Redevelopment of an Industrial Case Study Using Event-B and Rodin

Abdolbaghi Rezazadeh[1] Neil Evans[2] Michael Butler[1]

[1] School of Electronics and Computer Science, University of Southampton, UK
[2] AWE, Aldermaston, UK

Abstract. CDIS is a commercial air traffic information system that was developed using formal methods 15 years ago by Praxis[1], and it is still in operation today. This system is an example of an industrial scale system that has been developed using formal methods. In particular, the functional requirements of the system were specified using VVSL – a variant of VDM. A subset of the original specification has been chosen to be reconstructed on the Rodin[2] platform based on the new Event-B formalism. The goal of our reconstruction was to overcome three key difficulties of the original formalisation, namely the difficulty of comprehending the original specification, the lack of any mechanical proof of the consistency of the specification and the difficulty of dealing with distribution and atomicity refinement. In this paper we elucidate how new formal notation and tool can help to eliminate this difficulties.

1. Introduction

The CCF[3] Display and Information System (CDIS) is a computerised system that provides important airport and flight data for the duties of air traffic controllers based at the London Terminal Control Centre. Each user position is a workstation that includes a page selection device (to select CDIS pages) and an electronic display device (to display the selected pages). The original system was developed by Praxis in 1992 and has been operational ever since. This system is an example of an industrial scale system that has been developed using formal methods. In particular, the functional requirements of the system were specified using VVSL [1] – a variant of VDM [2]. The formal development resulted in about 1200 pages of specification documents and about 3000 pages of design documents. The reliability of the delivered system is encouraging for formal methods in large scale system development because the defect rate was a considerable improvement on other similarly sized projects [3]. However, no formal reasoning was applied to the specification.

This paper describes a case study of the RODIN project that is based on CDIS. The RODIN project is an EU IST project concerned with the provision of methods and tools for rigorous development of complex software-based systems. Contemporary tool support has been used to develop a formal specification. The objective of the experiment (that we shall refer to as `the CDIS subset') is to derive a methodology for large scale formal development. Redeveloping an existing system also allows us to reflect on the lessons learned from the original development. Our aim in this paper is to demonstrate how we have attempted to overcome the lack of comprehensibility, dealing with distribution and formal proof of the original CDIS development

[1] Praxis High Integrity Systems Ltd, UK
[2] RODIN (Rigorous Open Development Environment for Complex Systems) is an EU funded research project IST 511599 http://rodin.cs.ncl.ac.uk
[3] Central Control Function

by adopting a methodology that makes use of available tool support in an effective way. The rest of the paper is as follows: Section 2 gives an overview the case study and how it has been extracted form the original specification, Section 3 describes Event-B (which is our notation of choice), and Section 4 describes our methodology in defining an abstract specification of a generic display system and demonstrate how layered refinement used to achieve the main goals of the project. We conclude this approach in Section 5.

2. The CDIS Case Study

It is apparent from the previous section that CDIS system as a whole is a very complex system. In order to keep the case study manageable in the context of the RODIN project, a subset of the original CDIS has been carefully chosen for redevelopment [4]. However, rather than focusing on individual aspects of CDIS, a `vertical slice' has been taken so that all of the interesting features of the system are covered (albeit in a lesser form). At the heart of the CDIS subset is the `core specification' that gives the functional properties of the system, and shall be the focus of this paper. In addition to the core specification, there is a concurrency specification and a description of the user interface.

2.1. The Original Specification

The core specification is only one part of the overall CDIS documentation. It gives an idealised view of the entire functional behaviour of the system. (The design document states how this is actually realised.) In order to avoid ambiguity, in this paper we will often refer to the core specification as `the original VDM specification'.

The core specification consists of a number of VVSL modules, each of which contains type, constant and state definitions. (The bulk of the specification is made up of Boolean functions that are used in the pre/post conditions of other definitions.) A module can import other modules so that the imported definitions are available in the importing module. This gives a VVSL specification its structure. This approach encourages a bottom-up development in which the overall specification emerges from the way in which its modules are combined.

The core specification of CDIS comprises 15 modules. However, we can identify three main parts (or contexts):

- ***Airport-related*** data. This concerns airport-specific values such as weather or runway information. The Meta data module identifies the airport attributes and their value types. Functions are defined to update the values of the attributes. The Airport records module declares a state variable that holds all of the actual values of the attributes.

- ***Page-related*** data. This gives a device-independent and data-independent record of the pages that can be displayed by CDIS. Types are declared to model the layouts of pages. Actual pages are held in the state variables declared in the Pages module.

- ***Display-related*** data. This concerns the physical devices that are used to retrieve and display information.

Other subsidiary modules such as the date/time module are concerned with other important features of CDIS. By far the largest module in the core specification is ***EDD_displays*** that contains the operations of the system. All of the modules listed above are imported by ***EDD_displays*** to enable the definition of the operations.

2.2. Preliminary Observation and Reflection on the Original Specification

It is worth emphasising that the CDIS specification is necessarily complicated. Even though the core specification has been criticised for its complexity, it is unrealistic to expect any significant improvements in the size of a specification that captures all aspects of CDIS, regardless of the notation used. However, the bottom-up construction in VVSL forces a level of specification that is too detailed to get an appreciation of the overall system behaviour.

Too much complexity also precludes formal analysis. In order to reason about a specification formally, it is necessary to keep the level of detail as simple as possible. Otherwise mathematical proof becomes infeasible. Analysing monolithic specifications such as the CDIS core specification would be beyond the capabilities of contemporary formal methods tools without intense human intervention. This was not an issue during the original CDIS development because tool support was largely unavailable, and large-scale formal analysis was out of the question.

Another drawback of the original development is the lack of continuity from the specification to the design. In the idealised view of the core specification, updates are performed instantaneously at all user positions, whilst there is an inevitable delay in the actual system because the information must be distributed to the user positions. Hence, there is no natural refinement of the original specification (in the usual sense of the word) to the design. We are investigating more novel notions of refinement in order to find a suitable link between the two viewpoints.

3. New Event-B, Notation, Methodology and Development Environment

The original B method [5] is a well known approach to the formal specification and development of sequential computer programs. Inspired by action systems and other research in the area of distributed systems modelling, the B method has evolved to incorporate system modelling and distributed system development. This extension is called Event-B [6].

An abstract Event-B model comprises a static part called the context, and a dynamic part called the machine. The machine has access to the context via a SEES relationship. All sets, constants, and their properties are defined in the context. The machine contains all of the state variables. The values of the variables are set up using the **INITIALISATION** clause, and values can be changed via the execution of events. Ultimately, we aim to prove properties of the specification, and these properties are made explicit using the **INVARIANT** clause. The tool support generates proof obligations which must be discharged to verify that the invariant is maintained.

Events are specialised form on standard B operations [5]. In general, an event E is of the form

$$E \triangleq \textbf{WHEN } G(v) \textbf{ THEN } S(v) \textbf{ END}$$

where $G(v)$ is a Boolean guard and $S(v)$ is a generalised substitution (both of which may be dependent on state variable v). The guard must be true for the substitution to be performed (otherwise the event is blocked). There are three kinds of generalised substitution: *deterministic*, *empty*, and *non-deterministic*. The deterministic substitution of a variable x is an assignment of the form $x := E(v)$, for expression E, and the empty substitution is *skip*. The nondeterministic substitution of x is defined as:

$$\textbf{ANY } t \textbf{ WHERE } P(t , v) \textbf{ THEN } x := F(t , v) \textbf{ END}$$

Here, t is a local variable that is assigned non-deterministically according to the predicate P, and its value is used in the assignment of x via the expression F.

In order to refine an abstract Event-B specification, it is possible to refine the model and context separately. Refinement of a context consists of adding additional sets, constants or properties (the sets, constants and properties of the abstract context are retained).

Refinement of existing events in a model is similar to refinement in the B method: a gluing invariant in the refined model relates its variables to those of the abstract model. Proof obligations are generated to ensure that this invariant is maintained. In Event-B, abstract events can be refined by more than one event. In addition, Event-B allows refinement of a model by adding new events on the proviso that they cannot diverge (i.e. execute forever). This condition ensures that the abstract events can still occur. Since the new events operate on the state variables of the refined model, they must implicitly refine the abstract event *skip*. More information on Event-B methodology can be found in [6] and [7].

A contemporary tool has been developed under the RODIN project [8] to fully support the Event-B notation and methodology. Rodin is an open tool platform for the rigorous development of dependable complex software systems and services. It provides natural support for refinement and mathematical proof. This platform is based on the Eclipse framework and it is extendable with plug-ins. The extensibility of the main platform and its support for combination of different complementary tools, openness and customizability are very important aspects of the tool. They allow users to customize and adapt the basic tools to their particular needs.

4. CDIS Redevelopment in Event-B

In this section we describe our formal development of CDIS in Event-B. The original core specification of CDIS is our reference model and the main objectives are to address the following issues:

- The difficulty in comprehending the original specification and the need for better modularisation.
- The lack of any formal proof in the original development.
- The need to formally link the abstract specification to the distributed design.

In order to get a better overview of the entire process, we follow a top-down approach. At the top level, we ignore all of the airport-specific features to produce a specification describing a generic display system. Through an iterated refinement process, we introduce more features into the specification until all of the CDIS functionality is specified. This procedure is supported by the Rodin tool. At each step the tool generates a number of proof obligations which must be discharged in order to show that the models are consistent with their invariants. Since each refinement introduces only a small part of the overall functionality, the number of proof obligations at each step is relatively small (approximately less than 20).

4.1. Description of the Achieved Results - Centralised Version

In this stage we address the first two issues which we have identified in the previous section. Thus we shall be concerned with an idealised view of the system, as modelled in the original core specification. We model a system that has a centralised database from which information can be retrieved. We begin by constructing a specification for a generic system (which will be, of course, somewhat influenced by the original VDM specification) and, through subsequent refinements, introduce more and more airport specific details so that we produce a specification of the necessary complexity, and reason about it along the way. By providing a top-down sequence of refinements it is possible to select an appropriate level of abstraction to view the system: an abstract overview can be obtained from higher level specifications whilst specific details can be obtained from lower levels.

4.1. 1. Abstract Specification (*CDIS_Context + ABS_DISPLAY*)

The abstract specification for a generic system includes two parts. The static part, which defines the reference sets and constants, such as ***Pages***, ***Displays*** and other related attributes is named ***CDIS_Context***.
The second part, the dynamic part, is defining the variables, their related invariants, and events. The variable ***database*** represents the stored data, and ***page_selections*** records the page number currently selected at a user position. The variable ***private_pages*** holds the page contents of a page prior to release. This is intended to model an editor's ability to construct new pages before they are made public. Finally, ***trq*** models the `timed release queue' that enables a new version of a page to be stored until a given time is reached, whereupon it is made public.

Almost all of the events given in the ***ABS_DISPLAY*** correspond to operations defined in the original VDM specification. One exception is the ***VIEW_PAGE*** event that uses the ***disp_values*** function to output an actual display. This is an addition to the original VDM specification but, since outputs must be preserved during refinement, it forces us to ensure that the appearance of actual displays is preserved.

4.1. 2. First Refinement – Introducing Page Layout History

This refinement is not introducing significant changes into the specification. In this level we have just introduced the history for page layout. When an existing page layout has been updated by the editors, the system keeps the previous page layout as one step history of changes. In this level no new context has been introduced.

4.1. 3. Second Refinement – Adding time

The abstract specification omitted many of the features that characterise CDIS. However, this made it possible to give a broad overview of the system, including its state variables and operations, within a few pages. Now we use the specification as a basis for further refinements in which the omitted details are introduced. As a second refinement, we introduce a notion of time so that we can add age information to attributes, and add creation and release times to pages. This will give us the necessary apparatus to model the intended behaviour of the timed release queue. This refinement and subsequent refinements will demonstrate how important features of CDIS are added to the specification incrementally.

In terms of the CDIS subset, there are two main reasons for adding time: each piece of airport data has an age which affects how it is displayed, and the version of each page that is displayed is also time-dependent. In this stage both the context and the model have been extended with appropriate constructs to deal with the notion of time. To add the time we have used the proposed syntax for record types in [9]. This approach provides a method for gradual refinement of complex data types and upholds the principles of Event-B refinement.

4.1. 4. Third Refinement – Introducing Critical Fields and Acknowledge

Several other aspects of CDIS can affect the way values are displayed. One requirement is that there is some critical information which they subjected to regular updates. Any new updated values should be highlighted when they are displayed and they should be acknowledged by the operators. Hence, with each attribute value, we need to record whether it is a critical field or not. When a critical field has been updated it may effects many active pages currently viewed by different user positions. In this stage we have introduced the ***edd_acks_required*** and related sets

and constants to extent our model with the critical fields' requirement. Again here we have extended both the context and the model.

4.1. 5. Fourth Refinement – Introducing Page Overlays

Airport pages comprise a pair of graphic background overlays and a layout descriptor for transient data fields. One overlay is permanently displayed, the other is selectable using the reveal/conceal facility. Transient data fields are always displayed – they are unaffected by the reveal/conceal state. Airport pages need to be validated to ensure that none of the transient data fields are obscured by the background or overlay.

The reveal/conceal facility applies only to ***EDDs*** displaying pages. For all ***EDDs*** there is a means of toggling the reveal/conceal state of the display. This affects only those pages on display that consist of a permanent background and an overlay. Concealed displays models the set of ***EDDs*** for which the overlay is concealed. The overlays of pages displayed on all other **EDDs** are revealed. The act of toggling reveal/conceal at an **EDD** adds the ***EDD*** to the set if it is not a member beforehand, otherwise removes it. The specification has been augmented by a fourth refinement in which this features are introduced.

4.1. 6. Fifth Refinement – Highlighting Manual Interaction

Another aspect of CDIS that can affect the way values are displayed is manually updated values. One requirement is that any manually updated values should be highlighted when they are displayed. Hence, with each attribute value, we need to record whether it was updated manually. Once again, we use our notion of record refinement to achieve this.

The Boolean value associated with the new field manually updated indicates whether the attribute's latest recorded value (accessed via the value field) has been input manually. In this case, we extend the record type ***Attrs*** with a Boolean flag which indicate whether or not the field has been updated manually. We included this requirement in the fifth refinement level.

4.1. 7. Sixth Refinement – Introducing Concrete Values and Error Handling

The ultimate aim of the refinement process is to construct a specification in which constants and variables are associated with concrete values and events are defined to maintain the state accordingly. As part of this process, we have to separate a single abstract type into several subtypes. In the case of CDIS, this technique is used to introduce concrete attribute identifiers and value types into the specification. For example, the original VDM specification defines ***Attr*** value as a union type made up of value types such as ***Wind_direction*** and ***Wind_speed***. Although union types do not exist in B, we employ a separation technique to achieve the same goal. We define a new context in which ***Wind_direction*** and ***Wind_speed*** are defined as subtypes of ***Attr_value***.

Wind_direction and ***Wind_speed*** are just two examples of many different specialised values for the ***Attr_value.*** In the two subsequent refinements we have introduced many other examples of similar cases. From these refinements, it is necessary to amend the ***Update*** event to ensure that only values of the correct type update the database. Previously, ***SET_DATA_VALUE*** updated any attribute identifier with any attribute value. Now it must be refined in such a way to avoid having a collection of events, each referring to specific attribute identifiers and attribute values. Again here by the use of Constant mapping we have introduced a matching function to eliminate this barrier.

4.2. Description of the Achieved Results - Distributed Version

As it has been stated earlier the lack of formal connection between the idealised specification and the more realistic design, was one the weaknesses of the initial CDIS development. In this section we describe our attempt to construct a more realistic specification which unlike initial VVL specification can be liked to the distributed design. Using the experiences that we have achieved by developing the idealised version, we have developed a set on new models which includes both horizontal and vertical refinements. As usual these models include one specification and few refinement levels.

4.2. 1. Abstract Specification for Distributed Version of CDIS

This specification is an extension of the idealised version. We have introduced a history tracking system both for the transient data and the page layouts and selections. In a realistic system it is possible that different terminals in different user position have a different view of both transient data and page layout in a specific time. These differences arise due to delays in the system.

In our distributed specification we have assumed that we can model the actual behaviour of the system by storing the history of all applied changes over the time. By have the past history of all changes it is possible to model a system which allows different terminals to have different view of the system data.

4.2. 2. Refinements of Distributed CDIS

After devising this specification of the CDIS which allow tackling issues that arise in a distributed system, there are two methods to refine the initial specification. One possible approach is to apply all subsequent refinement of the idealised version before applying any vertical refinement. An expected advantage of this approach is the similarities between the generated proof obligations. This can assist the developers to discharge the interactive proof obligation more easily. After completing all horizontal refinements in which we introduce new requirements we can proceed to the vertical refinement. During this stage we have replaced the history tracking mechanism by a system which comprises a central database, a history of only changed data and a local database for each viewing position. The main advantage of this approach for implementation viewpoint is that storing the history of applied changes should need much smaller memory in comparison of storing the whole database history.

Another approach to the refinement of the distributed specification is that the vertical refinement precedes horizontal refinements. We have not pursued this path because we believe that the vertical refinement is a design decision and we should not mix specification with design decisions.

5. Overview of Achieved Results and Conclusions

A very different methodology and modelling style was adopted in the Event-B development than in the original VVSL development. The original VVSL development produced a single large specification that required a lot of effort to comprehend and impossible to reason about using the technology available at the time. In the Event-B development first we have focused our effort on tackling the comprehensibility issue and the issue of mechanical proof. We quickly found that both these issues could be tackled by using refinement to layer in the functionality of the system in series of steps rather than trying to model all the functionality in one large specification.

The layered development helped the comprehensibility considerably because we were able to capture the essential functionality of the system in the abstract specification. The abstract model is just under 4 pages of Event-B and we claim that this abstract model allows the reader to quickly grasp the essence of the system. Six subsequent refinements were used to introduce additional features of the system. The nature of these refinements was that they added additional details to the information structures and placed further constraints on when various actions could happen. The layered nature of their introduction means they can be absorbed in a stepwise fashion thus easing comprehensibility.

In the second stage we focused on the issue of distribution and by having the experience of the idealised version we developed this extended version very smoothly. We have successfully overcome this problem by extending the idealised version to a more realistic specification. We have demonstrated that this specification could be refined to a distributed model with a reasonable effort in the context of the Rodin tool.

We found the Rodin tool support for mechanical proof very helpful. All proofs were carried out using the Rodin tool and we found its prover very powerful. The layered development eased the proof of consistency of the specification since at each step we had a small number of relatively simple proof obligations. In addition to the consistency proofs the Rodin tool now is capable of handling wider forms of proof such as Well-Definedness. All of these increase the confident in to the produced system.

References

[1] C. Jones: Systematic Software Development using VDM, Prentice Hall, 1990.
[2] C. A. Middleburg: VVSL: A Language for Structured VDM Specifications, Formal Aspects of Computing, Vol. 1, No. 1, Springer, 1989.
[3] S. L. Pfleeger and L Hatton: Investigating the Influence of Formal Methods, Computer, Vol. 30, No. 2, IEEE, February 1997.
[4] RODIN Deliverable D4: Traceable Requirements Document for Case Studies, http://rodin.cs.ncl.ac.uk/deliverables/D4.pdf, 2005.
[5] Jean-Raymond Abrial. The B-Book: Assigning Programs to Meanings, Cambridge University Press, 1996.
[6] RODIN Deliverable D7: Event B language, http://rodin.cs.ncl.ac.uk/deliverables/D7.pdf
[7] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede and Laurent Voisin: An Open Extensible Tool Environment for Event-B, ICFEM, 2006, Springer, LNICS, Pages 588-605.
[8] Rodin platform: http://rodin-b-sharp.sourceforge.net/
[9] Neil Evans and Michael Butler: A Proposal for Records in Event-B, FM 2006, Springer, LNICS, Pages 221-235.