{lies\_in\_flight\_level\_range\_of} \ \ \mathit{Segment})$

$\ldots$

$\mathit{Axioms}:$

$\mathit{Segment1} \ = \ \mathit{Segment2}$

$<=>$

$[\mathit{the\_entry\_4D\_pt\_of}(\mathit{Segment1}) \ = \ \mathit{the\_entry\_4D\_pt\_of}(\mathit{Segment2})$

$\&$

$\mathit{the\_exit\_4D\_pt\_of}(\mathit{Segment1}) \ = \ \mathit{the\_exit\_4D\_pt\_of}(\mathit{Segment2})$

$\&$

$\mathit{the\_machno\_Val\_on}(\mathit{Segment1}) \ = \ \mathit{the\_machno\_Val\_on}(\mathit{Segment2})$

$\&$

$\mathit{the\_Profile\_containing}(\mathit{Segment1}) \ = \ \mathit{the\_Profile\_containing}(\mathit{Segment2})]$

$\ldots$

Figure 2: Part of the Sort “Segment”

It is possible to follow down chains of definitional axioms until one reaches primitive predicates and functions that require factual profile data (i.e. sort instances) to be evaluated. The specification itself does not include instances of sorts, but for the purposes of animation, the CPS was supplemented with particular details of an oceanic airspace, containing persistent information regarding sort instances (aircraft makes, airfield positions and so on). Finally, profiles themselves need to represented as sort instances to allow evaluation of the conflict prediction function.

An Alternative Formalism

As we considered the specification language Z to be the most serious rival to the choice of MSFOL, in this section we will briefly compare the two for this application. Z has been proposed for use in requirements capture [24] and although classed as a *model-based* formal specification language, it can just as easily be used to capture domain knowledge in a purely axiomatic way, by naming types and placing logical axioms in schemas around these types. This would result in the kind of loose specification that we argued for above.

We illustrate the difference between Z and the customised MSFOL by comparing the MSFOL encoding of the box conflict axiom shown on page 12 with the Z encoding shown in figure 3. Although the Z schema contains signatures of the relations as well as the axiom itself, inspection of the two encodings shows up little difference, except that the MSFOL version is arguably more readable and less “mathematical”. Readability was a key concern as it was important that the CPS was in a form (or easily translated to a form) that could be read by air traffic control experts for the purposes of validation. We also felt that the readability of the CPS would be increased if type information was separated from the main body of axioms and held as part of the grammar/lexical rule definition (although type information was suggested by the use of appropriate namings in the MSFOL).

Apart from readability, the reasons for selecting MSFOL over Z for capturing the functional requirements of the *oceanic ATC domain* are:

- MSFOL could be represented entirely with standard ASCII characters, so that files containing axioms did not require the use of special fonts such as those needed for Z. This makes editing and processing of axiom files more straightforward, and it enhances the portability of axiom files between different machines and software applications.

- in the *oceanic ATC domain*, no changes of state occur and so it was unlikely we would need to refine our specification to include a state model. Thus we would not need to call on Z's huge collection of set-based notation (as detailed in [25]).

- while tool support for MSFOL was not as readily available as in the case of Z , the use of a “mainstream” logic meant that tools were easy to construct or to import. Creating our own tools environment meant that we could easily interface to and extend it, an important factor in such an exploratory project.

$[SEGMENT, TIME]$

$$areSubjectToOceanicCpr : SEGMENT \leftrightarrow SEGMENT$$

$$isInOverlapTimeWindowFor : TIME \leftrightarrow (SEGMENT \times SEGMENT)$$

$$isAtOrLaterThan : TIME \leftrightarrow TIME$$

$$haveBoxConflictAt : (SEGMENT \times SEGMENT) \leftrightarrow TIME$$

$$theNextIntegerTimeInMinsAfter : TIME \rightarrow TIME$$

$$\begin{array}{l}
haveBoxConflictAtSomeTimeAtOrBetween : \\
\quad (SEGMENT \times SEGMENT) \leftrightarrow (TIME \times TIME) \\
\hline
\forall\, Segment1, Segment2 : SEGMENT;\ Time1, Time2 : TIME | \\
\quad (Segment1 \text{ and } Segment2 \in areSubjectToOceanicCpr \wedge \\
\quad Time1\ \underline{isInOverlapTimeWindowFor}\ Segment1 \text{ and } Segment2 \wedge \\
\quad Time2\ \underline{isInOverlapTimeWindowFor}\ Segment1 \text{ and } Segment2 \wedge \\
\quad Time2\ \underline{isAtOrLaterThan}\ Time1) @ \\
Segment1 \text{ and } Segment2\ \underline{haveBoxConflictAtSomeTimeAtOrBetween} \\
\qquad Time1 \text{ and } Time2 \\
\Leftrightarrow \\
(Segment1 \text{ and } Segment2\ \underline{haveBoxConflictAt}\ Time1 \\
\vee \\
Segment1 \text{ and } Segment2\ \underline{haveBoxConflictAtSomeTimeAtOrBetween} \\
\qquad \underline{theNextIntegerTimeInMinsAfter(Time1) \text{ and } Time2})
\end{array}$$

Figure 3: Example Z Rule

- it is a little awkward to express general relational predicates in Z. Firstly, n-ary relations, where $n > 2$, have to be split into pairwise relations, as in the example in figure 3. Secondly, predicate names naturally expressed in mix-fix form have to be expressed using one contiguous identifier.

The Validation Process

Validation of Formal Requirements Models

Many problems are associated with requirements validation, especially when knowledge has to be elicited from domain experts. In fact one could contend that a correct and complete model of the domain could not exist because there is rarely full agreement among experts, and their understanding of the domain tends to change over time. The optimal solution seems to be in promoting the "fit" between model and domain in various ways, whilst allowing efficient means of model maintenance. We identify two important features of the validation process that support this:

- Diversity: errors occurring in a model can be syntactic or semantic, and may be of omission or commission. Different kinds of validation may unmask errors of different kinds: hence a range of validation processes is advisable.

- Automation: a major factor in the design of the validation process is allowing for the process to be repeated many times. This repetition will initially be frequent, although even after the acceptance of the requirements model, re-validation of an updated model is essential. Hence there is an overwhelming need to automate as many parts of the process as possible.

A Framework for Validation

We outline five separate ways in which a formal requirements model could be validated; their relation to the requirements model is as shown in figure 4, referenced by process numbering. In both figure 4 and 5, boxes represent documents or datastores and ovals represent processes or processors. In this context, we use the word "validation" in the widest sense, to include the removal of any class of error from the model.

- **by syntactic checking (process 1):** Under this heading we group the removal of syntactic errors such as spelling mistakes, as well as illegal use of logical operators and type errors in predicate and function arguments. This check will be performed automatically by a parsing tool, and will form the front end to the FREE as shown in figure 4.

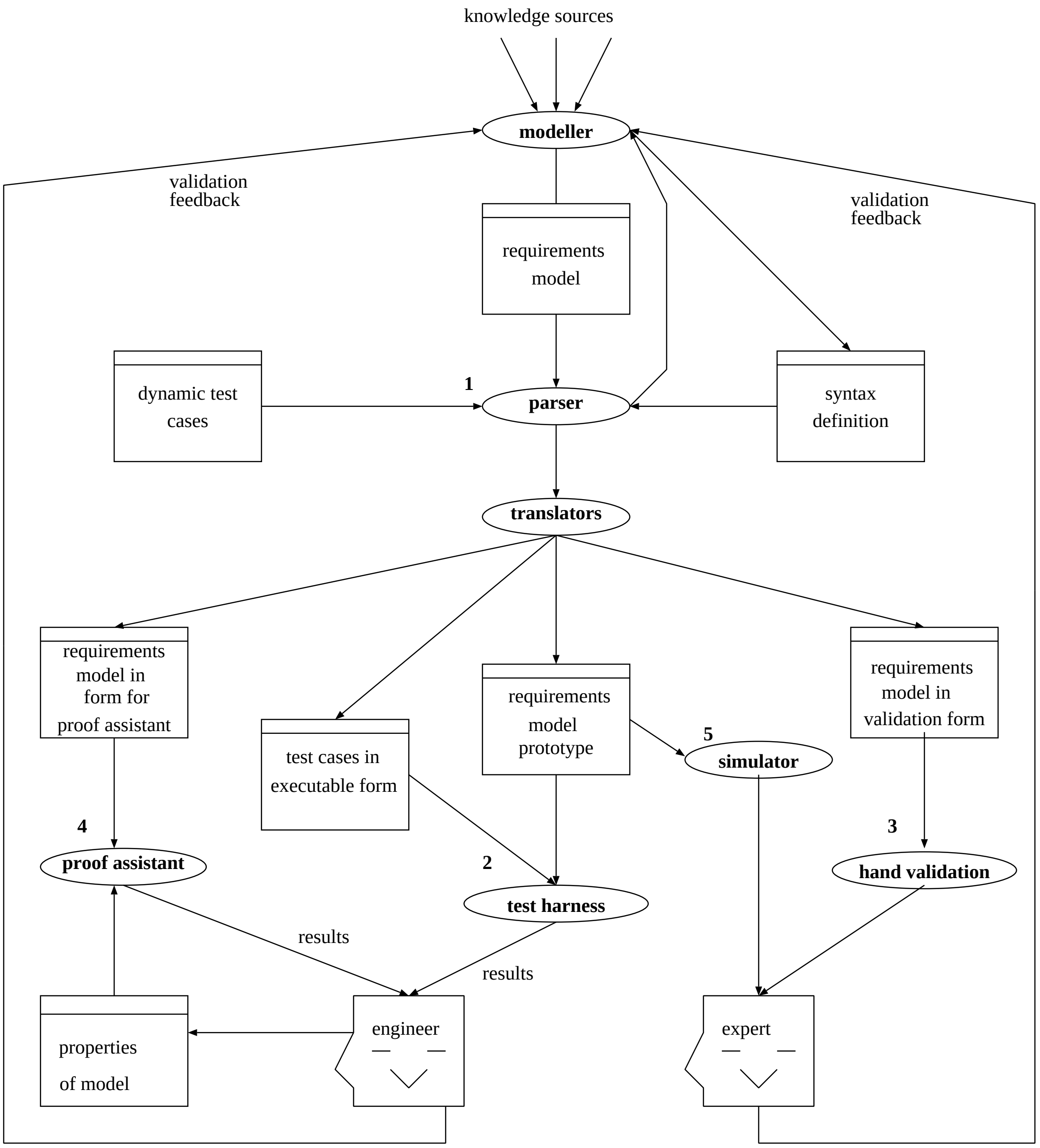

Figure 4: A Formal Requirements Engineering Environment (FREE)

- **by dynamic testing (process 2):** Being able to generate a prototype automatically has several advantages, principally that captured requirements can be immediately tested, without the need for any software development work. The ease and degree of automation involved in its production will depend upon the application. Historic data can be extracted from the application domain and systematic dynamic testing can be performed in a similar manner to program testing, using a test harness.

- **by hand (process 3):** We define hand validation as the use of domain experts to read through and comment on the validity of (a presentation of) the model. This is arguably the most time consuming and unpredictable form of validation, but helpful texts exist relating to the conduct of such interviews and meetings [26]. The FREE should output an easily readable form of the domain capture formalism, substituting mathematical symbols with a natural language translation, and producing diagrams describing the model's structure.

- **by formal reasoning (process 4):** Requirements specifications can be used and re-used for different applications and objectives, rather than just being used as a specification of a particular program (and in a way dynamic testing only tests one particular *behaviour*). We require, therefore, a way of reasoning with the model to investigate its general behaviour and logical consequences. Ideally the FREE would incorporate a proof assistant or theorem prover so that an engineer could formulate desirable properties of the model and set them up as theorems to be proved. The proof process often uncovers errors whereas a completed proof heightens confidence in the model. A fully automated route from the requirements model to the proof of model properties would mean easy re-execution of these proofs after model updates.

- **by simulator (process 5):** Testing of a more user-oriented kind may be performed with a simulator. This should be integrated with the automatically generated prototype via a custom-built interface, allowing the users to test the model themselves. If the simulator is constructed in such a way that the user can ask for explanations of the behaviour of the model, it can be used in conjunction with hand validation sessions (as in process 3 above).

No one form of validation should be used to convince us that a model is valid. For example, dynamic testing of requirements models gives a similar scenario to that of program testing - only showing the presence of errors but not their absence. The whole validation process should be systematic, and execution of its sub-processes sensibly ordered, with syntactic parsing of the specification preceding any other sub-process. In both hand validation and testing, the scope of validation should be recorded, as these processes may be iterated many times. In summary, a systematic approach should be imported into requirements testing from the conventional field of Software Testing.

Validation in the FAROAS Project

Our initial capture of the *oceanic ATC domain* model led to a document of about three hundred axioms, structured in a hierarchical form. Knowledge was acquired chiefly from documents, although several interview sessions were arranged with air-traffic control staff to elicit background knowledge. Once the model size stabilised, we set about tackling the validation stage.

Below we describe the validation processes that we used. The engineering environment that we in effect created is shown in figure 5. The FREE was implemented in Quintus Prolog[1] on a Sun workstation, using the Unix operating system. All key files were held under the Source Code Control System, a standard Unix configuration management tool, which provides facilities such as file protection, automated version control and logging of alterations.

Process 1: Syntax Checking

As a pre-condition for any other validation process, the whole of the CPS must parse successfully, thereby showing that its syntax and its defining grammar are mutually consistent. In effect this use of the parser is similar to tools such as *f*UZZ for Z [19]. The grammar that defines the syntax of the domain capture formalism has a level that applies to first order logic generally, and a customised level, which, for example, allows us to control the actual names of variables for each sort within axioms. Hence the content as well as the form of sentences in the formalism is strictly controlled, and any non-conformance will result in failure of the parse. Errors identified in this process may not only arise from oversights in the specification; it may be decided that the grammar itself is inadequate. The grammar was validated by the client through visual inspection of its definition and its use in the documents describing the CPS. As can be imagined, over the course of the project, syntax checking uncovered errors too numerous to count!

Process 2: Dynamic Testing

The most complex part of the parsing and translation process is the tool which produces an executable form of the CPS (the production of a Prolog prototype as shown in figure 5). It must be emphasised that the decision on what execution form to use was not made until after the initial requirements capture. If the execution form is known before the construction of the specification, then it can have an undue influence on the representation of the domain, possibly compromising its clarity and natural structure.

Inspection of the CPS showed that the logic could be transformed to Horn clauses, and

[1] Copyright @1991 Quintus Corporation

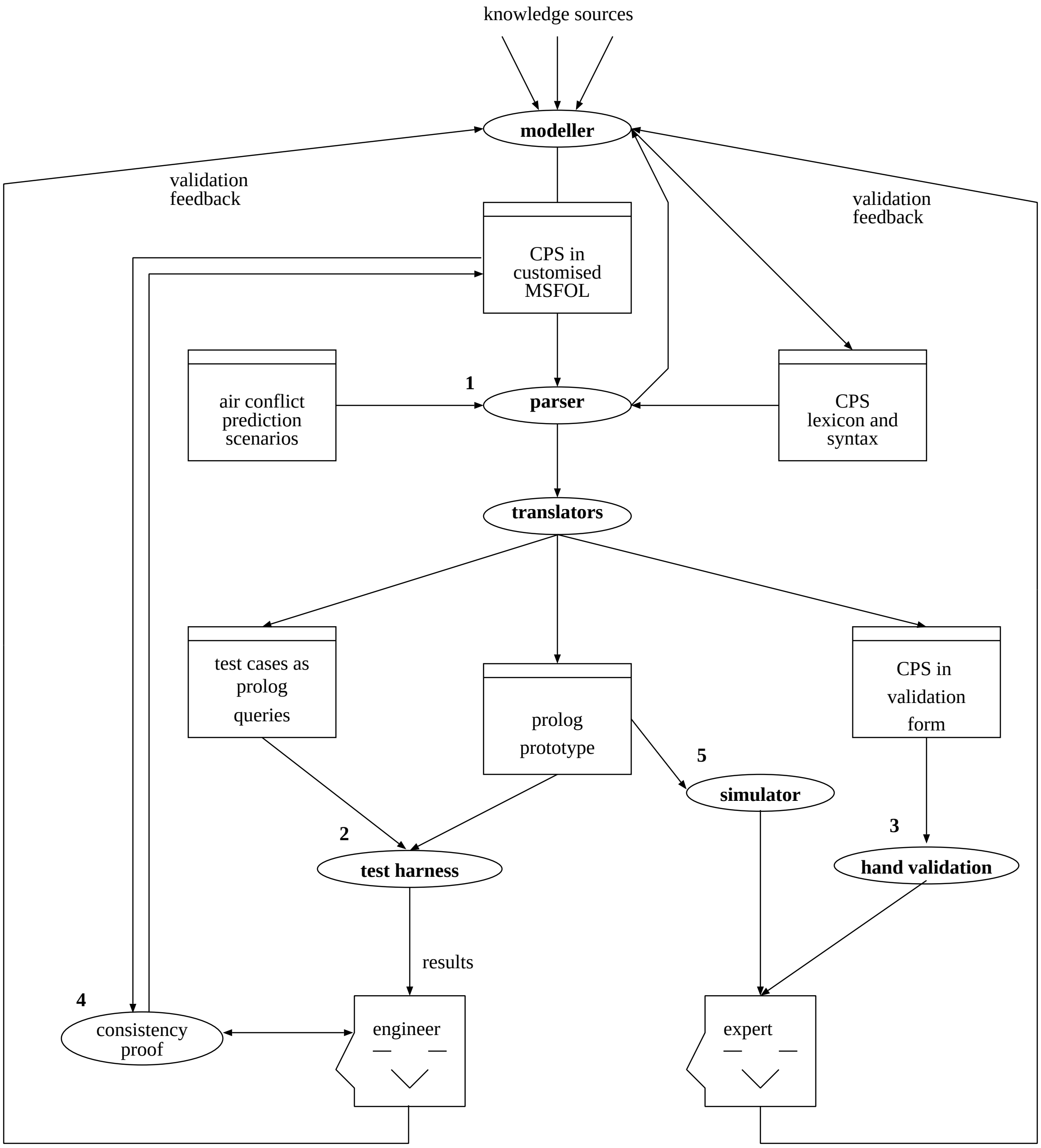

Figure 5: The FREE for the oceanic ATC domain

hence it was quite feasible to automate the process of translation to an executable Prolog prototype. The translation procedure (which takes about 1 minute to execute on our chosen architecture) was built so that each time the specification is updated and successfully parsed, the output parse-tree from the parser feeds into the translator which automatically creates the Prolog prototype (see figure 5). For example, the box conflict axiom referred to on page 12 is automatically translated to the following Prolog clause:

```
box_conflict_exists_between_linear_tracks_of_at_some_time_
    at_or_between(Segment1,Segment2,Time1,Time2):-
    are_subject_to_oceanic_cpr(Segment1,Segment2),
    is_in_overlap_time_window_for(Time1,Segment1,Segment2),
    is_in_overlap_time_window_for(Time2,Segment1,Segment2),
    Time2 is_at_or_later_than Time1,
    (box_conflict_exists_between_linear_tracks_of_at(Segment1,Segment2,Time1)
   ;
     the_next_integer_Time_in_mins_after(Time1,Time3),
     box_conflict_exists_between_linear_tracks_of_at_some_time_
 at_or_between(Segment1,Segment2,Time3,Time2)), !.
```

The prototype was used for dynamic testing with an historical test set of client supplied conflict scenarios that tested top-level conflict prediction tasks, and a set of “in-house” generated tests which were designed to systematically test lower level and auxiliary predicates (numbering about 400 tests in total).

Insecurities in Prolog to do with types were dealt with by ensuring that any use of the prototype was channelled through the FREE. Thus test data is input in the MSFOL language, and tools parse it, translate it into Prolog queries and then input it into a test harness which runs the prototype. After execution of all the tests the output contains the queries in a validation form together with the expected and actual results. The validation feedback loop shown in figure 5 was invoked for 5 tests which gave incorrect results, and this process eventually led to the uncovering of 3 errors which were were present in the CPS. Significantly, two of these errors to do with the boundaries of aircraft vertical separation had been initially missed at hand validation meetings, emphasising the need for multiple forms of requirements validation. The tests were run repreatedly until a 100% success rate was achieved on both the client supplied and in-house generated test sets.

Process 3: Hand Validation

A validation form was required so that the specification could be presented to air traffic control experts. As the domain capture formalism was already quite readable the validation form was obtained simply by replacing logical symbols with their natural language translation, and improving the layout and presentation of the axioms. Sentences in validation form were automatically output from the parsing and translation tools, as illustrated by the resultant form of the box conflict axiom below:

```
FOR ANY Time1, Segment1, Segment2 and Time2
    IF
       Segment1 and Segment2 are subject to oceanic
       conflict prediction and resolution
       AND
       Time1 is in the overlap time window for Segment1 and Segment2
       AND
       Time2 is in the overlap time window for Segment1 and Segment2
       AND
       Time2 is at or later than Time1
    THEN
       box conflict exists between the linear tracks of
       Segment1 and Segment2 at some time  at or
       between Time1 and Time2
       IF AND ONLY IF
 EITHER
   box conflict exists between the linear
   tracks of Segment1  and Segment2 at Time1
 OR
   box conflict exists between the linear
   tracks of Segment1 and Segment2 at
   some time  at or between the next integer Time in
   mins after Time1 and Time2
```

Hand validation meetings were arranged, initially to check the scope of the specification, and later to validate individual axioms. Tree diagrams were used to show the hierarchical interconnection of axioms, allowing validators to "navigate" through the model. As the domain capture formalism allows the structure of the *oceanic ATC domain* to be preserved, air traffic control experts found both it and its validation form understandable, stimulating debate and allowing them to easily uncover errors in our initial understanding of the domain. With so many axioms, however, hand validation was still a long and painstaking process, and there was time at each meeting to study only a part of the whole specification. During the course of the FAROAS project 4 validation meetings were held, each lasting 2-3 days and involving 4-5 personnel. For each meeting the number of errors and omissions found in the specification, ranging from the trivial to the serious, was in the range 10 – 25.

Process 4: Formal Reasoning

During the project we performed a proof of the overall consistency of the CPS without the use of "intelligent" computer-based support tools. The proof strategy used was to view the requirements model as a theory, and construct a particular interpretation for it which satisfied each of its axioms. As a preliminary to the proof, the set of axioms was reduced by sifting out all those that are definitional. These axioms can be regarded as expressing an extension to the language of the theory, in effect introducing a convenient "abbreviation" for more complex formulae involving lower-level predicates and functions. Approximately 110 axioms were unique, unconditional extensions of this nature, and were removed so that we could concentrate on proving the consistency of the reduced set of axioms. Then an interpretation function for the reduced set of axioms was constructed, and we used it to show that at least one set of objects existed for which the axioms are true. The main effort involved here was in producing an argument that separate parts of multiple conditional definitions for predicates or functions were mutually exclusive.

Proving this type of consistency draws attention to the overall structure of the specification, and, in the event, one error was removed from the specification during the proof process. On the other hand, generating hand proofs is a slow and potentially error prone process for specifications of this size, and a feasibility study to incorporate automated support as indicated in our idealised FREE in figure 4 was carried out in the project (and is discussed later under "Future Work").

Process 5: The Interface and Simulator

An interface for the FREE was produced using Quintus Pro-Windows[2], giving a consistent "look and feel" to the environment. This allows the CPS and the grammar defining its syntax to be securely maintained, and each time the specification is parsed successfully, a fresh executable prototype and validation form is generated. Any changes made to the CPS can thus be dynamically tested, and viewed in a validation form, in a matter of minutes.

The interface also provides the front-end to the simulator which consists of a windowing system that allows air traffic control experts to:

- input flight profiles
- run the conflict prediction function
- request explanations of conflict decisions

In the event of a detected conflict between two flight profiles the simulator can if required identify the segments which were in conflict, and indicate the required separation values (vertical, lateral and longitudinal) that were violated. The simulator can thus help air traffic controllers to validate that conflict decisions made by the prototype are made on the same basis as their own.

Results of the Validation Process

A slightly surprising result was that dynamic testing and the formal consistency proof uncovered relatively few errors. This may have meant they were not very effective, or that the

[2] Copyright @1991 Quintus Corporation

model was already a good fit for the domain. One argument to support the latter reason was that the syntax checking and hand validation processes were started well in advance of the other processes, with only one hand validation meeting occurring after the first round of dynamic testing. Thus many errors both of omission and commission had already been removed. On the other hand, our client supplied dynamic test sets were not exhaustive, and our consistency proof was only one aspect of what could be termed formal validation (and we return to these issues in the section on Future Work).

The criterion for adequate validation of the CPS (within the boundaries of the project) was agreed in an official test plan, and entailed a sequential error-free execution of processes 1, 2, and 4, after the errors uncovered at a final hand validation meeting had been removed. The scope of the hand validation process covered the Conflict Prediction Axioms and the Separation Value Axioms, while the dynamic testing was limited to the client supplied and "in-house" test set as mentioned above. At the end of the Project this criterion was met successfully.

Although the simulator (process 5) was not available until the end of the project, it has already shown a potential for use in the maintenance of the model: during the last hand validation meeting it proved possible to remove errors from the CPS, generate a new prototype and use the simulator to test the corrections. Finally, one often quoted point against formal specification is that a mathematical notation can be off putting to the customer. Our use of a readable formal specification language naturally capturing the current solution to this domain has contributed in no small way to the success of the validation process.

Conclusions

We have given an overview, using a real application, of a method for formal requirements capture and validation. In summary, this encompasses domain analysis, formalism choice, the development and customisation of a FREE, domain capture and validation.

This method addresses the problems of requirements capture through diverse, automated and systematic validation. The benefits of such a method include the user obtaining a validated, maintainable model of the domain which can be used as a prototype running system, for exploratory work on new requirements and for comparisons with any derived (or existing) implementations. If the prototype embedded in the simulator completely satisfies the users needs then software production at the sub-specification level has effectively been minimised to the production of tools such as translators and test-harness environments. On the other hand, the requirements model may be used as a sound specification from which software developers could generate efficient software that satisfies non-functional constraints, including interfacing and efficiency considerations, or the need to construct a running system within a certified programming language such as ADA.

Generalisation of the Method

Our method has been used for a particular industrial domain that is sufficiently important to require study and formal capture. The domain contained chiefly rule-based and object-centred types of technical, expert knowledge and a good deal of the knowledge could be extracted from documentation. These are the characteristics that seemed to make our approach appropriate to the *oceanic ATC domain.*

Using the method on similar but larger domains, resulting in larger axioms sets, would not seem to present a major problem. Our current CPS has a clear hierarchical structure (shown in figure 1) that could be combined with further structures, such as an axiom set capturing methods of conflict resolution of aircraft profiles (as opposed to conflict prediction).

Much more of a problem would lie in using the method on domains requiring a deeper representation, to cover probabilistic, modal or deontic information. In this case we feel problems to do with validation would be exacerbated, as the notation would be far less accessible to experts, and the generation of a prototype a lot less straightforward.

Future Work

While we believe our validation framework as outlined above encourages a rigorous approach to validation, the particular validation plan we carried out in the 20 month project was limited by time. Firstly, a considerably larger test set would be required to ensure every axiom was tested fully. Secondly, all the axioms, not just Conflict and Separation Value Axioms require hand validation.

Finally, the scope and operation of formal validation needs to be extended and improved. As well as performing a hand consistency proof, during the project we investigated the feasibility of using the B-Tool [27], a generic proof assistant, to support such activities, with a particular view to determining whether such a tool would be useful in maintaining the CPS. The B-Tool is a proof assistant which arose out of early work on the B methodology by J. R. Abrial [28]. Its main function is that of supporting formal methods experts in constructing proofs to demonstrate important properties of formal specifications. It is a generic tool, in the sense that it can be customised for use with specifications written in various notations: definitions of operators and inference rules used in first order predicate logic are built into the tool, but other definitions and rules can be added by the user. During our feasibility study we used the B-tool to prove completeness and consistency results for vertical separation rules of the CPS. Given this initial success future work could concentrate on providing an automatic link in the FREE to a proof assistant of this sort.

Acknowledgements

This work was aided by the expertise and advice of members of The Civil Aviation Authority. In particular, the interpretation of MATS-2 was aided by the separation rules in informal "production rule" form written by K.McClachlan, by the practical air-traffic control experience of E.Payne, and by R.Thomson's written answers to queries. Overall liaison and project meetings were organised initially by D.Snowden, and later on by T.Smith. We would also like to thank P.Allen of Huddersfield University for helpful discussions on how Z might have been applied to this application.

References

[1] A. W. Brown, A. N. Earl, and J. A. McDermid. *Software Engineering Environments: Automated Support for Software Engineers.* McGraw-Hill, 1992.

[2] M. Ryan, J. Fiadeiro, and T. Maibaum. Sharing Actions and Attributes in Modal Action Logic. In T. Ito and A. Meyer, editors, *Proceedings of the International Conference on Theoretical Aspects of Computer Science.* Lecture Notes in Computer Science 526, Springer Verlag, 1991.

[3] Manual of Air Traffic Services Part 2 – Operations. Technical report, The Civil Aviation Authority – National Air Traffic Services.

[4] M. Chesser. FDPS Conflict System Design Recommendation. Technical Report SSG/D/91/0014, The Civil Aviation Authority National Air Traffic Services, 1991.

[5] K. McLachlan. Requirement Specification for the CPR Task. Technical Report SSG/S/91/0020, The Civil Aviation Authority National Air Traffic Services, 1991.

[6] B. Sowerbutts. Formal Methods in Airborne Collision Avoidance Standards (Roke Manor Research). In *IEE Colloquium on "Software in ATC Systems – The Future"*, 1992.

[7] E. Brinksma. LOTOS – A formal description technique based on the temporal ordering of observational behaviour. Technical Report ISO DP 8807, International Standards Organisation, 1986.

[8] C. Taylor and Y. Naik. A Survey of Formalisms for the Specification of MATS-2. Technical Report CAA/FAROAS/05/005/02, The City University, Department of Computer Science, 1992.

[9] M. Minsky. A Framework for Representing Knowledge. In P. Winston, editor, *The Psychology of Computer Vision.* New York: McGraw-Hill, 1975.

[10] B. G. Buchanan and E. H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, 1984.

[11] P. Smets, P. Mamdani, E. H. Duboisand, and H. Prade.

[12] S. J. Greenspan, A.Borgida, and J. Mylopoulos. A requirements modeling language and its logic. *Information Systems*, 11(1), 1986.

[13] A. Dardenne, S. Fickas, and A. van Lamsweerde. Goal-directed concept acquisition in requirements elicitation. In *Proceedings of the Sixth International Workshop on Software Specification and Design.* IEEE Computer Society Press, 1991.

[14] C. Goguen, H. Kirchner, A. Kirchner, J. Megnelis, J. Meseguer, and T. Winkler. An

Introduction to OBJ3. Technical Report 88-R-001, Centre de Recherche en Informatique de Nancy, 1988.

[15] D. Coleman, C. Dollin, R. Gallimore, P. Arnold, and T. Rush. An Introduction to the Axis Specification Language. Technical Report Hewlett Packard Information Laboratory Report HPL-ISC-TR-88-031, Software Engineering Department, Hewlett Packard, Bristol, U.K., 1988.

[16] J. G. Turner and T. L. McCluskey. *The Construction of Formal Specifications: An Introduction to the Model-Based and Algebraic Approaches.* McGraw-Hill series in Software Engineering, 1994.

[17] C. B. Jones. *Systematic Software Development using VDM.* Prentice Hall International Series in Computer Science, 1990.

[18] B. Potter, J. Sinclair, and D. Till. *An Introduction to Formal Specification Using Z.* Prentice Hall International Series in Computer Science, 1991.

[19] J. M. Spivey. *The fUZZ Manual.* 1991.

[20] C. Taylor and Y. Naik. Initial Domain Analysis of MATS-2. Technical Report CAA/FAROAS/05/005/01, The City University, Department of Computer Science, 1992.

[21] B. Cohen, W. T. Harwood, and M. I. Jackson. *The Specification of Complex Systems.* Addison-Wesley, 1986.

[22] Meinke and J. Tucker. *Many Sorted Logic and Its Applications.* Wiley, 1992.

[23] W. Clocksin and C. Mellish. *Programming in Prolog.* Springer-Verlag, 1981.

[24] D. Bolton, S. Jones, D. Till, D. Furber, and S. Green. Using domain knowledge in requirements capture and formal specification construction. Technical Report TCU/CS/1992/24, City University, 1992. Proceedings of the Workshop on Requirements Capture and Analysis held at Oxford University, December 1991, to be published as Requirements Engineering, Academic Press Series on People and Computers.

[25] J. M. Spivey. The Z Notation: A reference manual. Technical report, 1988.

[26] E. S. Cordingley. Knowledge Elicitation Techniques for Knowledge-Based Systems. In D. Diaper, editor, *Knowledge Elicitation: Principles, Techniques and Applications.* Ellis Horowood, 1989.

[27] J.R. Abrial. B-tool reference manual, 1991. Version 1.1.

[28] J.R. Abrial, S. Davies, D. Nielson, P.N.Scharbach, and I.H. Sorensen. B-method Overview. Technical report, BP International Limited, 1992.