NAVASTIR III SOFTWARE SPECIFICATION TOOL

MK Benjamin and FB Six

Plessey Electronic Systems Corporation
Wayne, NJ

ABSTRACT

NAVASTIR III is an integrated software specification and program development tool that combines the development of a type–set quality requirements specification, the corresponding simulator code, the Ada coded operational software, and an Ada package specification of the Interface Control Document (ICD) into one process. In addition, NAVASTIR III performs checks on the input to insure consistent use of engineering units and dimensions in expressions and equations, and consistency of coordinate systems in vector and matrix expressions. This error checking eliminates entire classes of errors, many of which are difficult to find otherwise. NAVASTIR II, an earlier version, was used on an internal project to produce a high quality requirements specification. Many errors were corrected that would normally not be found until validation, and the simulator was generated at very little cost above that of producing the specification. The users and their management have been quite pleased with the system. NAVASTIR III enhances NAVASTIR II by automating the system interface description.

Introduction

NAVASTIR III is an integrated software specification and software development tool set that supports the analysis and statement of the problem, simulation and testing. It will provide

- A type–set quality Software Requirements Specification
- A Mechanization Simulator
- A Data Dictionary and I/O Tables
- A type–set quality Interface Control Document (ICD)
- An Ada coded representation of the Software Requirements Specification
- An Ada Package Specification of the ICD

To achieve the goal of generating the specification and program as part of one process, the problem is factored into four parts: a) analytics, b) description of the system interface, c) process timing requirements, and d) software implementation details.

The integration of what are normally separate steps in the development of software improves productivity and produces a much higher quality product. The analytics are checked for consistency. The description of the system interface and process timing are easily updated independently of the analytics. NAVASTIR III manages all of this information and ensures its consistency because a single copy is maintained. NAVASTIR II has proven reduction in cost and schedule of software products. We expect further gains from NAVASTIR III.

1. Analytics

The analytics can be expressed in terms of standard mathematical notation such as summation (Σ) and integral ($\int$) signs. The user defines variables and their associated attributes (relating to the underlying problem) such as engineering units, engineering dimensions, coordinate system for vectors, transformation coordinate systems, and numeric precision. As an executable specification, NAVASTIR III supports the propagation of not only the numeric values, but all the engineering attributes, including precision, through the calculation. Attribute propagation and consistency checking is an extremely powerful tool in determining the correctness and consistency of any engineering calculation.

NAVASTIR III provides a language for expressing calculations and formatting type–set output. The language includes arithmetic operations on scalars, vectors and matrices, as well as comparison, logical and control flow operations. A description of the language has been given elsewhere.[1]

[1]M Benjamin, NAVASTIR II Software Specification Tool, NSIA Quality Control and Reliability Assurance Fifth Annual Joint Conference on Software Quality and Productivity, Alexandria Va, 28 Feb 1989.

CH2759-9/89/0000-1970 $1.00 © 1989 IEEE

A user input Data Dictionary defines the variables required in calculations. Dictionary entries provide the type and dimension of the variable, engineering units, and a description of the variable. NAVASTIR III uses the Dictionary of Variables, together with a tree of subprogram calls it develops, to determine the module I/O tables and automatic buffering of variables between calculations running at different cycle rates.

Let us do a simple problem as an example. Suppose an Inertial Measurement Unit (IMU) is providing Δ velocities at 16 Hz. These data are integrated at the 16 Hz rate and buffered to a 4 Hz routine which updates the computer frame direction cosine matrix (C_P). C_P is then transmitted to the Guidance Computer where it is used for navigation. This problem was entered into NAVASTIR II which provided the following output:

The 16 Hz routine calculates $\vec{\rho}$ as

$$\vec{R} = \vec{R} + \vec{V} \cdot \Delta T_{16} + \Delta\vec{V} \cdot \frac{\Delta T_{16}^{\,2}}{2}$$

$$\vec{V} = \vec{V} + \Delta\vec{V} \cdot \Delta T_{16}$$

$$\vec{\rho} = \frac{\vec{V} \times \vec{R}}{|\vec{R}|^2}$$

where $\vec{R}$ is the position vector, $\vec{V}$ is the velocity vector and $\vec{\rho}$ is the angular transport vector across the earth's surface.

The 4 Hz routine performs the Cayley–Hamilton integration

$$C_P = C_P + C_P \cdot \left[\frac{\sin(|\vec{\rho}| \cdot \Delta T_4)}{|\vec{\rho}|} \cdot \Lambda + \frac{1 - \cos(|\vec{\rho}| \cdot \Delta T_4)}{|\vec{\rho}|^2} \cdot \Lambda^2 \right]$$

where the transport matrix $\Lambda_{3x3} = (\epsilon_{ijk}\, \rho_k)$ accounts for the earth's curvature.

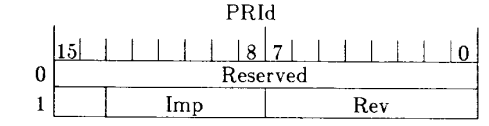
2. System Interface

NAVASTIR III allows the user to describe a system interface composed of messages, memory–mapped I/O, and memory shared among processors. Messages or memory layouts can include bit field, floating point, integer, and scaled fixed point variables, with alignment clauses providing a precise layout in memory. Messages can be automatically transmitted or received at a specified frequency or on an as required basis.

For example, a memory layout may be defined as

```
layout PRId {
    Reserved  s=15, l=16  ;
    Imp       s=13, l=6   ;
    Rev       s=7,  l=8   ;
} at hex 20
```

to define a Processor Revision Identifier location which appears as



Here, Imp and Rev define bit field variables of lengths 6 and 8. Reserved is a special **variable name** indicating that the space is reserved for future use. The unused space in the second word of the register is depicted as blank. The qualifiers '**s=**' and '**l=**' define the starting bit position and length of the field. The address clause **at hex 20** places the definition of this layout at 20_{16} in memory. Other optional qualifiers force alignment on word boundaries, such as **w= mod 2**, or locate the data field in a particular word numbered n, as **w=n**.

NAVASTIR III provides automation of the message handling, routing and buffering. Each distinct message is required to have a unique name given by the user. All the field names in messages are also required to be unique and distinct from variable names defined in the Dictionary of Variables. The user simply uses a field name in a calculation. NAVASTIR III associates the unique field name with its message, and extracts the value of that field from the most recent copy of the message. NAVASTIR III expects the user to supply device handler(s) to implement the protocol of the data link hardware and to decode each message received into the message name. For each input message the handler receives on the data link, it must make an entry call to rendezvous with the NAVASTIR III generated message handling task. The latter task supplies the data for the user software. To support output of messages, the device handler also must contain a procedure to send a single named message. This procedure may be used directly by the user as well as by NAVASTIR III generated code for automatic transmission of messages.

Continuing our IMU example, the following message definitions would be used

```
message IMU0 {
   ctrlword ;                  -- Control Word
   float DELTA V array(3), U=ft/sec  ;
                               -- Delta Velocities
}
```

```
message INU02 {
   RTAddr=4 w=0, s=4, l=5    ;
                          -- Remote Terminal Address
   SubAddr=1 w=0, s=10, l=5  ;
                          -- Sub Address
   fixed C sub P (1,1)  ,    -- Direction Cosines
         C sub P (2,1)  ,
         C sub P (3,1)  ,
         C sub P (1,2)  ,
         C sub P (2,2)  ,
         C sub P (3,2)   ;
}
```

The IMU01 message received from the IMU provides Δ velocities. The INU02 message contains the first two columns of the C_P matrix which are sufficient to determine C_P since the third column is just the vector cross product of the first two columns. The default fixed point used for the elements of C_P is a 16 bit two's complement fraction. The INU02 message is to be sent to the Guidance Computer, which is Subaddress 1 on MUX Remote Terminal Address 4. Using the message and memory layout features provided by NAVASTIR III, a user can implement any desired form of hand–shakes using messaging or other inter–processor communications, such as semaphores. For example, one processor might queue work for another in an area of shared memory with semaphores used to avoid conflicting accesses.

3. Timing Requirements

NAVASTIR III provides capabilities to support process timing and synchronization requirements. Messages can be automatically transmitted or received at specified rates. The execution rate of modules may be set to an appropriate cycle rate. NAVASTIR III determines the **primary source** of a variable to be the module running at the highest cycle rate in which the variable is set. It automatically buffers copies of the variable to modules running at rates different than that of the **primary source**. Similarly, buffered copies of messages are automatically maintained for modules which execute at a rate slower than the message rate or, alternatively, for modules which use several recent copies of the message.

Continuing our IMU example, we use

expect(IMU01) at 16 HZ else imu_err

in the system interface specification to receive a message named IMU01 at a 16 Hz rate. If the message is not received within the 16 Hz cycle, a time–out error has occurred, and the module imu_err is called to perform the error handling.

To provide output of the direction cosine matrix (C_P) to the guidance computer at a 4 Hz rate, we use

send(INU02) at 4 HZ else inu_err

in the system interface specification to send a message named INU02. If the message cannot be sent within the 4 Hz cycle, a time_out error has occurred and the module inu_err is called to handle the error processing.

The system specification can also designate routing of a particular set of messages to a specific module, say modx, as

on message (msg1, ... msg10) modx

If a module needs to send a message, it can do so by executing

send(msg_name,err_proc)

within a processing module. The return to the calling subprogram will be immediate after the message is queued and the message will be sent asynchronously. If an error occurs, the subprogram err_proc is called.

NAVASTIR III also provides a **delay** built–in–function call which allows a module to suspend processing for a designated delay time. All messages input at any given cycle rate are received before the processing is performed for that cycle rate. All messages transmitted at a specific cycle rate are output after the processing for that cycle rate.

NAVASTIR III provides a translation of the input specification into an executable mechanization simulator. This simulator implementation automatically includes routines that buffer data between processes which run at different cycle rates, and will be consistent in time–ordering of its execution. One must independently verify that the timing deadline requirements are met in the real–time implementation.

4. Software Implementation Details

Directives are provided for the user to influence the translation of the specification into Ada operational software code. These include directives for numeric data typing, Ada packaging, and machine representations of variables.

Several qualifiers may be added to Dictionary of Variables entries to influence how the translation to Ada code proceeds. The name of the variable in the generated Ada program can be specified as **N = name** which is useful for complex names, such as ΔT_{16}. If the user wants to name the package in which a set of variable names are declared, he may enter the **P = pkg_name** qualifier in each of the dictionary entries. The machine representation can be chosen by the **TY=type** qualifier. Finally, designation of a range by **R = lower , upper** will cause an Ada sub-type declaration to be produced for the variable.

In our example the Dictionary of Variables contains $\vec{R}$, $\vec{V}$, ΔT_4, ΔT_{16}, Λ, C_P and $\vec{\rho}$. The entry for $\vec{\rho}$ could be

```
rho vec  N=RHO TY= FLOAT(3) U= rad/sec
         T= Transport Rate P= pkg
         R= -.25 rad/hr , .25 rad/hr
```

where **U=rad/sec** specifies the units as radians and seconds, and the dimensions as 1/time. **T=Transport Rate** provides a descriptive tag for the variable in the automatically generated I/O tables. The buffered 4 Hz copy is automatically given the name 'RHO_4' and declared in the Ada package named **hz4**. This could be overridden by supplying a '**4=anyname**' qualifier in the Dictionary entry for $\vec{\rho}$.

The input for the IMU example and the Ada code output are shown in Appendices I and II, respectively.

Appendix I

The following is the input to NAVASTIR III for the IMU example:

```
message IMU01 {
   ctrlword ;                  -- Control Word
   float DELTA V array(3) U=ft/sec  ;
                               -- Delta Velocities
}

message INU02 {
   RTAddr=4 w=0, s=4, l=5     ;
                               -- Remote Terminal Address
   SubAddr=1 w=0, s=10, l=5  ;
                               -- Sub Address
   fixed C sub P (1,1)  ,     -- Direction Cosines
        C sub P (2,1)  ,
        C sub P (3,1)  ,
        C sub P (1,2)  ,
        C sub P (2,2)  ,
        C sub P (3,2)   ;
}
```

```
 expect(IMU01) at 16 HZ else imu_err
 send(INU02) at 4 HZ else inu_err

*SET CYCLE RATES 4 16
INTEG          HZ=16
UPDCOS         HZ=4
C sub P        N=CP      TY=FLOAT(3,3)  U=-
                         T=Direction Cosine Matrix
DELTA T sub 4  N=DT4     TY=FLOAT       U=sec
                         VAL=0.25
                         T=4 Hz Cycle Time
DELTA T sub 16 N=DT16    TY=FLOAT       U=sec
                         VAL=0.0625
                         T=16 Hz Cycle Time
LAMBDA                   TY=FLOAT(3,3)  U=rad/sec
                         R=-.25 rad/hr,+.25 rad/hr
                         T=Position Vector
R vec                    TY=FLOAT(3)    U=ft
                         R=-4200 mi,+4000 mi
                         T=Position Vector
rho vec        N=RHO     TY=FLOAT(3)    U=rad/sec
                         R=-.25 rad/hr,.25 rad/hr
                         T=Transport Rate P=pkg

V vec                    TY=FLOAT(3)    U=ft/sec
                         R=-1000 mi/hr,+1000 mi/hr
                         T=Velocity Vector

*SET BEGIN INTEG
*SET COPYTEXT ON
The 16 Hz routine calculates %rho vec% as
 .EQ R vec = R vec + V vec  dot  DELTA T sub 16 +
          DELTA V vec dot { DELTA T sub 16 sup 2 }
          over 2 .EN
 .EQ V vec = V vec + DELTA V vec dot
           DELTA T sub 16 .EN
 .EQ  rho vec = { V vec cross R vec }
             over | R vec | sup 2 .EN
```

where %R vec % is the position vector, %V vec% is the velocity vector and % rho vec % is the angular transport vector across the earth's surface.

```
*SET BEGIN UPDCOS
*SET COPYTEXT ON
The 4 Hz routine performs the Cayley-Hamilton
 integration
*SET COPYCALC OFF
 .EQ LAMBDA = left [ matrix     {
       ccol { 0 above -rho vec (3) above rho vec (2) }
       ccol { rho vec (3) above 0 above -rho vec (1) }
       ccol { -rho vec (2) above rho vec (1) above 0 }
       } right ] .EN
*SET COPYCALC ON
 .EQ  C sub P = C sub P + C sub P dot            left [
          sin ( | rho vec | dot DELTA T sub 4 )
          over | rho vec | dot LAMBDA  +
          { 1 - cos ( | rho vec |  dot  DELTA T sub 4 ) }
          over | rho vec | sup 2 dot  LAMBDA sup 2
      right ]  .EN
```

where the transport matrix % LAMBDA sub 3x3 = (epsilon sub ijk rho sub k) % accounts for the earth's curvature.

Appendix II

The following is the output Ada code to be generated by NAVASTIR III for the IMU example.

The package Nav_math supplies the definition of data types and over-loaded operators for NAVASTIR III generated code. It also defines the word size (WORD) and record boundary (REC_BDRY) for alignment of memory layouts and messages. The package is written to provide layout for 16-bit words, such as used in a MIL-STD-1750A computer, and to align records on double word boundaries for efficient access.

The packages **global**, *pkg and hz4 declare the variables the user has input into the Dictionary of Variables. All the variables not defined to be local and lacking a* **p=** *directive are assigned to the package* **global**, *by default. The package pkg was created to satisfy the user's directive. The package hz4 was automatically generated to declare variable(s) used in the 4 Hz cycle which are buffered copies of variables from other cycles. Note that the variable* $\vec{\rho}$ *is calculated in the 16 Hz cycle and used in the 4 Hz cycle; consequently the buffered copy RHO_4 is used in procedure UPDCOS.*

The packages INU02_p, IMU01_p and procedure fill_IMU02 are generated from the **message** *declarations in the system interface specification. The first two packages declare records to represent the messages and use Ada representation clauses to yield the desired layout. The fill_INU02 procedure fills in the copy of the INU02 message. Note that it converts the* C_P *matrix entries from the floating point representation specified in the Dictionary of Variables to the fixed point format required for the output message. The procedure fill_INU02 is called prior to sending the INU02 message.*

The procedures INTEG and UPDCOS are generated directly from the user input calculations. INTEG uses the Δ *velocities from the IMU01 message to calculate* $\vec{\rho}$.

```
pragma STORAGE_UNIT(8) ;
package Nav_math is
  BYTE      : CONSTANT := 8 ;
  WORD      : CONSTANT := 2 * BYTE ;
  DEL       : CONSTANT := 1.0/2.0**(WORD-1) ;
  REC_BDRY: CONSTANT := 2*WORD ;

  type FRAC  is delta DEL range -1.0..1.0-DEL ;
  type VEC   is array( INTEGER range <> ) of FLOAT ;
  type ARR   is array( INTEGER range <> ,
                       INTEGER range <> ) of FLOAT ;

  function "+"(X,Y:ARR) return ARR ;
  function "+"(X,Y:VEC) return VEC ;
  function "-" (X,Y:ARR) return ARR ;
  function "-" (X,Y:VEC) return VEC ;

  function "*" (X,Y:ARR) return ARR ;
  function "*"  (X,Y:VEC) return FLOAT ;
                              -- vector dot product
  function "*" (X:ARR; Y:VEC) return VEC ;
  function "*" (X:FLOAT; Y:VEC) return VEC ;
  function "*" (X:FLOAT; Y:ARR) return ARR ;
  function "/" (X:ARR; Y:FLOAT) return ARR ;
  function "/" (X:VEC; Y:FLOAT) return VEC ;
  function "***"(X:ARR;N:INTEGER) return ARR ;
  function "abs"(X:VEC) return FLOAT ;
  function cross (X:VEC; Y:VEC) return VEC ;
end Nav_math ;

  with Nav_math ;
  use  Nav_math ;
package global is
  LAMBDA    : ARR(1..3,1..3) ;
  CP        : ARR(1..3,1..3) ;
  DT4       : FLOAT := 0.25 ;
  DT16      : FLOAT :=0.0625 ;
  R_vec     : VEC(1..3) ;
  V_vec     : VEC(1..3) ;
end global ;

  with Nav_math ;
  use  Nav_math ;
package pkg is
  RHO       : VEC(1..3) ;
end pkg ;


  with Nav_math ;
  use  Nav_math ;
package hz4 is
  RHO_4     : VEC(1..3) ;
end hz4 ;



  with Nav_math ;
  use  Nav_math ;
package  IMU01_p is
  type IMU01_r is record
    ctrlword    : INTEGER ;
    DELTA_V  : VEC(1..3) ;
  end record ;
private
  for IMU01_r use
    record at mod REC_BDRY ;
      ctrlword at 0*WORD range 0..WORD-1 ;
      DELTA_V  at 1*WORD range 0..3*FLOAT'SIZE-1 ;
    end record ;
end IMU01_p ;
```

```
  with Nav_math ;
  use Nav_math ;
package INU02_p is
  type INU02_r is record
    RTAddr     : INTEGER range 0..31  ;
    SubAddr    : INTEGER range 0..31  ;
    CP_1_1 , CP_2_1 , CP_3_1 ,
    CP_1_2 , CP_2_2 , CP_3_2    : FRAC  ;
  end record ;
private
  for INU02_r use
    record at mod REC_BNDRY  ;
      RTAddr  at 0*WORD range 0..4   ;
      SubAddr at 0*WORD range 6..10  ;
      CP_1_1  at 1*WORD range 0..WORD-1 ;
      CP_2_1  at 2*WORD range 0..WORD-1 ;
      CP_3_1  at 3*WORD range 0..WORD-1 ;
      CP_1_2  at 4*WORD range 0..WORD-1 ;
      CP_2_2  at 5*WORD range 0..WORD-1 ;
      CP_3_2  at 6*WORD range 0..WORD-1 ;
    end record ;
end INU02_p  ;
```

```
  with global, Nav_math, INU02_p ;
  use  global, Nav_math, INU02_p ;
procedure fill_INU02(msg : in out INU02_r) is
begin
  msg.CP_1_1 := FRAC(CP(1,1)) ;
  msg.CP_2_1 := FRAC(CP(2,1)) ;
  msg.CP_3_1 := FRAC(CP(3,1)) ;
  msg.CP_1_2 := FRAC(CP(1,2)) ;
  msg.CP_2_2 := FRAC(CP(2,2)) ;
  msg.CP_3_2 := FRAC(CP(3,2)) ;
end fill_INU02 ;
```

```
  with IMU01_p, Nav_math, INU02_p ;
  use  IMU01_p, Nav_math, INU02_p ;
package messages is
  IMU01       : IMU01_r ;
  INU02       : INU02_r ;
end messages ;
```

```
  with global, Nav_math, messages, IMU01_p, pkg ;
  use  global, Nav_math, messages, IMU01_p, pkg ;
procedure INTEG is
begin
  -- The 16 Hz routine calculates % rho vec % as
  R_vec := R_vec + DT16 * V_vec +
           DT16 **2/2.0 * IMU01.DELTA_V ;
  V_vec := V_vec + DT16 * IMU01.DELTA_V ;
  RHO := cross( V_vec , R_vec ) /
         ( abs R_vec ) **2  ;
  -- where % R vec % is the position vector, % V vec %
  --  is the velocity vector and % rho vec % is the
  --  angular transport vector across the earth's
  --  surface.
end INTEG ;
```

```
  with global, Nav_math, hz4 ;
  use  global, Nav_math, hz4 ;
procedure UPDCOS is
begin
  -- The 4 Hz routine performs the Cayley-Hamilton
  --  integration
  LAMBDA := (1=>(0.0 , RHO_4(3) , -RHO_4(2) ),
             (2=>(-RHO_4(3) , 0.0 , RHO_4(1) ),
             (3=>(RHO_4(2) , -RHO_4(1) , 0.0 )) ;
  CP := CP + CP * (
      sin((abs RHO_4)*DT4)/(abs RHO_4) * LAMBDA +
      (1.0 - cos((abs RHO_4)*DT4))/(abs RHO_4) **2 *
      LAMBDA **2              ) ;
  -- where the transport matrix % LAMBDA sub 3x3 =
  --  ( epsilon sub ijk rho sub k ) % accounts
  --  for the earth's curvature.
end UPDCOS  ;
```

The above shows only code generated directly by NAVASTIR III from the user input calculations, system interface specification and the Dictionary of Variables, and the library units this code references. Additional support code generated by NAVASTIR III is discussed below.

NAVASTIR III generates the **executive** *which calls procedures INTEG and UPDCOS at the appropriate cycle rates to implement a mechanization simulator. The mechanization simulator will be consistent in its time–ordering of execution of modules and message passing events although the simulator does not run in real time. Other code is generated to support buffering and message handling. The executive calls buffering routines to buffer variables and/or messages between procedures executing at different cycle rates.*

A dummy device handler is supplied within the implementation of the mechanization simulator. The device handler is expected to service both reception and transmission of messages. The dummy handler contains an asynchronous task which reads input messages from a file and makes an entry call on a task within the **executive** *to pass input messages to it. The dummy handler also contains a procedure called to output messages. This procedure simply writes the messages to a file.*

As described above, the package Nav_math provides layout for 16 bit words. To execute this code on DEC's VAX Ada V1.4–33 it is necessary to set WORD to 32 since this compiler does not permit 16–bit fixed point objects as desired for the FRAC data type. The resulting code will be portable between VAX computers, and probably also to other 32 bit computers. However, it does not provide the desired packing of the message. Thus, use of the code to communicate between a VAX and a MIL–STD–1750A computer would require message packing and unpacking within the VAX device handler.

Another aspect of Ada causes potential problems in sharing memory layouts between computers. Ada permits implementations to use either the **lsb** *(least significant bit) or* **msb** *(most significant bit) as bit number zero. NAVASTIR III expects user input with the* **lsb** *as bit zero. It should provide an option to generate code for implementations where the* **msb** *is numbered zero. For example, a field specified with* **s=5, l=5** *will be placed into bits numbered 9..13, if the option for* **msb** *is invoked, so that shared memory layouts will work correctly.*

Although we chose floating point for our IMU01 example because we are communicating with known hardware, floating point is generally not portable across computers and interfaces, and should be avoided in layouts and messages.