Modeling and Validating SAFER in VDM-SL*

Sten Agerholm and Peter Gorm Larsen
IFAD
Forskerparken 10, DK-5230 Odense M, Denmark
Email: {sten,peter}@ifad.dk

Abstract

Formal methods can be applied with different levels of rigor. The more rigorously used, the more confidence is obtained in a formal model of a computer system. However, rigorous development using formal verification requires skilled personnel and is costly. Based on our experience of introducing formal specification to some European industrial companies, e.g. British Aerospace [7] and Aerospatiale [3], we believe that a less rigorous approach using validation by testing is a complement to formal verification, which engineers can use cost-effectively early in their formal methods careers. When they become more confident with constructing formal models, it would be natural to take the next step and introduce verification. In this paper we illustrate how testing-based validation can be applied to the SAFER example used throughout [9].

1 Introduction

Historically, NASA's involvement in formal methods has concentrated on formal verification using mechanical theorem provers [1, 2, 9]. In technology transfer projects supported by NASA such formal verification has been in focus. This is clearly understandable considering the criticality of the systems being developed in the avionics sector. However, we believe that the technology transfer process is more likely to have significant effect if the formal methods technology is incorporated in smaller "deltas" into the existing practice [5]. We only consider the transfer fully successful when the engineers normally developing the avionics systems, rather than the formal methods experts, feel that the technology is accessible to them and can be applied in a cost-effective manner.

With formal methods, engineers construct abstract models of computer systems before they start coding them. Among others, many of NASA's technology transfer projects have shown that benefits can be obtained already by formally specifying a system, but validation by testing and verification can increase confidence in models. Software engineering practice is heavily based on testing, and so this technique is well-known to engineers. We therefore advocate the use of testing to validate formal models. If animation is supported by tools, this prototyping activity also supports the presentation of models to others. A further advantage is that the formal specification can also be used as a basis for verifying interesting properties. Verification requires more skilled personnel than specification, and thus we see the technology transfer as a two step process. When the engineers feel confident in using formal specification and testing, it seems feasible to introduce verification.

In this paper we illustrate how a testing-based validation approach can be applied to the SAFER[1] example from [9] using VDM-SL [10] and the IFAD VDM-SL Toolbox [8, 4]. In [9], SAFER is specified in the PVS notation and properties are proved using the PVS theorem prover [11]. The VDM-SL notation is quite close to the PVS notation, but it is not the notational differences we find interesting. Instead, we focus on the kind of analysis that can be performed using a validation by testing approach rather than formal verification. This analysis reveals some interesting "unclarities" in [9]. To support the testing approach, we exploit the Dynamic Link facility of the Toolbox for rapid prototyping [6], linking graphical visualization models (compiled code) with our executable specification. We believe that this kind of animation support is very useful, in particular for presenting a specification to customers and non-trained staff members

*To appear in *Proceedings of the Fourth NASA Langley Formal Methods Workshop (Lfm97)*, September 1997. See http://atb-www.larc.nasa.gov/Lfm97/.

[1] SAFER is an acronym for "Simplified Aid For EVA (Extravehicular Activity) Rescue".

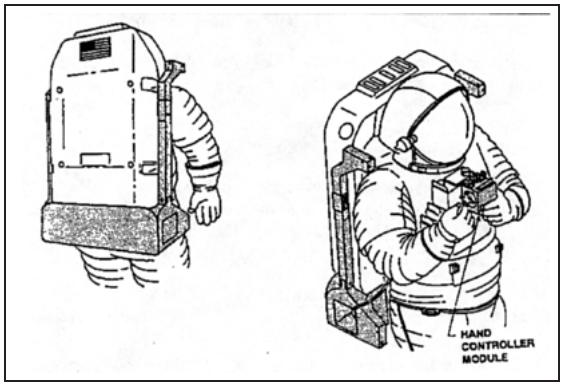

Figure 1: Front and back views of SAFER system worn by NASA crewmember.

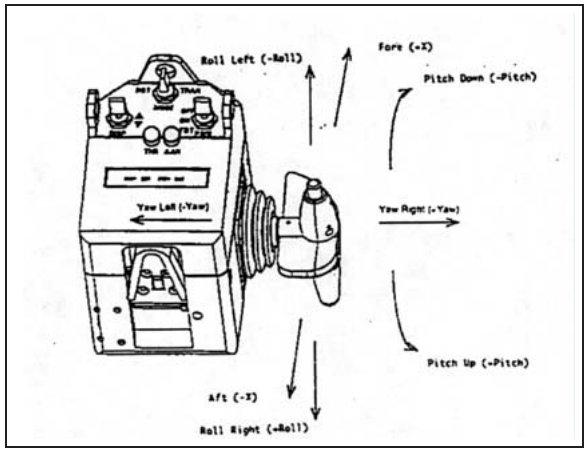

Figure 2: SAFER hand controller.

(e.g. management).

The remaining part of this paper starts with an overview of the SAFER system and the VDM-SL specification of SAFER. This is followed by a section in which we illustrate how a validation approach can be used to investigate different properties. Finally, a few concluding remarks are provided. The reader is assumed to have a basic knowledge of VDM-SL or a similar notation such as PVS.

2 Overview of the SAFER System

This overview of the SAFER system is based on, and partly copied from, the NASA report [9], which describes a cut-down version of a real SAFER system.

SAFER is a small, lightweight propulsive backpack system designed to provide self-rescue capabilities to a NASA space crewmember separated during an EVA (Extravehicular Activity). This type of contingency can arise if a safety tether breaks, or if it is not correctly fastened, during an EVA on a space station or on a Space Shuttle Orbiter docked to a space station. SAFER attaches to the underside of the Extravehicular Mobility Unit (EMU) primary life support subsystem backpack and is controlled by a single hand controller that is attached to the EMU display and control module (see Figure 1). SAFER provides an attitude hold capability and sufficient propellant power to automatically detumble and manually return a separated crewmember.

The SAFER system works by firing 24 gaseous-nitrogen (GN_2) thrusters, four in each of the positive and negative X, Y and Z directions. These are placed on the upper and lower edges of the grey part of the backpack in Figure 1 (see also Figure 13). The main focus of our specification is on the thruster selection logic, for example taking into account that translational hand controller commands are prioritized, with the priority order being X, Y and Z. Other aspects of the SAFER are ignored, e.g. the calculation of control output in the Automatic Attitude Hold (AAH), and various display units and switches on the hand controller which are not directly related to the selection of thrusters.

The hand controller is a four-axis mechanism with three rotary axes and one transverse axis using a certain hand controller grip (see Figure 2). A command is generated by moving the grip from the center null position to mechanical hardstops on the hand controller axes. Commands are terminated by returning the grip to the center position. The hand controller can operate in two modes, selected via a switch, either in translation mode, where X, Y, Z and *pitch* commands are available, or in rotation mode, where *roll*, *pitch*, *yaw* and X commands are available (see Figure 3). Note that X and *pitch* commands are available in both modes. *Pitch* commands are issued by twisting the hand grip around its transverse axis, while the other commands are obtained around the rotary axes.

A push-button switch on top of the grip initiates and terminates AAH according to a certain protocol. If the button is pushed down once the AAH is initiated, while the AAH is deactivated if the button is pushed down twice within 0.5 seconds.

As indicated above there are various priorities

among commands. These make the thruster selection logic slightly complicated. Translational commands issued from the hand controller are prioritized to provide acceleration along a single translational axis, with the priority being X first, Y second, and Z third. When rotation and translation commands are present simultaneously from the hand controller, rotations take priority and translations are suppressed. Moreover, rotational commands from the hand grip takes priority over control output from the AAH, and the corresponding rotation axes of the AAH remain off until the AAH is reinitialized. However, if hand grip rotations are present at the time when the AAH is initiated, the corresponding hand controller axes are subsequently ignored, until the AAH is deactivated.

In Appendix C.2 of the NASA report, a number of requirement properties for SAFER are listed. Below we list a few relevant ones for our specification of the SAFER, and in Section 4.3 we discuss some scenarios which, among others, could be used to test that the specification satisfies the requirements.

(18) The pushbutton switch shall activate AAH when depressed a single time.

(19) The pushbutton switch shall deactivate AAH when pushed twice within 0.5 seconds.

(37) The avionics software shall disable AAH on an axis if a crewmember rotation command is issued for that axis while AAH is active.

(38) Any hand controller rotation command present at the time AAH is initiated shall subsequently be ignored until a return to the off condition is detected for that axis or until AAH is disabled.

(39) Hand controller rotation commands shall suppress any translation commands that are present, but AAH-generated rotation commands may coexist with translations.

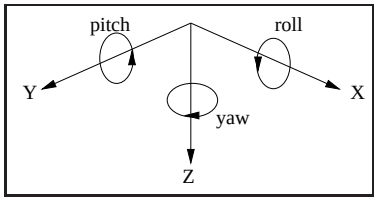

Figure 3: Six degree-of-freedom commands. Arrow direction is positive.

(40) At most one translation command shall be acted upon, with the axis chosen in priority order X, Y, Z.

(41) The avionics software shall provide accelerations with a maximum of four simultaneous thruster firing commands.

These may seem straightforward and clear, but the discussion in Section 4.3 below will show that in fact they are not.

3 VDM-SL Specification of SAFER

In [9], a PVS specification is provided of a cut-down version of SAFER. We have translated this specification to VDM-SL, a relatively straightforward task since the two notations share many constructs. We have further simplified the model by cutting out parts which are irrelevant to the thruster selection logic, such as display units and interface functions to external sensors. These parts were only modeled in a very implicit way in PVS anyway, using e.g. uninterpreted functions.

We present some excerpts from the VDM-SL specification. We will make this presentation independent of the module structure of the specification, which contains five modules: *AUX* (auxiliary definitions), *HCM* (hand controller module), *AAH* (automatic attitude hold), *TS* (thruster selection) and *SAFER* (main control cycle). Modules may use definitions from other modules by prefixing with the module name, e.g. *HCM`HandGripPosition* and *AUX`RotCommand*.

3.1 Main Control Cycle

The SAFER system works by calculating thruster settings very frequently. As in the PVS model we assume that this happens by iteratively calling the main control operation, called *ControlCycle*. The main purpose of this operation is to calculate the thruster settings according to the hand controller commands and AAH control output, and to maintain the state of the AAH protocol. The operation is defined as shown in Figure 4[2].

VDM-SL supports both a functional subset, where a state can only be modeled indirectly in

[2] The notation mk-$R(\ldots)$ is used to construct an element of a record type R.

```
ControlCycle : HCM`SwitchPositions × HCM`HandGripPosition × AUX`RotCommand →o
               TS`ThrusterSet
ControlCycle(mk-HCM`SwitchPositions(mode, aah), raw-grip, aah-cmd) △
  let grip-cmd = HCM`GripCommand(raw-grip, mode),
      thrusters = TS`SelectedThrusters(grip-cmd, aah-cmd, AAH`ActiveAxes(), AAH`IgnoreHcm()) in
  (AAH`Transition(aah, grip-cmd, clock);
   clock := clock + 1;
   return thrusters )
```

Figure 4: The *ControlCycle* operation.

```
SelectedThrusters : AUX`SixDofCommand × AUX`RotCommand × AUX`RotAxis-set ×
                    AUX`RotAxis-set → ThrusterSet
SelectedThrusters(hcm, aah, active-axes, ignore-hcm) △
  let mk-AUX`SixDofCommand(tran, rot) = IntegratedCommands(hcm, aah, active-axes, ignore-hcm),
      mk-(bf-mandatory, bf-optional) = BFThrusters(tran(X), rot(PITCH), rot(YAW)),
      mk-(lrud-mandatory, lrud-optional) = LRUDThrusters(tran(Y), tran(Z), rot(ROLL)),
      bf-thr = if rot(ROLL) = ZERO
               then bf-optional ∪ bf-mandatory
               else bf-mandatory,
      lrud-thr = if rot(PITCH) = ZERO ∧ rot(YAW) = ZERO
                 then lrud-optional ∪ lrud-mandatory
                 else lrud-mandatory in
  bf-thr ∪ lrud-thr
```

Figure 5: The *SelectedThrusters* function.

the signature of functions, and an imperative subset which supports states, operations and programming language statements directly. Here, *ControlCycle* is defined as an operation and not a function, since it depends on the AAH state (see Section 3.3) and the SAFER state, which is simply a natural number valued clock (as in [9] we do not use a real-time clock):

```
state SAFER of
  clock : ℕ
  init s △ s = mk-SAFER(0)
end
```

The result of a call to *ControlCycle* is a set of thruster settings, i.e. a set of thruster names to be turned on. The arguments of *ControlCycle* are: (1) switch positions on the hand controller, telling whether the mode is translation or rotation and whether the AAH button on the grip is up or down, (2) hand grip positions, a record containing four fields corresponding to the transverse axis and the three rotation axes, and (3) external input from the AAH control laws whose calculation is based on data measured by sensors. This AAH control function in (3) is not modeled either in the PVS specification of [9], though it is mentioned in a very implicit way.

The control cycle first transforms the "raw" grip commands to translation and rotation commands. Next it calculates the thruster settings and then the AAH state is updated in the body of the let-statement. We treat each of these steps in separate subsections below. Note that values of AAH state variables are fetched by calling the operations *ActiveAxes* and *IgnoreHcm*, which both return sets of rotation axes.

3.2 Thruster Selection

The six degree-of-freedom of the translation and rotation commands is modeled using a record type:

```
SixDofCommand :: tran : TranCommand
                 rot  : RotCommand;
```

whose two fields are finite maps, i.e. a kind of tables, from translation and rotation axes respectively to axis commands. For example, the type of translation commands is defined as follows:

```
BFThrusters : AUX'AxisCommand × AUX'AxisCommand × AUX'AxisCommand →
                ThrusterSet × ThrusterSet
BFThrusters (A, B, C) ≜
  cases mk-(A, B, C) :
    mk-(NEG, ZERO, ZERO)  → mk-({B1, B4}, {B2, B3}),
    mk-(ZERO, ZERO, ZERO) → mk-({}, {}),
    mk-(POS, NEG, ZERO)   → mk-({F1, F2}, {}),
    ...                   → ...
  end;

LRUDThrusters : AUX'AxisCommand × AUX'AxisCommand × AUX'AxisCommand →
                  ThrusterSet × ThrusterSet
LRUDThrusters (A, B, C) ≜
  cases mk-(A, B, C) :
    mk-(NEG, NEG, ZERO)  → mk-({}, {}),
    mk-(NEG, ZERO, ZERO) → mk-({L1R, L3R}, {L1F, L3F}),
    mk-(POS, ZERO, POS)  → mk-({R2R}, {R2F, R4F}),
    ...                  → ...
  end;
```

Figure 6: Extracts from *BFThruster* and *LRUDThrusters*.

$TranCommand = TranAxis \xrightarrow{m} AxisCommand$

inv $cmd \triangleq \mathsf{dom}\ cmd = \{X, Y, Z\}$;

where the invariant ensures that command maps are total. The type of rotation commands is defined similarly. Enumerated types are used for axis commands and translation and rotation axes:

$AxisCommand = \text{NEG} \mid \text{ZERO} \mid \text{POS}$;

$TranAxis = X \mid Y \mid Z$;

$RotAxis = \text{ROLL} \mid \text{PITCH} \mid \text{YAW}$;

In the *SelectedThrusters* function in Figure 5 grip commands from the hand controller (with six-degree-of freedom) are integrated with the AAH control output. Then thrusters for back and forward accelerations and left, right, up and down accelerations are calculated by two separate functions. These represent a kind of look-up tables, modeled using cases expressions. Note that they return two sets of thruster names, representing mandatory and optional settings respectively. Cut-down versions of the functions are presented to save space, see Figure 6. Note that thrusters are named according to which direction they provide acceleration for, and the number indicates which of four quadrants they are positioned in. The third letter indicates whether the thruster is positioned to the rear or front in a quadrant (see Figure 13 below). The first case in *LRUDThrusters* yields two empty sets since it is a "Not Applicable" case.

The more interesting parts of the specification can be found in the *IntegratedCommands* function and the auxiliary functions it uses. Together these define the selection logic, modeling for instance the various priorities among translation and rotation axes. Below, null translation and rotation commands map all axes to ZERO.

The *IntegratedCommands* function is presented in Figure 7. The function treats two cases, depending on whether or not the set of active axes in the AAH is empty (tested in *AllAxesOff*). If there are rotation commands from the hand controller then these take priority over translation commands using the *PrioritizedTranCmd* function (see Figure 7). Otherwise translation commands are prioritized in the order *X*, *Y*, *Z*.

If there are rotation commands from both the AAH and the hand controller then these must be combined such that the hand controller commands take priority, unless they were also issued when the AAH was initiated (recorded in *ignore-hcm*). This is done by the *CombinedRotCmds* function (see Figure 7). Note that the $\cup^m$ operator used in the *CombinedRotCmds* function is simply a union between maps. *AUX'rot-axis-set* is defined as the set of all rotation axes. This completes the description of the thruster selection logic.

$$
\begin{array}{l}
IntegratedCommands : AUX`SixDofCommand \times AUX`RotCommand \times AUX`RotAxis\text{-set} \times \\
\qquad AUX`RotAxis\text{-set} \to AUX`SixDofCommand \\
IntegratedCommands(\mathsf{mk}\text{-}AUX`SixDofCommand(tran, rot), aah, active\text{-}axes, ignore\text{-}hcm) \triangleq \\
\quad \mathsf{if}\ AAH`AllAxesOff(active\text{-}axes) \\
\quad \mathsf{then\ if}\ RotCmdsPresent(rot) \\
\qquad \mathsf{then}\ \mathsf{mk}\text{-}AUX`SixDofCommand(AUX`null\text{-}tran\text{-}command, rot) \\
\qquad \mathsf{else}\ \mathsf{mk}\text{-}AUX`SixDofCommand(PrioritizedTranCmd(tran), AUX`null\text{-}rot\text{-}command) \\
\quad \mathsf{else\ if}\ RotCmdsPresent(rot) \\
\qquad \mathsf{then}\ \mathsf{mk}\text{-}AUX`SixDofCommand(AUX`null\text{-}tran\text{-}command, \\
\qquad\qquad CombinedRotCmds(rot, aah, ignore\text{-}hcm)) \\
\qquad \mathsf{else}\ \mathsf{mk}\text{-}AUX`SixDofCommand(PrioritizedTranCmd(tran), aah); \\
\\
PrioritizedTranCmd : AUX`TranCommand \to AUX`TranCommand \\
PrioritizedTranCmd(tran) \triangleq \\
\quad \mathsf{if}\ tran(\mathrm{X}) \neq \textsc{Zero} \\
\quad \mathsf{then}\ AUX`null\text{-}tran\text{-}command \dagger \{\mathrm{X} \mapsto tran(\mathrm{X})\} \\
\quad \mathsf{elseif}\ tran(\mathrm{Y}) \neq \textsc{Zero} \\
\quad \mathsf{then}\ AUX`null\text{-}tran\text{-}command \dagger \{\mathrm{Y} \mapsto tran(\mathrm{Y})\} \\
\quad \mathsf{elseif}\ tran(\mathrm{Z}) \neq \textsc{Zero} \\
\quad \mathsf{then}\ AUX`null\text{-}tran\text{-}command \dagger \{\mathrm{Z} \mapsto tran(\mathrm{Z})\} \\
\quad \mathsf{else}\ AUX`null\text{-}tran\text{-}command; \\
\\
CombinedRotCmds : AUX`RotCommand \times AUX`RotCommand \times AUX`RotAxis\text{-set} \to AUX`RotCommand \\
CombinedRotCmds(hcm\text{-}rot, aah, ignore\text{-}hcm) \triangleq \\
\quad \mathsf{let}\ aah\text{-}axes = ignore\text{-}hcm \cup \{a \mid a \in AUX`rot\text{-}axis\text{-}set \cdot hcm\text{-}rot(a) = \textsc{Zero}\}\ \mathsf{in} \\
\quad \{a \mapsto aah(a) \mid a \in aah\text{-}axes\} \overset{\mathrm{m}}{\cup} \{a \mapsto hcm\text{-}rot(a) \mid a \in AUX`rot\text{-}axis\text{-}set \setminus aah\text{-}axes\};
\end{array}
$$

Figure 7: The *IntegratedCommands* function and its auxiliary functions.

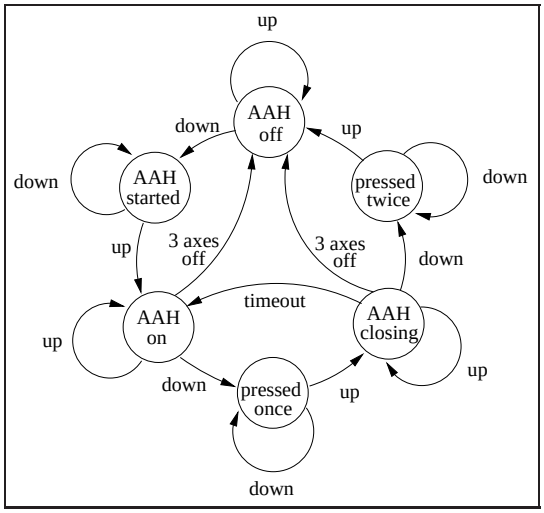

Figure 8: The AAH protocol state machine.

$$
\begin{array}{l}
Transition : HCM`ControlButton \times AUX`SixDofCommand \times \mathbb{N} \xrightarrow{o} () \\
Transition(button\text{-}pos, hcm\text{-}cmd, clock) \triangleq \\
\quad \mathsf{let}\ engage = ButtonTransition(toggle, button\text{-}pos, active\text{-}axes, clock, timeout), \\
\qquad starting = (toggle = \text{AAH\_OFF}) \wedge (engage = \text{AAH\_STARTED})\ \mathsf{in} \\
\quad (active\text{-}axes := \{a \mid a \in AUX`rot\text{-}axis\text{-}set \cdot starting \vee \\
\qquad\qquad (engage \neq \text{AAH\_OFF} \wedge a \in active\text{-}axes \wedge \\
\qquad\qquad\ (hcm\text{-}cmd.rot(a) = \text{ZERO} \vee a \in ignore\text{-}hcm))\}; \\
\qquad ignore\text{-}hcm := \{a \mid a \in AUX`rot\text{-}axis\text{-}set \cdot (starting \wedge hcm\text{-}cmd.rot(a) \neq \text{ZERO}) \vee \\
\qquad\qquad (\neg\, starting \wedge a \in ignore\text{-}hcm)\}; \\
\qquad timeout := \mathsf{if}\ toggle = \text{AAH\_ON} \wedge engage = \text{PRESSED\_ONCE} \\
\qquad\qquad\quad \mathsf{then}\ clock + click\text{-}timeout \\
\qquad\qquad\quad \mathsf{else}\ timeout; \\
\qquad toggle := engage);
\end{array}
$$

Figure 9: The *Transition* operation.

3.3 AAH Transitions

Finally, we present excerpts from the AAH module. The AAH push-button protocol can be viewed as a state machine, presented in Figure 8. In VDM-SL, the state of the AAH is represented by:

$$
\begin{array}{l}
\mathsf{state}\ AAH\ \mathsf{of} \\
\quad active\text{-}axes : AUX`RotAxis\text{-}\mathrm{set} \\
\quad ignore\text{-}hcm : AUX`RotAxis\text{-}\mathrm{set} \\
\quad toggle\ \ \ : EngageState \\
\quad timeout\ \ \ : \mathbb{N} \\
\quad \mathsf{init}\ s \triangleq s = \mathsf{mk}\text{-}AAH(\{\}, \{\}, \text{AAH\_OFF}, 0) \\
\mathsf{end}
\end{array}
$$

where the actual state machine states are presented as an enumerated type:

$$
\begin{array}{l}
EngageState = \text{AAH\_OFF} \mid \text{AAH\_STARTED} \mid \\
\qquad\qquad\ \text{AAH\_ON} \mid \text{PRESSED\_ONCE} \mid \\
\qquad\qquad\ \text{AAH\_CLOSING} \mid \text{PRESSED\_TWICE}
\end{array}
$$

The main operation in the AAH module is *Transition* which is defined in Figure 9. It uses the function *ButtonTransition*, a direct encoding of Figure 8, to calculate the next state. Note that *Transition* uses a *starting* predicate for the situation when the AAH is initiating. The starting predicate is used to calculate the active axes and axes for which the hand controller must be ignored.

4 Validation Using Testing

The NASA guidebook [9] focuses mainly on formal verification as a method for analysis of requirements and high-level design. However, we find that alternative approaches to validating specifications based on testing techniques are also beneficial, though they should not replace verification. From a technology transfer viewpoint, testing is cheaper to introduce, but in most situations it is also cheaper to apply than verification and would therefore, in particular, be suitable for the early stages of analysis like model construction.

In this section we illustrate the use of testing techniques to validate the VDM-SL model of SAFER presented above. Our specification is written in the executable subset of VDM-SL, which is supported by the interpreter of the IFAD VDM-SL Toolbox. We shall illustrate the use of a test coverage facility of the Toolbox, and we shall try to use testing techniques and the Dynamic Link facility to investigate the properties listed in Section 2.

4.1 Test Coverage of Specifications

In addition to basic facilities for checking specifications, such as syntax and type checking, the IFAD Toolbox supports a large executable subset of VDM-SL. This means that the Toolbox supports validation techniques such as testing which are usually not encouraged by theorem provers. We shall not turn this into an exercise in test case selection, but limit our attention to just a few test cases for illustration.

In testing, the focus is often on more concrete properties than in verification; given some input we test whether some function computes the desired result. This kind of testing, which to a large extent also could be performed by theorem provers, e.g. if rewriting is available, is not considered nor mentioned in [9]. However, it does make sense to test, for instance, that thrusters are fired correctly according to hand grip commands. The actual selection is based on some fairly large tables which have

```
SAFER`ControlCycle(mk-HCM`SwitchPositions(TRAN, UP),
                   mk-HCM`HandGripPosition(ZERO, POS, ZERO, ZERO), AUX`null-rot-command)
-------------------------------------------------------------------------------------
{mk-(mode, a, b, c, d) ↦ SAFER`ControlCycle(mk-HCM`SwitchPositions(mode, UP),
                                            mk-HCM`HandGripPosition(a, b, c, d), AUX`null-rot-command)
| mode ∈ {TRAN, ROT}, a, b, c, d ∈ {NEG, POS, ZERO}}
```

Figure 10: Compact test expressions.

```
LRUDThrusters : AUX`AxisCommand × AUX`AxisCommand × AUX`AxisCommand →
                ThrusterSet × ThrusterSet
LRUDThrusters(A, B, C) ≜
  cases mk-(A, B, C) :
    mk-(NEG, NEG, ZERO) → mk-({}, {}),
    mk-(NEG, ZERO, ZERO) → mk-({L1R, L3R}, {L1F, L3F}),
    mk-(POS, ZERO, POS)  → mk-({R2R}, {R2F, R4F}),
    ...                  → ...
  end;
```

Figure 11: Extracts from *LRUDThrusters* with test coverage information.

been translated by hand to the specification, and of course such translation could have introduced errors.

Test expressions can be typed in manually for the interpreter. For example, in the first expression in Figure 10 a call of *ControlCycle* with a forward acceleration command is issued. The interpreter will immediately evaluate this to the correct result, which is the set {F1,F2,F3,F4}.

If one wishes to test all hand grip commands without typing all combinations in by hand, then a compact test expression could be written in VDM-SL using a map comprehension expression as shown as the second expression in Figure 10. Assuming that the AAH is switched off, this will cover all hand grip translational and rotational commands, in total $2 * 3^4 = 162$ cases. This is executed in seconds. The result is a large map from variable values to results (thruster settings). Interestingly, this test only yields 17 different settings and does not cover the functions *BFThrusters* and *LRUDThrusters* very well; recall these convert six degree-of-free commands to thruster names. A test coverage facility is provided with the Toolbox, which computes basic statistical information and colors uncovered parts of specifications as illustrated in Figure 11. The two thruster functions are both called 162 times, but they are only covered 41% and 46% respectively.

The reason for this bad coverage is due to priorities for hand controller translational axes and for rotation and translation commands from the hand controller. A larger test where the AAH pushbutton and rotation command output are also variable (8748 cases, executed in 7 minutes) yields 189 different thruster settings and covers the *BFThrusters* function 100%. However, the *LRUDThrusters* function is still only 72% covered. But the uncovered parts are precisely those that have the "Not Applicable" label in the requirement specification from [9], so this is good. The test coverage coloring shows this very clearly in the boxes in Figure 11 (we have cut the specification down to the same cases as in Section 3.2). The sub-expressions written in grey have never been reached in the interpretation of the test argument(s). Note that in principle it would be possible to manually inspect the 8748 test results to check that the thruster settings are correct, but this may not be feasible in all situations. Also note that the tests mentioned above only test the AAH state machine protocol in an arbitrary way, and therefore the AAH transition function is not fully covered by this kind of testing.

4.2 Using Dynamic Link Modules

Such tests as described above could be difficult to understand for staff members who have not been trained in the notation used for the formal specification. The value of the model would also increase

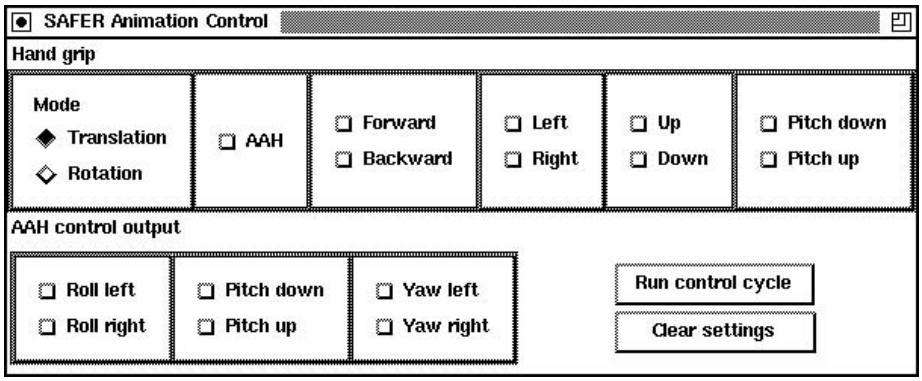

Figure 12: The interface model of the hand controller.

in the discussion with someone outside the development team, such as a customer, if some kind of prototype could be produced where the interface to the formal model is easier to understand.

In order to ease this kind of testing we can exploit the Dynamic Link facility of the IFAD Toolbox for combining compiled code and VDM-SL specifications [6]. We have made a simple interface model of the hand controller using Tcl/Tk and linked this with the SAFER specification. Figure 12 shows a screen dump of the interface, where the mode (translational or rotational) and the AAH switch together form the first parameter to *ControlCycle*. The hand grip positions form the second parameter, and the AAH control outputs form the third parameter simulating the external input from the AAH control laws whose calculation would be performed from sensor data of movements of the astronaut.

Moreover, a simplified model of the SAFER backpack in a 3D Graphics tool called Geomview[3] is also linked to the specification and used to visualize thruster settings (see Figure 13). This simple figure could naturally be enhanced significantly, but our purpose here is simply to illustrate the feasibility of this approach. For example, if we move the hand grip forward, we expect to see the forward thrusters fire in the backpack model. Hence, we can very easily make basic tests of the SAFER thruster selection logic. This animation approach is useful for testing many requirement properties of SAFER (see Section 4.3).

If effort would be put into calculating how the astronaut would be moving depending upon the thrusters fired then naturally a more advanced Dynamic Link module could be made to show the movement in 3D. We have a naive solution to this based on a simple model of astronaut movements in space.

[3]Geomview is available at `http://www.geom.umn.edu/docs/software/viz/geomview/geomview.html`.

4.3 Validating Properties

We shall now discuss a testing approach to validating, i.e. checking but not verifying, that some of the requirement properties listed in Section 2 hold in our VDM-SL model; we do not have room in this paper to consider all of them. The testing raised interesting questions regarding the requirements and found various omissions/problems in our specification as well as in the PVS specification.

4.3.1 Automatic Checking of Property (41)

Recall property (41):

> The avionics software shall provide accelerations with a maximum of four simultaneous thruster firing commands.

Consider also:

E1 Thruster firing consistency: No two selected thrusters should oppose each other, i.e., have canceling thrust with respect to the center of mass. (Mentioned in Section C.4.1 of the NASA report.)

These are properties that should hold for all thruster settings output from the SAFER *ControlCycle* and therefore they can conveniently be stated as a post-condition on *ControlCycle*, see Figure 14.

The Toolbox interpreter can be requested to automatically check that the post-condition holds for

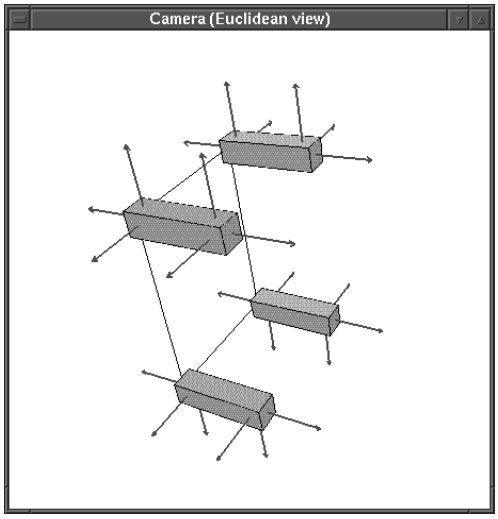

Figure 13: The interface model of the SAFER backpack.

$$
\begin{array}{l}
ControlCycle : HCM`SwitchPositions \times HCM`HandGripPosition \times AUX`RotCommand \xrightarrow{o} TS`ThrusterSet \\
ControlCycle(\mathsf{mk}\text{-}HCM`SwitchPositions(mode, aah), raw\text{-}grip, aah\text{-}cmd) \triangleq \\
\quad \mathsf{let}\ grip\text{-}cmd = HCM`GripCommand(raw\text{-}grip, mode), \\
\qquad thrusters = TS`SelectedThrusters(grip\text{-}cmd, aah\text{-}cmd, AAH`ActiveAxes(), AAH`IgnoreHcm()))\ \mathsf{in} \\
\quad (AAH`Transition(aah, grip\text{-}cmd, clock); \\
\quad\ clock := clock + 1; \\
\quad\ \mathsf{return}\ thrusters\ ) \\
\mathsf{post\ card}\ RESULT \leq 4 \wedge \\
\qquad ThrusterConsistency(RESULT)\ ; \\
\\
ThrusterConsistency : TS`ThrusterName\text{-}\mathsf{set} \to \mathbb{B} \\
ThrusterConsistency(thrusters) \triangleq \\
\quad \neg(\{\mathrm{B1}, \mathrm{F1}\} \subseteq thrusters) \wedge \\
\quad \neg(\{\mathrm{B2}, \mathrm{F2}\} \subseteq thrusters) \wedge \\
\quad \neg(\{\mathrm{B3}, \mathrm{F3}\} \subseteq thrusters) \wedge \\
\quad \neg(\{\mathrm{B4}, \mathrm{F4}\} \subseteq thrusters) \wedge \\
\quad \neg(thrusters \cap \{\mathrm{L1R}, \mathrm{L1F}\} \neq \{\} \wedge thrusters \cap \{\mathrm{R2R}, \mathrm{R2F}\} \neq \{\}) \wedge \\
\quad \neg(thrusters \cap \{\mathrm{L3R}, \mathrm{L3F}\} \neq \{\} \wedge thrusters \cap \{\mathrm{R4R}, \mathrm{R4F}\} \neq \{\}) \wedge \\
\quad \neg(thrusters \cap \{\mathrm{D1R}, \mathrm{D1F}\} \neq \{\} \wedge thrusters \cap \{\mathrm{U3R}, \mathrm{U3F}\} \neq \{\}) \wedge \\
\quad \neg(thrusters \cap \{\mathrm{D2R}, \mathrm{D2F}\} \neq \{\} \wedge thrusters \cap \{\mathrm{U4R}, \mathrm{U4F}\} \neq \{\})
\end{array}
$$

Figure 14: The *ControlCycle* operation and the *ThrusterConsistency* function.

all results from *ControlCycle* (this feature is optional). Hence, if a result failed the check, the interpreter would give a run-time error with position information (line and column) in the specification document. This never happened in any of our tests (including the big test mentioned above).

In [9] a proof of the maximum thruster property (41) is given. The validation made here is not as strong as that, because we would have to test it with all values in a model-checking fashion. However, the validation carried out here has definitely increased our confidence in the maximum thruster property being satisfied with our model.

Point: In cases where a property does not hold, the validation technique can be an efficient way of discovering counter-examples. It can also increase confidence in a model.

4.3.2 Property (18)

> The pushbutton switch shall activate AAH when depressed a single time.

This property is related to the protocol of the AAH given in Figure 8, and thus it may be a good idea to revisit this figure to be able to follow the possible test scenarios. Assume the hand grip is in center position throughout this test; this means that no translation or rotation commands are issued from the hand grip. Initially, the AAH is in the "off state". Thus, initially any AAH control rotation output must be ignored. Using various arbitrary examples we can test that the thruster settings are empty. The AAH is activated by pushing the AAH button down. The AAH goes to the "started state", but it does not go to the "on state" until the button is released. We can run the same and other AAH rotation commands to see that they are now taken into account (and are performed correctly). Note that property (38) would affect this test if a rotation command from the hand grip was issued at the same time as the AAH was initiated.

Point: In this case the test has increased our confidence that property (18) is satisfied, such that it is sufficient to push the AAH button without releasing it again.

4.3.3 Property (19)

> The pushbutton switch shall deactivate AAH when pushed twice within 0.5 seconds.

This property is also related to the AAH protocol so Figure 8 must be used again. The timeout of 0.5 seconds is for test purposes represented as 10 control cycles in our specification. As in [9] we do not represent a real-time clock. We first turn the AAH on by pressing and then releasing the AAH button. We then press and release the AAH button twice within 10 cycles and check that the AAH is turned off by running a number of tests with rotation commands from the AAH. Turn AAH on again, press and release the AAH button once, wait at least 11 cycles, press and release again and check that the AAH still works. Finally, press, release and press the AAH button within 10 cycles. The report [9] is unclear here: Strictly speaking, property (19) says that this should deactivate the AAH while the PVS specification and the AAH state diagram does not deactivate the AAH until the button is released.

Point: Since the word "pushed" (rather than "pushed and released") is used in property (19) our validation have discovered a discrepancy between requirement (19) and both the PVS model and the state transition diagram from Figure 8.

4.3.4 Property (37)

> The avionics software shall disable AAH on an axis if a crewmember rotation command is issued for that axis while AAH is active.

This property is related to the combination of the AAH protocol and the thruster selection logic for rotation commands. We can test this by first inspecting how SAFER behaves when the AAH is switched on and the AAH control output wishes to rotate over an axis. Afterwards we can inspect what happens if the hand grip controller issues a rotation command on the same axis. In this case property (37) requires SAFER to disable the AAH for that axis.

One possible first test scenario for this property could be: Assume the hand grip is in center position (all directions are ZERO). Turn the AAH on (button down and then up) and issue an AAH rotation (control output) on an axis, e.g. roll left. Keep this rotation while making a rotation command on the hand grip, e.g. roll right (in rotation mode). Check the output to see that this takes precedence over AAH. Keep the AAH rotation but center the hand grip. The AAH axis should remain off, as

stated explicitly in Section C.4.1 of the NASA report.

However, when executing the test scenario on our specification, this appears not to be the case. The test shows that the AAH axis still has influence on the selected thrusters, in fact, the result is exactly the same as before the axis was deactivated. The problem is that our model (like the PVS model) implicitly assumes that the AAH control output does not provide a rotation command for an axis which is not active. In our manually generated AAH output, this may however be the case. We could capture this problem using a pre-condition on the SAFER *ControlCycle* operation. In the PVS specification, an obvious place to take this into account would be in a post-condition of the `AAH_control_out` function, which could be done using the subtype facility in PVS, but in [9] this function is implicitly defined and does not give any information whatsoever about implementation requirements. When translating the PVS specification to VDM-SL, we cut out the very implicit `AAH_control_out` and modeled this as the third argument of *ControlCycle* instead. A post-condition on the PVS `AAH_control_out` would therefore have been translated to a pre-condition on *ControlCycle* in our model.

As a consequence of the implicitness concerning the AAH control output, it is not possible to verify property (37) in a black-box fashion in PVS, as discussed above. However, a white-box approach of showing that the "active" flag is properly reset and stays off is possible. But in the NASA report it is not documented anywhere that the `AAH_control_out` should take this flag into account, which then seems to be an omission, at least. As mentioned above, this requirement would be easy to specify in a post-condition, which would enable black-box verification. The verification of other properties is also weakened by this omission. For example:

> Once AAH is turned off for a rotational axis it remains off until a new AAH cycle is initiated.

which appears in Section C.4.1 of the NASA report.

Point: In validating this property we discovered that neither our model nor the PVS model in [9] is sufficiently strong to prove certain desirable properties.

4.3.5 Property (38)

> Any hand controller rotation command present at the time AAH is initiated shall subsequently be ignored until a return to the off condition is detected for that axis or until AAH is disabled.

A question raised immediately here when trying to design a test scenario was: what do they mean by "a return to the off condition"? If one carefully reads the PVS specification, and in particular looks at the way that *ignore-hcm* is updated in *Transition*, it seems clear that the only way for a hand controller rotation command to become reconsidered is by manually disabling the AAH.

Point: This is an example where seeking to come up with a test case for a certain situation discovered an ambiguity in the meaning of this requirement.

5 Concluding Remarks

In this paper we have illustrated the analysis of a formal specification using a testing-based approach to validation. We believe that the ratio between the insight gained by this kind of analysis and the effort spent on the analysis is promising, in particular in a technology transfer context, because less skilled engineers are required for this kind of validation than if formal verification was applied. Moreover, many errors can be found (relatively) cheaply using testing which otherwise might require a great deal of effort to single out using formal verification.

The kind of tool support demonstrated in this paper, including the combined use of a specification executor and Dynamic Link modules, can in our opinion help in making the formal methods technology accessible to more engineers. Specifically with respect to the SAFER example, we believe that the prototyping and animation facilities provide an easy way for a specifier to demonstrate the consequences of the thruster selection logic to someone unfamiliar with formal notations. Moreover, in the SAFER example there is very little trade-off in going from the PVS model to an executable VDM-SL model, because the PVS model is already relatively concrete and essentially executable. Generally speaking, models for verification might be more abstract than executable models to ease verification, and so there could be a trade-off.

We see validation using simulation and formal verification as complementary techniques which can be fruitfully applied in the same project. Given

that the system is sufficiently critical to justify the costs, we imagine that the most productive approach would be to use the validation technology first, and to continue with a formal verification of the properties which cannot simply be tested, after the worst bugs in the model have been removed. These different kinds of analyzes tend to discover different kinds of problems. Hence, though some verification of the PVS specification of SAFER had already been carried out in [9], we were still able to detect a few points that need further clarification using the validation approach (see Section 4.3).

Throughout this paper, we have assumed that verification is costly. However, for certain restricted domains automatic verification using model checking has proved to be feasible, and hence potentially very cheap to apply. One might view the use of such automatic verification tools as a compromise towards fully verified models. Typically, the price for this compromise is a more restrictive notation increasing the effort required to produce models.

In fact, we have continued the experiment above together with Arne Borälv from Logikkonsult, in order to investigate the potential power of automatic verification. Arne Borälv has taken our VDM description and manually translated it to a model in NP-Tools [12], using only propositional logic extended with integer arithmetic and enumerated types. Using this model it was possible to automatically prove the maximum thruster and the thruster consistency requirements, as well as some other properties. Proof execution times were within the order of seconds (45 seconds for the hardest requirement, maximum thruster). For rather finite systems like the SAFER example we envisage that it would be possible to automatically translate VDM models to models in NP-Tools and in this way be able to automatically verify properties fast. This approach seems appealing and will be further investigated.

Acknowledgments

We would like to thank Erik Toubro Nielsen, Ole Storm Pedersen and Anders Søndergaard for developing the different parts of the Dynamic Link modules. We are also grateful for the constructive comments we got on earlier versions of this paper from Hanne Carlsen, Benny Graff Mortensen, Paul Mukherjee and Anne Berit Nielsen. Finally PGL would like to thank John Kelly for asking for review comments on [9].

References

[1] Ricky W. Butler, James L. Caldwell, Victor A. Carreno, C. Michael Holloway, Paul S. Miner, and Ben L. Di Vito. Nasa langley's research and technology transfer program in formal methods. In *Tenth Annual Conference on Computer Assurance (COMPASS 95)*. Gaithersburg, MD, June 1995. (expanded version available from http://atb-www.larc.nasa.gov/fm.html).

[2] James L. Caldwell. Formal methods technology-transfer: a view from nasa. In S. Gnesi and D. Latella, editors, *Proceedings of the ERCIM Workshop on Formal Methods for Industrial Critical Systems*, Oxford England, March 1996.

[3] Lionel Devauchelle, Peter Gorm Larsen, and Henrik Voss. PICGAL: Lessons Learnt from a Practical Use of Formal Specification to Develop a High Reliability Software. In *DASIA'97*. ESA, May 1997.

[4] René Elmstrøm, Peter Gorm Larsen, and Poul Bøgh Lassen. The IFAD VDM-SL Toolbox: A Practical Approach to Formal Specifications. *ACM Sigplan Notices*, 29(9):77–80, September 1994.

[5] J.S. Fitzgerald and P.G. Larsen. Formal specification techniques in the commercial development process. In M. Wirsing, editor, *Position Papers from the Workshop on Formal Methods Application in Software Engineering Practice, International Conference on Software Engineering (ICSE-17), Seattle*, April 1995. ftp://ftp.ifad.dk/pub/papers/icse.ps.gz.

[6] Brigitte Fröhlich and Peter Gorm Larsen. Combining VDM-SL Specifications with C++ Code. In Marie-Claude Gaudel and Jim Woodcock, editors, *FME'96: Industrial Benefit and Advances in Formal Methods*, pages 179–194. Springer-Verlag, March 1996.

[7] Peter Gorm Larsen, John Fitzgerald, and Tom Brookes. Applying Formal Specification in Industry. *IEEE Software*, 13(3):48–56, May 1996.

[8] Paul Mukherjee. Computer-aided Validation of Formal Specifications. *Software Engineering Journal*, pages 133–140, July 1995.

[9] NASA. Formal methods, specification and verification guidebook for software and computer systems – a practitioner's companion. Technical Report Draft 2.0, Washington, DC 20546, USA, November 1996.

[10] P. G. Larsen and B. S. Hansen and H. Brunn N. Plat and H. Toetenel and D. J. Andrews and J. Dawes and G. Parkin and others. Information technology — Programming languages, their environments and system software interfaces — Vienna Development Method — Specification Language — Part 1: Base language, December 1996.

[11] PVS World Wide Web page. `http://www.csl.sri.com/pvs/overview.html`.

[12] Gunnar Stålmarck. A System for Determining Propositional Logic Theorems by Applying Values and Rules to Triplets that are Generated from a Formula, 1989. Swedish Patent No. 467 076 (approved 1992), U.S. Patent No. 5 276 897 (1994), European Patent No. 0403 454 (1995).