# Formalizing New Navigation Requirements for NASA's Space Shuttle*

Ben L. Di Vito

VíGYAN, Inc., 30 Research Drive, Hampton, Virginia 23666, USA

**Abstract.** We describe a recent NASA-sponsored pilot project intended to gauge the effectiveness of using formal methods in Space Shuttle software requirements analysis. Several Change Requests (CRs) were selected as promising targets to demonstrate the utility of formal methods in this demanding application domain. A CR to add new navigation capabilities to the Shuttle, based on Global Positioning System (GPS) technology, is the focus of this industrial usage report. Portions of the GPS CR were modeled using the language of SRI's Prototype Verification System (PVS). During a limited analysis conducted on the formal specifications, numerous requirements issues were discovered. We present a summary of these encouraging results and conclusions we have drawn from the pilot project.

## 1 Introduction

Among all the software developed by the U.S. National Aeronautics and Space Administration, Space Shuttle flight software is generally considered exemplary. Nevertheless, much of the quality assurance activity in early lifecycle phases remains a manual exercise in need of more precise analysis techniques. Software upgrades to accommodate new missions and capabilities are continually introduced. Such upgrades underscore the need recognized in the NASA community, and in a recent assessment of Shuttle flight software development, for "state-of-the-art technology" and "leading-edge methodologies" to meet the demands of software development for increasingly large and complex systems [12, p. 91].

Over the last three years, NASA's Langley Research Center (LaRC) has investigated the use of formal methods (FM) in space applications, as part of a three-center demonstration project involving LaRC, the Jet Propulsion Laboratory (JPL), and the Johnson Space Center (JSC). The goal of NASA's Formal Methods Demonstration Project for Space Applications is to find effective ways to use formal methods in requirements analysis and other phases of the development lifecycle. The Space Shuttle program has been cooperating in several pilot projects to apply formal methods to live requirements analysis activities such as the upgrades supporting the recent MIR docking missions, improved algorithms for the newly automated three-engine-out contingency abort maneuvers (3E/O), and the recent optimization of Reaction Control System Jet Selection

* To appear in proceedings of Formal Methods Europe (FME '96), Oxford, England, March 1996.

(JS) [4, 6]. Other programs participating in the demonstration effort include the Cassini deep-space probe and the International Space Station [9, 7].

We focus in this paper on the formal methods-based analysis of a new Global Positioning System (GPS) navigation capability for the Shuttle. This work was performed in the context of a broader program of formal methods activity at LaRC [2]. The effort consisted of formalizing selected Shuttle software (sub)system modifications and additions using the PVS specification language and interactive proof-checker [13]. Our objective was to explore and document the feasibility of formalizing critical Shuttle software requirements.

The key technical results of the project include a clear demonstration of the utility of formal methods as a complement to the conventional Shuttle requirements analysis process. Although proof-based analysis was a goal of the project, the effort has thus far been limited to formalization of the requirements. Nevertheless, the GPS project uncovered anomalies ranging from minor to substantive, many of which were undetected by existing requirements analysis processes. These results corroborate the experiences of others in formalizing requirements [3, 1]. Dissemination of these techniques to the aerospace community should encourage further experimentation [14, 11]. Full details of the GPS study will appear in a forthcoming report [5].

## 1.1 Shuttle Software Background

NASA's prime contractor for the Space Shuttle is the Space Systems Division of Rockwell International. Loral Space Information Systems (formerly IBM, Houston) is their software subcontractor. Draper Laboratory also serves Rockwell, providing requirements expertise in Guidance, Navigation and Control.

Shuttle flight software executes in four redundant general purpose computers (GPCs), with a fifth backup computer carrying dissimilar software. Much of the Shuttle software is organized into major units called *principal functions*, each of which may be subdivided into *subfunctions*. Software requirements are written using conventions known as Functional Subsystem Software Requirements (FSSRs) — low-level software requirements specifications written in English prose with strong implementation biases, and accompanied by pseudo-code, tables, and flowcharts. Interfaces between software units are specified in input-output tables. Inputs can be variables or one of three types of constant data: *I-loads* (fixed for the current mission), *K-loads* (fixed for a series of missions), and physical constants (never changed).

Shuttle software modifications are packaged as Change Requests (CRs), that are typically modest in scope, localized in function, and intended to satisfy specific needs for upcoming missions. Roughly once a year, software releases called Operational Increments (OIs) are issued incorporating one or more CRs. Shuttle CRs are written as modifications, replacements, or additions to existing FSSRs. Loral Requirements Analysts (RAs) conduct thorough reviews of new CRs, analyzing them with respect to correctness, implementability, and testability before turning them over to the development team. Their objective is to identify and correct problems in the requirements analysis phase, avoiding far more costly fixes later in the lifecycle.

## 2 Overview of the Enhanced Shuttle Navigation System

GPS is a satellite-based navigation system operated by the U.S. Department of Defense (DoD), comprising a constellation of 24 satellites in high earth orbits. Navigation is effected using a receive-only technique. Dedicated hardware receivers track four or more satellites simultaneously and recover their signals from the code division multiplexing inherent in their method of transmission. Receivers solve for position and velocity, with a horizontal position accuracy of 100 meters for the Standard Positioning Service mode of operation.

The GPS retrofit to the Shuttle was planned in anticipation of DoD's phase-out of TACAN, a ground-based navigation system currently used during entry and landing. Originally, GPS was required for navigation only during the entry flight phase after the disappearance of TACAN, but the scope has been broadened to cover all mission phases. As one of the larger ongoing Shuttle Change Requests (CRs), the GPS CR involves a significant upgrade to the Shuttle's navigation capability. Shuttles are to be outfitted with GPS receivers and the primary avionics software will be enhanced to accept GPS-provided positions and integrate them into navigation calculations. In particular, the GPS CR will provide the capability to update the Shuttle navigation filter states with selected GPS state vector estimates similar to the way state vector updates currently are received from the ground. In addition, the new functions will provide feedback to the GPS receivers and will support crew control and operation of GPS/GPC processing.

### 2.1 GPS Change Request

The GPS upgrade is being conducted according to a two-phase integration plan. First, a single-string implementation will be carried out involving only a single GPS receiver. After adequate testing, the full-up implementation involving three receivers will provide the operational configuration. Software requirements are structured to accommodate the three-receiver setup from the outset, requiring only minimal changes to go to the full-up version.

Figure 1 shows the integrated architecture for the enhanced navigation subsystem. GPS receivers are managed by the GPS Subsystem Operating Program (SOP), which acts as a device driver. The new principal function GPS Receiver State Processing accepts GPS state vectors, and selects and conditions a usable one for presentation to the appropriate navigation user. Another new principal function, GPS Reference State Processing, maintains reference states for the receivers and navigation functions. Inertial measurement units (IMUs) provide acceleration data and Redundancy Management (RM) functions maintain failure status information.

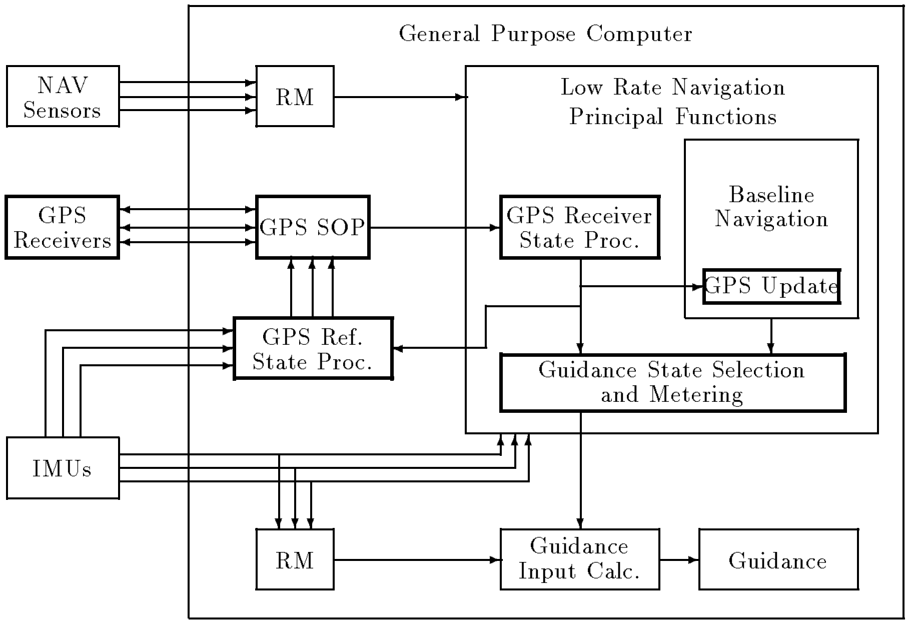

**Fig. 1.** Architecture for integrating GPS into navigation subsystem.

The GPS formalization focused on a few key areas because the CR itself is very large and complex. After preliminary study of the CR and discussions with the GPS RAs, we decided to concentrate on two major new principal functions, emphasizing their interfaces to existing navigation software and excluding crew I/O functions. The two principal functions, known as GPS Receiver State Processing and GPS Reference State Processing, select and modify GPS state vectors for consumption by the existing entry navigation software. As these functions are entirely new, we felt that concentrating on these areas would yield a high return on our formalization investment. Moreover, this choice obviated the need to model large amounts of existing Shuttle functionality.

The two chosen principal functions, in turn, are organized into several subfunctions each.

- GPS Receiver State Processing
  1. GPS IMU Assign
  2. GPS Navigation State Propagation
  3. GPS State Vector Quality Assessment
  4. GPS State Vector Selection
  5. GPS Reference State Announced Reset

    6. GPS Downlist Computation
- GPS Reference State Processing
    1. GPS External Data Snap
    2. IMU GPS Selection
    3. GPS Reference State Initialization and Reset
    4. GPS Reference State Propagation

The subset of the GPS CR represented here contains approximately 110 pages of requirements in the form of prose, pseudo-code, and tables. The entire CR is about 1000 pages long.

### 2.2 Characteristics of Application

The nature of the GPS CR application is that of a significant augmentation to a mature body of complex navigation functions. Interfaces among components are broad, containing many variables. Typical classes of data include:

- Flags to indicate status, to request services, and to select options among processing choices.
- Time values and time intervals both to serve as timestamps within state vectors and to control when operations should be performed.
- Navigation-related values such as positions and velocities.
- Arrays of all these types indexed by GPS receiver number.
- Various numeric quantities representing thresholds, tolerance values, etc.

Navigation state vectors are of the form $(\mathbf{r}, \mathbf{v}, t)$, where $\mathbf{r}$ is a position, $\mathbf{v}$ is a velocity, and $t$ is the time at which the position and velocity apply. A position $\mathbf{r}$ or a velocity $\mathbf{v}$ is a three-element vector relative to a Cartesian or geodetic coordinate system. Usually the Shuttle uses an inertial coordinate system called the "Aries mean of 1950" system, abbreviated as "M50."

An important operation on state vectors is *propagating* them to a new instant of time. If we have a state vector $(\mathbf{r}, \mathbf{v}, t)$, and we have a measurement or estimate of the accelerations experienced by the vehicle over the (short) time interval $[t, t']$, we can propagate the state to a new state vector $(\mathbf{r}', \mathbf{v}', t')$ using standard techniques of physical mechanics. This type of operation is typically performed to synchronize state vectors to a common point in time.

Processing requirements within the CR are generally expressed in an algorithmic style using high-level language assignments and conditional statements. Within conditionally invoked assignments, the assumption is the usual procedural one that a variable not assigned retains its previous value, which may or may not have a meaningful interpretation in the current context. Flag variables are used to indicate when other (non-flag) variables hold currently valid data.

## 3 Technical Approach

The formal methods approach is loosely based on earlier work conducted by the inter-center team during 1993 on subsystems called Jet Select and Orbit

DAP [10]. Those techniques were adapted to accommodate the needs of this new area of the Shuttle software. All work has been mechanically assisted by the PVS toolset. PVS (Prototype Verification System) is an environment for specification and verification developed at SRI International's Computer Science Laboratory [13]. The distinguishing characteristic of PVS is a highly expressive specification language coupled with a very effective interactive theorem prover that uses decision procedures to automate most of the low-level proof steps.

### 3.1 State Machine Models

We have devised a strategy to model Shuttle principal functions based on the use of a conventional abstract state machine model. Each principal function is modeled as a state machine that takes inputs and local state values, and produces outputs and new state values. This method provides a simple computational model similar to popular state-based methods such the A-7 model [8, 15].

One transition of the state machine model corresponds to one scheduled execution of the principal function, e.g., one cycle at rate 6.25 Hz or other applicable rate. All of the inputs to the principal function are bundled together and a similar bundling of the outputs is arranged. The state variable holds values that are (usually) not delivered to other units, but instead are held for use on the next cycle.

The state machine transition function is a mathematically well-defined function that takes a vector of input values and a vector of previous-state values, and maps them into a vector of outputs and a vector of next-state values.

$$M : I \times S \rightarrow [O \times S]$$

This function $M$ is expressed in PVS and forms the central part of the formal specification. We construct a tuple composed of the output and state values so only a single top-level function is needed in the formalization. Some values may appear in both the output list and the next-state vector, i.e., they are not mutually exclusive.

While the function $M$ captures the functionality of the software subsystem in question, the state machine framework can also serve to formalize abstract properties about the behavior of the subsystem. The common approach of writing assertions about *traces* or sequences of input and output vectors is easily accommodated. For example, we can introduce sequences $I(n) = < i_1, \ldots, i_n >$ and $O(n) = < o_1, \ldots, o_n >$ to denote the flow of inputs and outputs that would have occurred if the state machine were run for $n$ transitions. A property about the behavior of $M$ can be expressed as a relation $P$ between $I(n)$ and $O(n)$ and formally established, i.e., we can prove that the property $P$ does indeed follow from the formal specification $M$ using the PVS proof-checker.

### 3.2 Expression in PVS

Figure 2 shows the abstract structure of a Shuttle principal function rendered in PVS notation. Key features of this structure are:

```
pf_result: TYPE = [# output: pf_outputs, state: pf_state #]

principal_function (pf_inputs, pf_state,
                    pf_I_loads, pf_K_loads,
                    pf_constants) : pf_result =

          (# output := <output expression>,
             state  := <next-state expression>
           #)
```

**Fig. 2.** PVS model of a Shuttle *principal function.*

- Principal functions use two kinds of variable data (input values, previous-state values) and three kinds of constant data (I-loads, K-loads, constants).
- Executing a principal function produces output values and next-state values.
- All externally visible effects on variables are to be captured by this model.

The PVS definition assumes all input and state values have been collected into the structures `pf_inputs` and `pf_state`. Additionally, all I-load, K-load, and constant inputs used by the principal function are collected into similar structures. The `pf_result` type is a record that contains an output component and a next-state component. Each of these objects is, in turn, a structure containing (possibly many) subcomponents.

The output and next-state expressions in the general form above describe the effects of invoking the subfunctions belonging to the principal function. In practice, this can be very complicated so a stylized method of organizing this information has been devised. It is based on the use of a LET expression to introduce variable names corresponding to the intermediate inputs and outputs exchanged among subfunctions.

## 3.3 Deviations from CR/FSSR Requirements

In deriving the preceding specification method, we have tried to be faithful to the FSSR method of expressing requirements. A few deviations and omissions, however, should be noted.

- The concept of state variables is not explicitly mentioned in FSSR-style requirements. Their use has been inferred and a method has been provided for their specification to make the final requirements more clear.
- No provision was introduced to capture initialization requirements for state variables. This issue can be handled at the next higher level of modeling.
- Conditional assignments in algorithmic requirements occasionally leave variable values unspecified. We assign default values to such cases when it is clear that the variable's value on one branch of a conditional is a "don't care."

## 4 Formalizing the Requirements

Initially, the relevant portions of the CR were analyzed to determine the basic structure of the principal functions and how they are decomposed into subfunctions. Based on this organization, a general approach for modeling the functions and expressing the formal specifications in PVS was devised. A document on this prescribed technique for writing formal specifications for the GPS CR was written and sent to the Loral requirements analysts.

Next, the interfaces of the principal functions and their subfunctions were carefully scrutinized. Particular emphasis was placed on being able to identify the types of all inputs and outputs, and to match up all the data flows that are implicit in the tabular format presented in the requirements. While conducting this analysis and preparing to write the formal specifications, various minor discrepancies were detected in the CR and these were reported to Loral requirements analysts.

A set of preliminary formal specifications was developed for the principal functions known as GPS Receiver State Processing and GPS Reference State Processing, using the language of PVS. Assumptions were made as needed to overcome the discrepancies encountered. Enough detail was provided in the formal specifications to characterize the functions with high precision. In parallel with this activity, several Loral RAs have been learning formal methods and PVS and positioning themselves to carry out this work after the trial project is completed.

Formalization of the two principal functions in PVS has been completed and revised three times to keep up with requirements changes. Because of the breadth of this CR, convergence has been slow. Requirements changes have been frequent and extensive as the CR was worked through the review process. Our initial formal specification was based on a preliminary version of the CR, before the two-phase implementation plan was adopted. Subsequent versions were written to model the single-string GPS CR and its revisions. PVS versions were written for Mod B, Mod D/E and Mod F/G of the CR. (Revisions or modifications are denoted Mod A, Mod B, etc.)

Excerpts from the formalization are shown in Figures 3 through 8. The full formal specifications contain over 3300 lines of PVS notation (including comments and blank lines), packaged as eleven PVS theories.

Figure 3 shows a portion of the vector and matrix utilities needed to formalize operations in this application domain. Using a parameterized theory such as this made it easy to declare vectors of reals where the index type differs from one vector type to the next. Figure 4 illustrates the declaration of some typical types found in this application and how the vector types are incorporated. All the types needed are rather simple and concrete; structured types are all of fixed size. As is customarily done in PVS, vectors and arrays are represented by function types.

Figure 5 presents one of the subfunctions from GPS Receiver State Processing. The outputs are bundled together into a single record type and used as the result type for the PVS function used to model the Shuttle software subfunction. The definition of the function contains a single expression, a record constructor

```
vectors [index_type: TYPE]: THEORY
BEGIN

vector:         TYPE = [index_type -> real]

i,j,k:          VAR index_type
a,b,c:          VAR real
U,V:            VAR vector

zero_vector:       vector = (LAMBDA i: 0)
vector_sum(U, V):  vector = (LAMBDA i: U(i) + V(i))
vector_diff(U, V): vector = (LAMBDA i: U(i) - V(i))
scalar_mult(a, V): vector = (LAMBDA i: a * V(i))

 . . .

END vectors
```

**Fig. 3.** Vector operations organized as a PVS theory.

```
major_mode_code:    TYPE = nat
mission_time:       TYPE = real
GPS_id:             TYPE = {n: nat | 1 <= n & n <= 3}

receiver_mode:      TYPE = {init, test, nav, blank}
AIF_flag:           TYPE = {auto, inhibit, force}

M50_axis:           TYPE = {Xm, Ym, Zm}

IMPORTING           vectors[M50_axis]

M50_vector:         TYPE = vector[M50_axis]

position_vector:    TYPE = M50_vector
velocity_vector:    TYPE = M50_vector
GPS_positions:      TYPE = [GPS_id -> position_vector]
GPS_velocities:     TYPE = [GPS_id -> velocity_vector]

GPS_predicate:      TYPE = [GPS_id -> bool]
GPS_times:          TYPE = [GPS_id -> mission_time]
GPS_FOM_vector:     TYPE = [GPS_id -> GPS_figure_of_merit]
```

**Fig. 4.** Selected type declarations.

```
ref_state_anncd_reset_out: TYPE = [#
      GPS_anncd_reset_avail:    GPS_predicate,
      GPS_anncd_reset:          GPS_predicate,
      R_ref_anncd_reset:        GPS_positions,
      T_anncd_reset:            GPS_times,
      T_ref_anncd_reset:        GPS_times,
      V_IMU_ref_anncd_reset:    GPS_velocities,
      V_ref_anncd_reset:        GPS_velocities
      #]

ref_state_announced_reset(DT_anncd_reset,
                          GPS_DG_SF,
                          GPS_SW_cap,
                          R_GPS,
                          T_anncd_reset,
                          T_current_filt,
                          T_GPS,
                          V_current_GPS,
                          V_GPS) : ref_state_anncd_reset_out =

   (# GPS_anncd_reset_avail := GPS_DG_SF,
      GPS_anncd_reset       :=
        (LAMBDA I: IF GPS_DG_SF(I)
                     THEN (T_current_filt - T_anncd_reset(I)
                            >= DT_anncd_reset)
                     ELSE false
                  ENDIF),
      R_ref_anncd_reset     :=
        (LAMBDA I: IF GPS_DG_SF(I) THEN R_GPS(I)
                                   ELSE null_position ENDIF),
      T_anncd_reset         :=
        (LAMBDA I: IF GPS_DG_SF(I) AND
                     (T_current_filt - T_anncd_reset(I)
                       >= DT_anncd_reset)
                     THEN T_current_filt
                     ELSE null_mission_time
                  ENDIF),
      T_ref_anncd_reset     :=
        (LAMBDA I: IF GPS_DG_SF(I) THEN T_GPS(I)
                                   ELSE null_mission_time ENDIF),
      V_IMU_ref_anncd_reset :=
        (LAMBDA I: IF GPS_DG_SF(I)
                     THEN V_current_GPS(I)
                     ELSE null_velocity
                  ENDIF),
      V_ref_anncd_reset     :=
        (LAMBDA I: IF GPS_DG_SF(I) THEN V_GPS(I)
                                   ELSE null_velocity ENDIF)
    #)
```

**Fig. 5.** Sample subfunction of Receiver State Processing.

that gives values for each of the required outputs. In this case they are all structured objects with GPS_id as the index type. Therefore, lambda-expressions with the variable $I$ ranging over GPS_id are used to construct suitable values.

To further illustrate the approach, consider the following example:

```
(LAMBDA I: IF GPS_DG_SF(I) THEN R_GPS(I) ELSE null_position ENDIF)
```

This expression evaluates to a function from $\{1, 2, 3\}$ to position vectors. For GPS receiver $I$, if its "data good" flag is set (GPS_DG_SF(I) holds), then use the position value R_GPS(I) derived from the input R_GPS, otherwise use a default position value.

In several cases, the subfunction requirements are fairly complex and it was necessary to introduce intermediate PVS functions to decompose the formalization. While this is a natural thing to do, it does cause some loss of traceability to the original requirements. Clarity and readability were judged more important, however, and such decompositions were introduced as needed.

Figure 6 shows the method of modeling principal function interfaces as records of individual values corresponding to Shuttle program variables. Because the interfaces at this level are quite broad, some of these lists become moderately long, on the order of 20 or 30 elements. In reality, these inputs and outputs are not actually "passed" in any programming language sense during execution; they are usually accessed as global variables and thus can be thought of as having the semantics of "call by reference." Consequently, our formalization must necessarily be viewed as a model of the software structure, and in some cases there are unpleasant artifacts of the difference between the model and the real system.

Figures 7 and 8 depict the top-level structure of the GPS Receiver State Processing model. Its interface types are given by the declarations shown in Figure 6. Its body is of the form

```
LET sf_1_out = subfun-1(...),
     . . .      . . .
    sf_n_out = subfun-n(...)
IN
   (# output := (# ... #),
      state  := (# ... #)
    #)
```

Each local variable assignment of the LET-expression represents the invocation of a subfunction and the storage of its intermediate results. Those values can be used directly as principal function outputs or passed to later subfunctions on the list. The final expression denotes the ultimate principal function result, which has the form of output values plus state values.

## 5 Results

The formalization step demonstrated that it is not difficult to bring the precision of formalization to bear on the type of requirements we examined. Expressing

```
rec_sp_inputs: TYPE = [#
      crew_deselect_rcvr:     GPS_predicate,
      earth_pole:             position_vector,
       . . .                   . . .
      V_GPS_ECEF:             GPS_velocities_WGS84,
      V_last_GPS:             GPS_velocities
      #]

rec_sp_state: TYPE = [#
      G_two_prev:             GPS_accelerations,
      GPS_DG_SF_prev:         GPS_predicate,
       . . .                   . . .
      V_last_GPS_sel:         velocity_vector,
      V_last_GPS_two:         velocity_vector
      #]

rec_sp_I_loads: TYPE = [#
      acc_prop_min_GPS:       real,
      acc_prop_thresh_GPS:    real,
       . . .                   . . .
      SF_vel:                 real,
      sig_diag_GPS_nom:       cov_diagonal_vector
      #]

rec_sp_K_loads: TYPE = [#
      acc_prop_min:           real,
      GPS_SW_cap:             num_GPS
      #]

rec_sp_constants: TYPE = [#
      deg_to_rad:             real,
      earth_rate:             real,
      G0:                     real,
      nautmi_per_ft:          real
      #]

rec_sp_outputs: TYPE = [#
      corr_coeff_GPS:         corr_coeff_vector,
      crew_des_rcvr_rcvd:     GPS_predicate,
       . . .                   . . .
      V_IMU_ref_anncd_reset:  GPS_velocities,
      V_ref_anncd_reset:      GPS_velocities
      #]

rec_sp_result: TYPE = [# output: rec_sp_outputs,
                         state:  rec_sp_state  #]
```

**Fig. 6.** Principal function interface types.

```
GPS_receiver_state_processing((rec_sp_inputs:    rec_sp_inputs),
                              (rec_sp_state:     rec_sp_state),
                              (rec_sp_I_loads:   rec_sp_I_loads),
                              (rec_sp_K_loads:   rec_sp_K_loads),
                              (rec_sp_constants: rec_sp_constants) )
                              : rec_sp_result =

  LET IMU_assign_out =
        IMU_assign(
            GPS_installed       (rec_sp_I_loads),
            GPS_SW_cap          (rec_sp_K_loads),
            nav_IMU_to_GPS      (rec_sp_inputs),
            V_current_filt      (rec_sp_inputs),
            V_last_GPS_two      (rec_sp_state) ),

      nav_state_prop_out =
        nav_state_propagation(
            acc_prop_min        (rec_sp_K_loads),
            acc_prop_min_GPS    (rec_sp_I_loads),
             . . .               . . .
            V_last_GPS_prev     (IMU_assign_out),
            V_last_GPS_sel      (rec_sp_state) ),

      SV_qual_assess_out =
        state_vector_quality_assessment(
            G_two               (nav_state_prop_out),
            GPS_DG_SF           (nav_state_prop_out),
             . . .               . . .
            V_GPS               (nav_state_prop_out),
            V_GPS_prev          (rec_sp_state) ),

      state_vect_sel_out =
        state_vector_selection(
            corr_coeff_GPS_nom  (rec_sp_I_loads),
            crew_deselect_rcvr  (rec_sp_inputs),
             . . .               . . .
            V_GPS               (nav_state_prop_out),
            V_GPS_sel           (nav_state_prop_out) ),

      ref_st_ann_reset_out =
        ref_state_announced_reset(
            DT_anncd_reset      (rec_sp_I_loads),
            GPS_DG_SF           (nav_state_prop_out),
             . . .               . . .
            V_current_GPS       (IMU_assign_out),
            V_GPS               (nav_state_prop_out) ),
```

**Fig. 7.** Principal function specification.

```
    GPS_downlist_out =
        GPS_downlist_computation(
            crew_deselect_rcvr   (rec_sp_inputs),
            DT_QA2               (SV_qual_assess_out),
             . . .                . . .
            SF_vel               (rec_sp_I_loads),
            V_GPS_sel            (state_vect_sel_out) )

IN (# output := (#
       corr_coeff_GPS :=     corr_coeff_GPS     (state_vect_sel_out),
       crew_des_rcvr_rcvd := crew_des_rcvr_rcvd (state_vect_sel_out),
        . . .                 . . .               . . .
       V_IMU_ref_anncd_reset :=
                 V_IMU_ref_anncd_reset          (ref_st_ann_reset_out),
       V_ref_anncd_reset :=  V_ref_anncd_reset  (ref_st_ann_reset_out)
       #),

      state := (#
       G_two_prev :=         G_two_prev         (SV_qual_assess_out),
       GPS_DG_SF_prev :=     GPS_DG_SF_prev     (SV_qual_assess_out),
        . . .                 . . .               . . .
       V_last_GPS_sel :=     V_last_GPS_sel     (nav_state_prop_out),
       V_last_GPS_two :=     V_last_GPS_two     (nav_state_prop_out)
       #)
    #)
```

**Fig. 8.** Principal function specification (cont'd).

the requirements in the language of an off-the-shelf verification methodology was straightforward. We found PVS effective for this purpose; we feel other languages would also fare well.

This much was unsurprising. What was more of a pleasant discovery was the number of problems found in the requirements as a simple consequence of carrying out the formalization. While many have claimed this as a benefit of formal methods, we can offer another piece of anecdotal evidence to support it. All of the errors identified so far have been due to carrying the analysis only to the point of typechecking. It was also our intention to take up some theorem proving as well, but this has had to wait for the requirements themselves to reach a firmer state of convergence.

Based on our initial results, some Shuttle RAs are optimistic about the potential impact of formal methods. Others in the Shuttle community are curious about the potential benefits of formalization. The RAs' feedback indicated our approach was helpful in detecting three classes of errors:

1. Type 4 — requirements do not meet CR author's intent.
2. Type 6 — requirements not technically clear, understandable and maintain-

able.
3. Type 9 — interfaces inconsistent.

An example of Type 4 errors encountered in the CR is omission due to conditionally updating variables. Suppose, for example, one branch of a conditional assigns several variables, leaving them unassigned on the other branch. The requirements author intends for the values to be "don't cares" in the other branch, but occasionally this is faulty because some variables such as flags need to be assigned in both cases. Similar problems encountered are those due to overlapping conditions, leading to ambiguity in the correct assignments to make.

Examples of Type 9 errors include numerous, minor cases of incomplete and inconsistent interfaces. Missing inputs and outputs from tables, mismatches across tables, inappropriate types, and incorrect names are all typical errors seen in the subfunction and principal function interfaces. Most are problems that could be avoided through greater use of automation in the requirements capture process.

All requirements issues detected during the formalization were passed on to Loral representatives. Those deemed to be real issues, that is, not caused by the misunderstandings of an outsider, were then officially submitted on behalf of the formal methods analysis as ones to be addressed during the requirements inspections. Severity levels are attached to valid issues during the inspections. This allowed us to get "credit" for identifying problems and led to some rudimentary measurements on the effectiveness of formalization.

| Issue Severity | Mod B | Mod D/E | Mod F/G | Totals |
|---|---|---|---|---|
| High Major | 1 | 0 | 0 | 1 |
| Low Major | 7 | 3 | 0 | 10 |
| High Minor | 19 | 40 | 6 | 65 |
| Low Minor | 8 | 0 | 2 | 10 |
| Totals | 35 | 43 | 8 | 86 |

**Fig. 9.** Summary of issues detected by formal methods.

Figure 9 summarizes a preliminary accounting of the issues identified during our analysis. The issues are broken out by severity level for the three inspections of the CR that took place during the formal methods study. A grand total of 86 issues were submitted for the three inspections. Of these issues, 72 of the 86 were of Type 9 (interfaces inconsistent). The rest were primarily scattered among Type 4 (requirements do not meet CR author's intent) and Type 6 (requirements not technically clear, understandable and maintainable). Note that many issues submitted at a given inspection remained unresolved in the next revision. These were not resubmitted, however, meaning all the issues cited in the table are distinct.

The meaning of the severity codes used in Figure 9 is as follows:

1. High major — Loral cannot implement requirement.
2. Low major — Requirement does not correctly reflect CR author's intent.
3. High minor — "Support" requirements are incorrect or confusing.
4. Low minor — Minor documentation changes.

As can be seen by these results, the added precision of formalization used early in the lifecycle can yield tangible benefits. While many of these issues could have been found with lighter-weight techniques, the use of formal specifications can detect them *and* leave open the option of deductive analysis later on. Thus, these results by themselves suggest a potential boost from the use of formal methods plus the promise of additional benefits if proving is ultimately attempted.

It is worth noting that most errors detected in the CR during the formalization exercise were not directly found by typechecking or other automated analysis activity, but were detected during the act of writing the specifications or during the review and preparation leading up to the writing step. Additional problems were found during the typechecking phase as well. When we reach the point of modeling higher level properties and carrying out proofs, we expect to see fewer errors still. This is consistent with general observations practitioners have about inspections and reviews. Light-weight forms of analysis applied early detect more problems and detect them quickly, but they are usually superficial. As more powerful analysis methods are introduced, we find more subtle problems, but they tend to be less numerous.

The next step in the application of formal methods to GPS, which was still in progress as of this writing, is to identify and formalize important behavioral properties of the processing of GPS position and velocity vectors. In particular, the feedback loop shown in Figure 1 involving the principal functions Receiver State Processing and Reference State Processing is fertile ground for investigation. Proving that suitable properties hold would offer a powerful means of further shaking out the requirements before passing them on to development.

Perhaps the most encouraging outcome of the study was a serious interest on the part of the requirements analysts to learn formal methods and continue the formalization activity themselves. Loral and JSC personnel received a training course at NASA Langley and intend to maintain and extend the GPS formal specifications during the implementation phase. Other CRs are being examined for potential evaluation as well. We are hopeful that this will lead to a continuing involvement by the NASA space community.

## 6 Conclusions

Experience with the GPS effort showed that the outlook for formal methods in this requirements analysis domain is quite promising. PVS has been used effectively to formalize this application, and the custom specification approach should be easy to duplicate for other areas. There are good prospects for continuation of the effort by Shuttle personnel. Some Shuttle RAs are optimistic about the potential impact of formal methods. Although the specification activity was assisted by tools, doing manual specification is also feasible here, albeit with reduced benefits.

PVS provides a formal specification language of considerable theoretical power while still preserving the syntactic flavor of modern programming languages. This makes the specifications fairly readable to nonexperts and makes their development less difficult than might otherwise be the case with specification languages whose features are more limiting. The scheme detailed here leads to specifications that RAs and others from the Shuttle community can and did learn to read and interpret without having to become PVS practitioners. Moreover, the mere construction of formal specifications using this method can and did lead to the discovery of flaws in the requirements. Future efforts can use the specifications as the foundation for more sophisticated analyses based on the use of formal proof. This additional tool provides the means to answer nontrivial questions about the specifications and achieve a higher level of assurance that the requirements are free of major flaws.

The methods outlined for formally specifying requirements were devised to meet the needs of the chosen CR. They are methods having fundamental utility that should lend themselves to other avionics applications. Tailoring a scheme for other uses or fine tuning it for the intended CR is easily accomplished. Alternative specification styles could readily be adopted. Experience in using the methods on live applications will help determine what direction future refinements should take.

In addition to specifications to capture the functionality of the principal functions, often it is desirable to formalize abstract properties about the long-term behavior of software subsystems. Formulating such properties is a way of assuring that certain critical constraints on system operation are always observed, allowing us to reason in a "longitudinal" manner by expressing what should be true about the software behavior over time rather than merely what holds at the current step. The specification framework sketched here can be extended easily to accommodate invariants or other property-oriented assertions.

The requirements analysis process used on the Shuttle program was originally put in place in the 1970s, and consists largely of manual, best-effort scrutiny whose effectiveness depends on the diligence of the analyst. Consider how formalizing requirements would help overcome several often-cited deficiencies of this process:

1. *There is no methodology to guide the analysis.*
   Formal methods offer rigorous modeling and analysis techniques that bring increased precision and error detection to the realm of requirements.
2. *There are no completion criteria.*
   Writing formal specifications and conducting proofs are deliberate acts to which one can attach meaningful completion criteria.
3. *There is no structured way for RAs to document the results of their analysis.*

Formal specifications are tangible products that can be maintained and consulted as analysis and development proceed. When provided as outputs of the analysis process, formalized requirements can be used as evidence of thoroughness and coverage, as definitive explanations of how CRs achieve their objectives, and as permanent artifacts useful for answering future questions that may arise.

## Acknowledgements

The author is grateful for the cooperation and support of the requirements analysts and other staff members at Loral Space Information Systems: Larry Roberts, Mike Beims, and, in earlier phases of this work, David Hamilton and Dan Bowman. Their enthusiasm for the project made the work much more meaningful. Thanks are also due to John Kelly (JPL), John Rushby and Judy Crow (SRI), and Rick Butler (LaRC). This work was supported in part by the National Aeronautics and Space Administration under Contract No. NAS1-19341.

## References

1. A. Arnold, M-C. Gaudel, and B. Marre. An Experiment on the Validation of a Specification by Heterogeneous Formal Means: The Transit Node. In *5th IFIP Working Conference on Dependable Computing for Critical Applications (DCCA-5)*, Champaign-Urbana, IL, 1995.
2. Ricky W. Butler, James L. Caldwell, Victor A. Carreño, 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)*, pages 135–149, Gaithersburg, MD, June 1995.
3. Dan Craigen, Susan Gerhart, and Ted Ralston. An international survey of industrial applications of formal methods; Volume 1: Purpose, approach, analysis and conclusions; Volume 2: Case studies. Technical Report NIST GCR 93/626, National Institute of Standards and Technology, Gaithersburg, MD, April 1993.
4. Judy Crow. Finite-State Analysis of Space Shuttle Contingency Guidance Requirements. Technical Report SRI-CSL-95-17, Computer Science Laboratory, SRI International, Menlo Park, CA, December 1995. Also forthcoming as a NASA Contractor Report for Task NAS1-20334.
5. Ben L. Di Vito and Larry Roberts. Using Formal Methods to Assist in the Requirements Analysis of the Space Shuttle GPS Change Request. Contractor report, NASA Langley Research Center, Hampton, VA, 1996. To appear.
6. David Hamilton, Rick Covington, and John Kelly. Experiences in Applying Formal Methods to the Analysis of Software and System Requirements. In *WIFT '95: Workshop on Industrial-Strength Formal Specification Techniques*, pages 30–43, Boca Raton, FL, 1995. IEEE Computer Society.
7. David Hamilton, Rick Covington, and Alice Lee. Experience Report on Requirements Reliability Engineering Using Formal Methods. In *ISSRE '95: International Conference on Software Reliability Engineering*, Toulouse, France, 1995. IEEE Computer Society.

8. K. L. Heninger. Specifying Software Requirements for Complex Systems: New Techniques and Their Application. *IEEE Transactions on Software Engineering*, SE-6(1):2–13, January 1980.
9. Robyn R. Lutz and Yoko Ampo. Experience Report: Using Formal Methods for Requirements Analysis of Critical Spacecraft Software. In *19th Annual Software Engineering Workshop*, pages 231–248. NASA GSFC, 1994. Greenbelt, MD.
10. Multi-Center NASA Team from Jet Propulsion Laboratory, Johnson Space Center, and Langley Research Center. *Formal Methods Demonstration Project for Space Applications – Phase I Case Study: Space Shuttle Orbit DAP Jet Select*, December 1993. NASA Code Q Final Report (Unnumbered).
11. National Aeronautics and Space Administration, Office of Safety and Mission Assurance, Washington, DC. *Formal Methods Specification and Verification Guidebook for Software and Computer Systems, Volume I: Planning and Technology Insertion*, July 1995.
12. National Research Council Committee for Review of Oversight Mechanisms for Space Shuttle Flight Software Processes, National Academy Press, Washington, DC. *An Assessment of Space Shuttle Flight Software Development Practices*, 1993.
13. Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.
14. John Rushby. Formal Methods and the Certification of Critical Systems. Technical Report SRI-CSL-93-7, Computer Science Laboratory, SRI International, Menlo Park, CA, December 1993. Also issued under the title *Formal Methods and Digital Systems Validation for Airborne Systems* as NASA Contractor Report 4551, December 1993.
15. A. John van Schouwen. The A-7 Requirements Model: Re-Examination for Real-Time Systems and an Application to Monitoring Systems. Technical Report 90-276, Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, May 1990.

This article was processed using the LaTeX macro package with LLNCS style