AAS 99-063

USE OF THE MATRIXX INTEGRATED TOOLKIT ON THE MICROWAVE ANISOTROPY PROBE ATTITUDE CONTROL SYSTEM

David K. Ward, Stephen F. Andrews, David C. McComas, James R. O’Donnell, Jr., Ph.D
NASA’s Goddard Space Flight Center.

Recent advances in analytical software tools allow the analysis, simulation, flight code, and documentation of an algorithm to be generated from a single source, all within one integrated analytical design package. NASA's Microwave Anisotropy Probe project has used one such package, Integrated Systems' MATRIXx suite, in the design of the spacecraft's Attitude Control System. The project's experience with the linear analysis, simulation, code generation, and documentation tools will be presented and compared with more traditional development tools. In particular, the quality of the flight software generated will be examined in detail. Finally, lessons learned on each of the tools will be shared.

INTRODUCTION

The economics of the space industry continually force engineers to evaluate and improve their development practices. With that in mind, NASA’s Goddard Space Flight Center (GSFC) evaluated its work on the Tropical Rainfall Measuring Mission (TRMM) and the Rossi X-Ray Timing Explorer (RXTE) in preparation for a new program called MIDEX. MIDEX, short for Medium Explorers, is a program to launch a space science mission every year with a cost cap of $70 million. The evaluation of the TRMM and RXTE missions revealed that, while the programs were extremely successful, there were areas that would need to be improved in preparation for the MIDEX mission, the Microwave Anisotropy Probe (MAP)[1].

One of the areas that needed improvement was the design, development, and testing of the Attitude Control System (ACS) algorithm and its associated flight software. An overview of the previous design process, depicted in Figure 1, shows that instead of branching the analysis, simulation, code, and test efforts from a single design point, each was developed separately from scratch. In fact, a separate engineer was employed solely for the purpose of manually documenting the algorithm changes and making sure they were incorporated in the flight software. Additionally, the Hybrid Dynamic Simulator (HDS), designed to test the flight software, was designed by a separate team and was unable to benefit from any efficiencies of co-development.

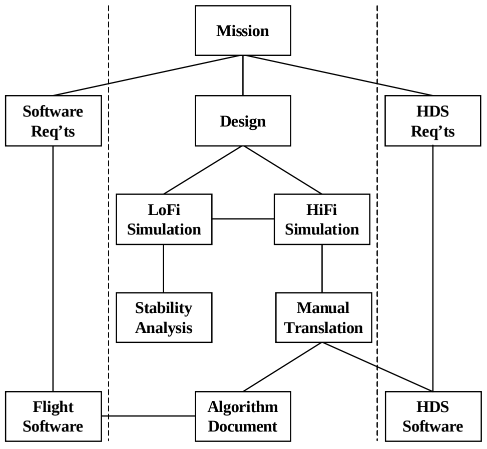


Figure 1: Old Development Method

A quick review of the high level MAP requirements revealed that the ACS subsystem would require another complex design[2]. Thus, members of GSFC's Guidance, Navigation and Control Center (GNCC) determined that a significant process improvement would be possible with the use of an integrated software design toolkit. The MATRIXx software suite from Integrated Systems, Inc. was selected after an internal trade study that included some of the engineers who would be responsible for using the new tools. The software would be used to create a single design point in a high fidelity simulation (HiFi) that every team member could use (rather than needing one HiFi and several low fidelity simulations (LoFi's) to be used by other team members). That same HiFi could be used for the linear stability analysis (rather than another LoFi that would be used with a separate piece of software) and, using other modules in the MATRIXx toolkit, could be the basis of automatic code and documentation generation. This would eliminate a significant level of effort in performing the manual documentation and translation function and would have the significant additional benefit of ensuring that all members of the ACS design team were working from the same single design point. Also, it would reduce the risk of errors associated with the manual translation effort. A representation of this ideal development process is shown in Figure 2.

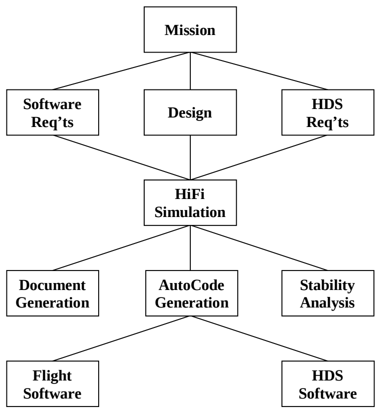


Figure 2: Proposed New Development Method

This paper will detail the MAP ACS team's efforts to use the MATRIXx toolkit and the lessons learned during that effort. The paper is roughly divided according to the different functionality in the toolkit itself; there are sections on the object-oriented simulation development, the automatic flight software code generation, the automated documentation generation, and the linear analysis efforts. These are followed by a brief discussion of other business factors that will play a part in the decision to use this particular brand of software. The paper will conclude with general lessons learned and an overall impression of the success of the effort.

USE OF SYSTEMBUILD IN SIMULATION DEVELOPMENT

SystemBuild is the component of the MATRIXx integrated tool suite that is used primarily for developing and running nonlinear simulations. As will be discussed in later sections, some of MATRIXx's linear analysis, automatic code generation, and documentation generation capabilities are also based on the simulation developed within SystemBuild. This section will focus on SystemBuild's simulation capabilities and contrast the use of SystemBuild on MAP to develop a HiFi simulation with what has been done in the past.

SystemBuild is based on a graphical block diagram editor used to construct the desired simulation. Its two basic element types are "blocks", which represent the functional elements of a simulation, and "SuperBlocks", which can be used to group other blocks and/or other SuperBlocks into hierarchies. SystemBuild comes with a wide assortment of blocks that can implement linear and nonlinear systems, continuous and discrete systems, as well as a variety of user-definable blocks that can be used to include arbitrary functionality into a simulation.

Description of System to be Modeled

Figure 3 is the top-level SystemBuild SuperBlock of the MAP simulation, which gives an idea of the pieces of the MAP system that are being modeled, as well as how the simulation is organized. As shown, the top level has been organized into three SuperBlocks, named "ACS", "ACE", and "Models". These SuperBlocks are used for the following simulation elements:

- **ACS**—The ACS SuperBlock contains elements of the simulation that correspond to components of the MAP flight software hosted on their main spacecraft processor. Most of the contents of the ACS SuperBlock correspond to elements of MAP's flight software, including those elements from which flight software will be automatically generated (see the Code Generation section, later in this paper). Other than Flight Software (FSW), the ACS SuperBlock also models some of the ground-based commanding.

- **ACE**—The ACE SuperBlock models the needed elements of the MAP Attitude Control Electronics (ACE). On the MAP spacecraft, the ACE is used to implement the independent Safehold Mode, and to provide the interface to most of MAP's sensors and actuators. Because the MAP simulation uses engineering units, and does not go to the level of counts, currents, or voltages, it was not necessary to model those interfaces in the ACE SuperBlock. Instead, the two functions that are modeled are the independent Safehold and the interface that takes the thruster commands from the ACS SuperBlock and turns them into an appropriately sized pulse width.

- **Models**—In the Models SuperBlock, the actual physics of the MAP spacecraft and environment are modeled. This includes the models of the spacecraft attitude, position, and velocity, environmental disturbance models, and models of MAP's sensors and actuators.

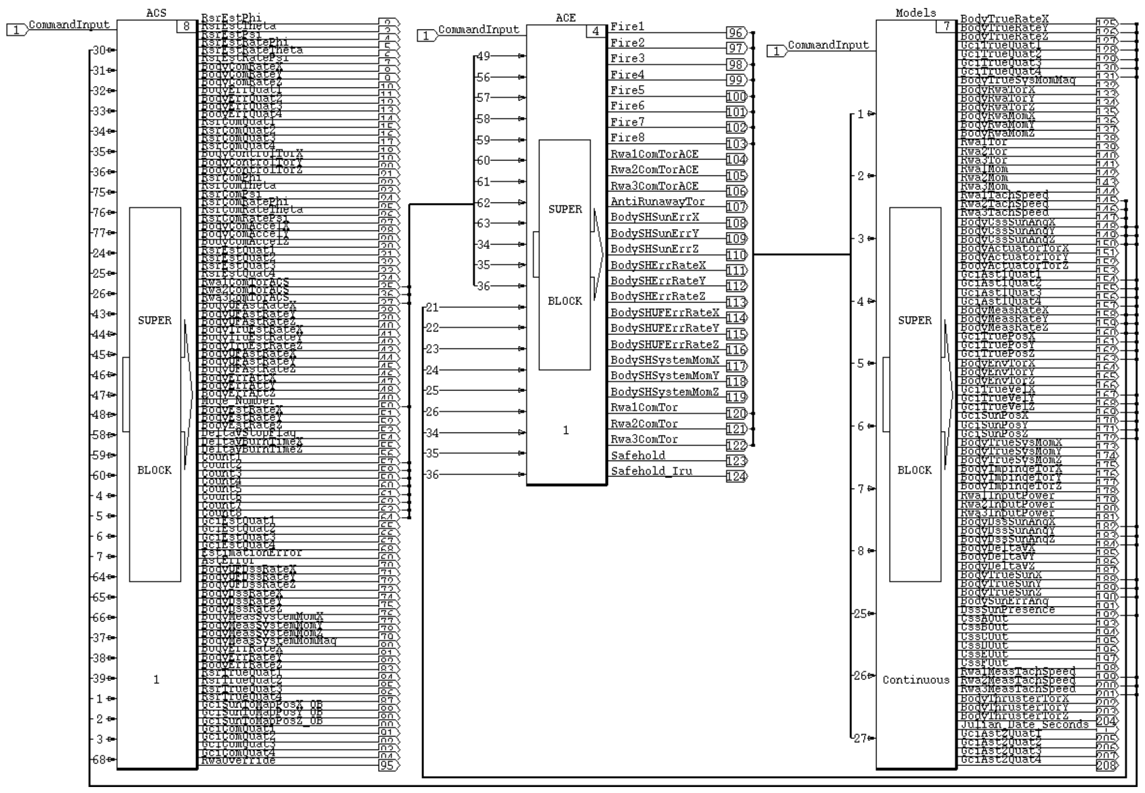


Figure 3: MAP Simulation Top-Level SuperBlock

Description of Previous Process

The MAP simulation and design effort differed from that used for previous missions in a number of important areas. In previous missions, the HiFi was generally developed in a conventional programming language such as FORTRAN or C. Typically, one member of the ACS design and analysis team would be dedicated to developing, testing, and running the HiFi in support of the other members. This person would accept inputs from the other members of the team designing various pieces of the attitude control system and would implement the algorithms that they developed into the HiFi.

Once the HiFi, and the ACS algorithms implemented therein, had reached a certain level of maturity, another member of the ACS team would prepare a document describing the control algorithms and other necessary components of the system that needed to be implemented onboard. This algorithm document and the HiFi software would then be used as reference material by other teams to develop the flight software and the HDS.

If this were simply a linear process, flowing from the HiFi to the algorithm document to the flight software, it would be very easy to manage. In reality, however, the design of the ACS subsystem is iterative. Testing of the flight software might reveal algorithmic or implementation problems. Fixing these problems can cause changes in the HiFi, which must be reflected in the algorithm document, and then used to change the flight software. This process can be very difficult to manage.

Simulation Design Standards

One of the goals of using the MATRIXx integrated tool suite was to alleviate some of the problems in translating an ACS design into fully tested ACS flight software. The primary way in which MATRIXx allowed the MAP team to meet this goal was through the use of the automatic code and documentation generation tools, AutoCode and DocumentIt. Through the use of these tools, the HiFi became the central point of configuration control for many pieces of the MAP ACS development. Unlike in previous projects, though, changes in HiFi could be incorporated *automatically* into the flight software or the algorithm documentation, without a separate manual translation step.

To do this properly, it was important to establish some design standards when the HiFi was being developed. The most important of these was to clearly define which parts of the simulation would be used for automatic code generation. In addition, to be able to interface the automatically generated code with the rest of the flight software, it was necessary to clearly define the input and output interfaces. Figure 4 shows the ACS SuperBlock, divided primarily into the portions to be used to automatically generate code (the Autocodable SuperBlock) and those not being used for that purpose (the Non-AutoCode SuperBlock).

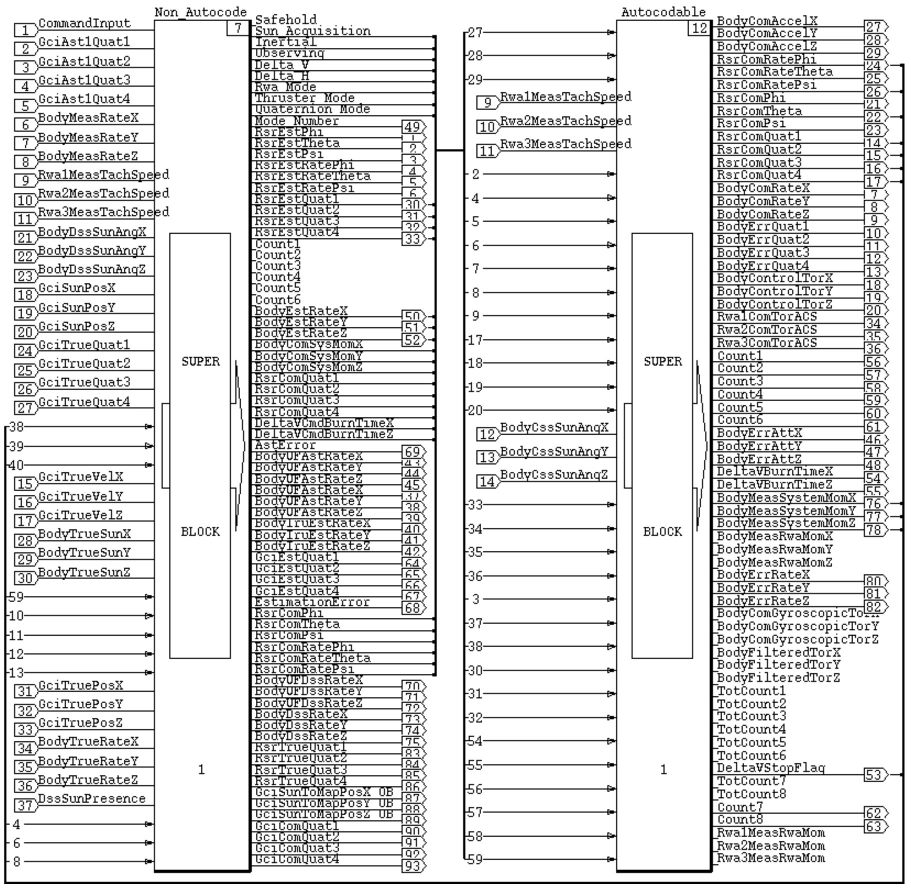


Figure 4: MAP Simulation ACS SuperBlock

Given this structure, it was necessary then to define design standards covering three aspects of the development of the HiFi simulation:

- **Scope of Autocodable SuperBlock**—In order to minimize risk on MAP, which is one of the first missions in which GSFC has used automatic code generation, the simulation elements chosen to be used with AutoCode were carefully defined. This scoping analysis is described in greater detail below.

- **Input/Output Interface**—The inputs and outputs of the Autocodable SuperBlock provided the main method for interfacing the AutoCode and non-AutoCode portions of the MAP ACS flight software. Very early in the design of the HiFi, this interface was clearly defined between the HiFi and flight software developers. This design was quite successful, requiring only minor changes as the design matured[3].

- **ACS Parameters**—SystemBuild uses what it refers to as %VARs to parameterize variables in a simulation. These are parameters that can be included within SystemBuild blocks that are tied to XMath variables, allowing them to be set and changed when a simulation is begun. In the MAP ACS flight software, parameter tables are used to fulfill the same function. The initial stages of the design set the standard of using the %VARs in the Autocodable SuperBlock as a single flight software table, which allowed them to be consistently set and changed in either the HiFi or the flight software.

Tying together the HiFi, the flight software, and the HDS was the flight software parameter database. This database served a role similar to the HiFi in the MATRIXx integrated development environment. Just as the HiFi served as a central point for *algorithmic* development and testing, with changes to the portions for which code was being automatically generated being automatically disseminated, the flight software database was used as the clearinghouse for the parameters used when implementing these algorithms. For example, control gain changes, once tested in the HiFi, could be entered into the flight software database and would then automatically be used by the flight software.

Comparison with Previous Practices

Although the differences in the TRMM, RXTE and MAP algorithms makes it difficult to make an apples-to-apples comparison between the previous method of developing simulations and SystemBuild, there are several observations that can be made. The first is the relative ease in the initial development of HiFi on MAP as compared to the previous programs. It took about a month for one engineer to build a working high fidelity simulation from scratch. That time is roughly the same as it took for another, similarly experienced engineer to develop the TRMM HiFi, except in that case the development was created from an existing HiFi simulation. If the additional infrastructure work (such as developing an integration routine) is factored into this development, SystemBuild was much easier to use.

The real benefit of using MATRIXx comes from sharing work between engineers. Two relevant experiences are the MAP team's ability to use the MAP HiFi and another program's use of the MAP HiFi in the development of their HiFi. In the first case, the ease of use meant the entire ACS team, including software developers and systems engineers, were able to develop and run their own HiFi cases. One situation where this turned out to be extremely useful was in flight software testing. A tester had the ability to run a test case on HiFi to match what was run on the flight system and to use MATRIXx's plotting capabilities and pre-developed scripts to compare the two cases. This is a substantial improvement over the previous system that did not easily allow other personnel to develop and run their own test cases and use those for comparison with flight data.

In the second case, the Triana mission took less than two weeks to build a working HiFi from the basis of the MAP HiFi simulation. The two missions are similar, but not identical. This is a substantial improvement over previous missions, which would not have the ability to develop more that a low fidelity simulation in such a short period of time. The relative ease of this development is extremely important with respect to the strategic mission of GSFC, as it will allow engineers to supply science teams with high fidelity feasibility studies quickly in support of early mission analysis.

AUTOMATIC CODE GENERATION

AutoCode is the component of the MATRIXx integrated tool suite that is used to automatically generate C code from SystemBuild models. A script written in ISI's Template Programming Language (TPL) controls the code generation process. The code produced for an individual block cannot be altered, but the user can customize the order in which the SystemBuild blocks are processed, the organization of the C source files, and the code that interfaces to the automatically generated code. The topmost SuperBlock is supplied to AutoCode and code is generated for this SuperBlock and every subordinate block. Code can be generated for multi-rate systems using the "scheduled–subsystem" option, and for single rate systems using the "procedures-only" option. This section describes how AutoCode changed the software development process, the rationale for how AutoCode was used, and the implementation.

ACS FSW Development Process

Figure 5 illustrates the MAP ACS FSW development process. The solid lines represent the traditional process and the dashed lines indicate where the process has changed for MAP. The MATRIXx tool set has impacted about 50% of the process.

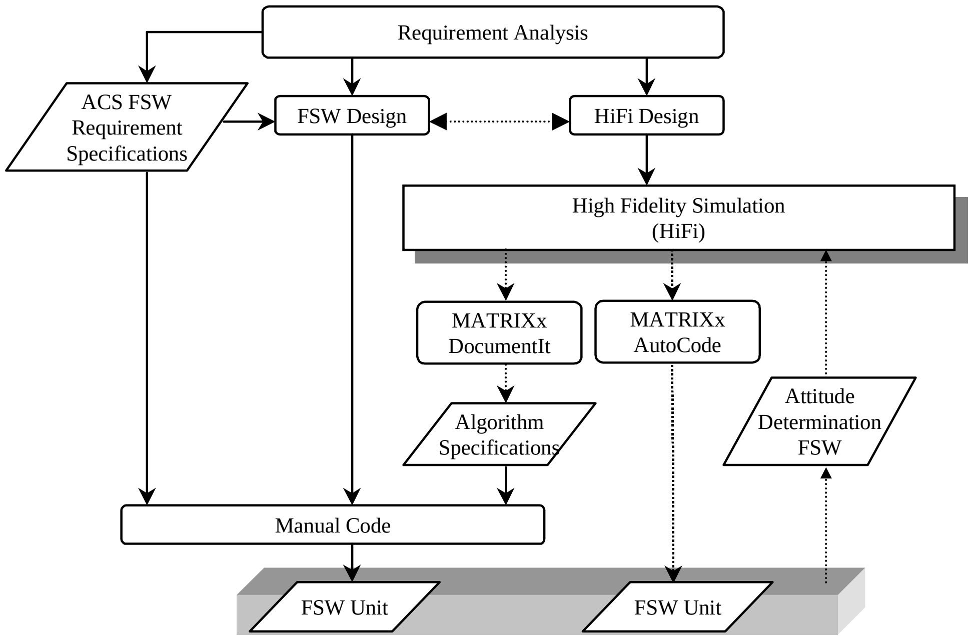


Figure 5: ACS FSW Unit Development Process

The process began with an ACS flight software (FSW) requirement analysis. This is a system engineering process primarily involving Guidance Navigation and Control (GNCC) analysts, FSW specialists, and spacecraft system engineers. This activity produced an ACS FSW Requirements Specification and fed directly into both the FSW and the high-fidelity (HiFi) simulator designs. HiFi was used by the GNCC analysts to validate the algorithms needed in the ACS FSW. On previous GSFC missions, the ACS FSW architecture and the HiFi software architectures were developed independently, with minimal-to-none commonality between the two software systems. The commonality between the two designs existed primarily at the function level.

The use of AutoCode required that the FSW and HiFi architectures account for each other's environment. The graphical HiFi environment encapsulates function and data into a component called a SuperBlock. In order for a translated SuperBlock to plug into the FSW environment, HiFi must emulate the FSW environment or the SuperBlock must avoid interfacing to or relying on FSW components. Any differences that exist between the two environments must be accounted for by the automatic code generation process or by manually changing code after it has been generated. The analysis and design section describes the translation strategies that were taken on MAP.

The implementation phase received its inputs from the ACS requirement specifications and the ACS algorithm specifications. The ACS requirement specifications defined the ACS FSW functional and performance requirements. The algorithm specification was a companion to the requirements, and it defined the mathematical details that needed to be implemented by the FSW in order to meet the functional and performance criteria. Traditionally all of the FSW has been manually coded from these two inputs. Three significant changes were made on MAP. First, the generation of the ACS algorithms specification was automated using ISI's DocumentIt. In the past, the generation and maintenance of the ACS algorithm specifications was a laborious job that has required a dedicated analyst. The second change eliminated part of the manual coding effort by automatically generating code using AutoCode. The third change reversed the traditional process. A verified FSW Kalman Filter, used on RXTE and TRMM for attitude determination, was linked into the HiFi simulation.

Figure 5 does not show the iterative aspect to the development process, which is important with respect to configuration management. The FSW was delivered to the build test team in incremental builds, and the requirements and design were continually refined for successive builds. In the past, the development team identified which algorithm specification version was being implemented by a particular FSW build, but the MAP team was not so disciplined in configuring a HiFi version that supplied the inputs to the algorithms. Since AutoCode produces a product that is directly incorporated by the FSW, it forced the team to improve the HiFi configuration management. Every FSW build is directly associated with a HiFi release. The improved configuration management has also improved the build test verification effort since testers are always comparing apples-to-apples.

Analysis and Design

The primary goal of the analysis and design phase with respect to AutoCode was to define the scope of the automatically generated code. Both business and technical forces shaped AutoCode's boundary. The primary business issues were that the schedule be met while delivering a high quality, testable product that implements the requirements. MAP is Goddard's first mission to use an automatic code generator for its FSW, and prior to MAP, Goddard had no experience with ISI's code generator. MAP's strategy towards mitigating risks was to reduce the scope of the automatically generated code. Essentially two criteria were used to identify candidate components. First, the component must be loosely coupled with the FSW environment in order to minimize the impact of the FSW environment upon the HiFi. Second, the component should have a high algorithm-to-code ratio. These criteria would select components that minimize the impact to the analysts while taking advantage of the biggest benefit of AutoCode, which is to eliminate the error prone process of manually translating algorithms to flight code.

Technically, automatic code generation is the translation of a design from the HiFi environment to the FSW environment. The HiFi environment must model any aspects of the FSW environment if the generated code is to be linked with the flight code without any manual changes to the automatic code. MAP uses a custom built library called the software bus as the inter-task communication mechanism. The software bus isolates the application tasks from the underlying operating system. Opting not to model the software bus in HiFi, the team immediately limited the scope of each AutoCode invocation to an intra-task scope and ISI's real-time operating system, pSOSystem, was not even considered. Additional flight unique interfaces include ground command processing, asynchronous event message generation, and fault detection notification. Again, the team opted not to model these interfaces in the HiFi, further restricting the application of AutoCode.

With these guidelines in hand, the portion of the ACS FSW to be automatically generated could be identified. Figure 6 shows a simplified high-level block diagram of MAP's flight control software. This software was suitable for a single task because all of its components execute at 1Hz and have fairly strong data cohesion. Sensors measure spacecraft position and rates. Attitude determination uses sensor measurements to update the onboard estimated attitude, which is supplied to the controller subsystem. The desired spacecraft attitude is either supplied by mode management or internally computed by command generation. Attitude error computes control errors for the control law based on a combination of sensor measurements, estimated attitude, and commanded attitude. The control law computes control torques which are output to the actuators.

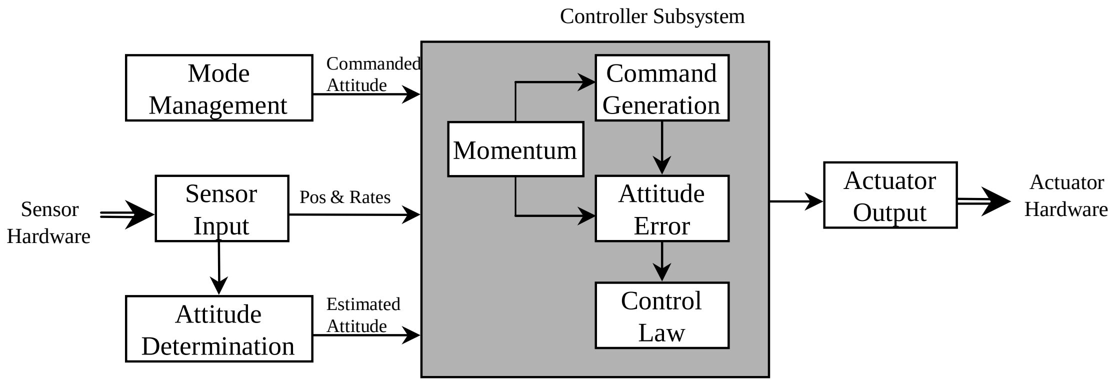


Figure 6: ACS FSW block diagram

The shaded controller subsystem in Figure 6 corresponds to the Autocodable SuperBlock in Figure 4 and identifies what is automatically generated. This final design takes advantage of the controller subsystem's relatively small and simple set of inputs and outputs. The controller subsystem's local rotating-sun-reference coordinate frame is encapsulated entirely within the AutoCoded portion of the FSW. Since the controller subsystem components execute at the same rate, the "procedures-only" AutoCode option could be used, which greatly simplifies the automatic code. Attitude determination shares many of the same attributes as the controller subsystem with respect to being suitable for AutoCode, but it was not chosen to be automatically generated since MAP could adapt an existing attitude determination subsystem from RXTE and TRMM..

Implementation

Generating the automatic code was a straightforward process. First, the HiFi was loaded into SystemBuild and default %VAR values were defined by executing MAP-specific MathScripts (XMath's scripting language). When AutoCode was run, the user identified the topmost SuperBlock in the hierarchy to be AutoCoded and supplied the name of TPL file that will control the code generation process. From a SuperBlock perspective, code generation is closed, which means the user cannot change the code generated for a particular SuperBlock. However, other aspects of the code generation process can be manipulated from the TPL file.

The MAP TPL script outputs each SuperBlock into a separate source file with a corresponding header file and it generates two interface functions that are called by the manually coded FSW. The separate source files facilitate configuration management as the software matures. Only the files that have changed need to be delivered. The two interface functions, achifi_Initialize() and achifi_Execute(), provide a clear and simple interface. Outputs from achifi are exported via a global data structure named achifi_Output. The manual code treats this as a read-only data structure; although no mechanisms enforce this rule. Figure 7 contains excerpts from the FSW illustrating how the automatic code is managed.

```
Controller file scope excerpt
#include "achifi.h"                /* Autocode header file  */
achifi_Input_Rec  achifi_Input;    /* Autocode input record */


Controller_Init() excerpt
achifi_Init(&achifi_Input,TRUE);   /* TRUE means copy %VARs from EEPROM */


Controller_Execute() excerpt
achifi_Input.BodyEstRateX = DataMgr.Config.BodyRate.Comp[X];
achifi_Input.BodyEstRateY = DataMgr.Config.BodyRate.Comp[Y];
achifi_Input.BodyEstRateZ = DataMgr.Config.BodyRate.Comp[Z];

achifi_Input.Rwa1MeasTachSpeed = RWA_ProcData.Speeds[0];
achifi_Input.Rwa2MeasTachSpeed = RWA_ProcData.Speeds[1];
achifi_Input.Rwa3MeasTachSpeed = RWA_ProcData.Speeds[2];
. . .
achifi_Execute(&achifi_Input,DataMgr.Config.DeltaTime);

AttCtl.BodySysMom.Comp[X] = (float)achifi_Output.BodyMeasSystemMomX;
AttCtl.BodySysMom.Comp[Y] = (float)achifi_Output.BodyMeasSystemMomY;
AttCtl.BodySysMom.Comp[Z] = (float)achifi_Output.BodyMeasSystemMomZ;
AttCtl.BodySysMomMag = (float)Vector3f_Mag(&AttCtl.BodySysMom);
. . .
```

Figure 7: Automatic code interface

Overall, the automatic code may be considered less readable than the manual code shown above, but this is primarily due to where structures are defined and how they are initialized. Some may consider the inclusion of SystemBuild's numeric block identifiers in the variable names as cumbersome, but they aid in tracing the code to the design. The automatic code generation is very systematic and once one gets a feel for how status information, state information, inputs, outputs, and initialization are managed, the code is relatively easy to read. The code excerpt shown in Figure 8 is from the Inertial Mode controller. By convention, every function uses "U" as a pointer to the input structure and "Y" as a pointer to the output structure. Comments are inserted in the code to identify which block is being coded. The code's readability is further enhanced by having each SuperBlock output to a separate file, using data naming conventions in the HiFi, and labeling all block I/O lines in the HiFi. Unlabelled block I/O lines decrease the code's readability since variable names will be automatically generated.

```
/* ---------------------------  Gain Block */
      /* {Ctrl_Inertial.Position_Gain.1} */
      Position_Gain_1 = Inertial_Kp[0]*BodyLimErrAttX;
      Position_Gain_2 = Inertial_Kp[1]*BodyLimErrAttY;
      Position_Gain_3 = Inertial_Kp[2]*BodyLimErrAttZ;
      /* ---------------------------  Gain Block */
      /* {Ctrl_Inertial.Rate_Gain.12} */
      Rate_Gain_1 = Inertial_Kr[0]*U->BodyErrRateX;
      Rate_Gain_2 = Inertial_Kr[1]*U->BodyErrRateY;
      Rate_Gain_3 = Inertial_Kr[2]*U->BodyErrRateZ;
      /* ---------------------------  Gain Block */
      /* {Ctrl_Inertial.Integral_Gain.98} */
      Integral_Gain_1 = Inertial_Ki[0]*LimitedIntegrator_19_1;
      Integral_Gain_2 = Inertial_Ki[1]*LimitedIntegrator_19_2;
      Integral_Gain_3 = Inertial_Ki[2]*LimitedIntegrator_19_3;
      /* ---------------------------  Sum of Vectors */
      /* {Ctrl_Inertial.Summer_3.3} */
      Y->BodyControlTorX = Position_Gain_1 + Rate_Gain_1 + Integral_Gain_1;
      Y->BodyControlTorY = Position_Gain_2 + Rate_Gain_2 + Integral_Gain_2;
      Y->BodyControlTorZ = Position_Gain_3 + Rate_Gain_3 + Integral_Gain_3;
      /* ---------------------------  Procedure Super Block */
      /* {Filter_Inertial.2} */
      Filter_Inertial_2_u.BodyControlTorX = Y->BodyControlTorX;
      Filter_Inertial_2_u.BodyControlTorY = Y->BodyControlTorY;
      Filter_Inertial_2_u.BodyControlTorZ = Y->BodyControlTorZ;
      Filter_Inertial(&Filter_Inertial_2_u, &Filter_Inertial_2_y, &S->
         Filter_Inertial_2_s, &I->Filter_Inertial_2_i);
      Y->BodyFilteredTorX = Filter_Inertial_2_y.Filter_4o_4_1;
      Y->BodyFilteredTorY = Filter_Inertial_2_y.Filter_4o_4_2;
      Y->BodyFilteredTorZ = Filter_Inertial_2_y.Filter_4o_4_3;
      iinfo[0] = I->Filter_Inertial_2_i.iinfo[0];
      if( iinfo[0] != 0 ) {
         I->Filter_Inertial_2_i.iinfo[0] = 0; goto EXEC_ERROR;
      }
      /* ---------------------------  Sum of Vectors */
      /* {Ctrl_Inertial.Summer_15.15} */
      BodyComTorX = Y->BodyFilteredTorX - U->BodyFeedforwardTorX;
      BodyComTorY = Y->BodyFilteredTorY - U->BodyFeedforwardTorY;
      BodyComTorZ = Y->BodyFilteredTorZ - U->BodyFeedforwardTorZ;
      /* ---------------------------  Algebraic Expression */
```

Figure 8: Sample of Automatically Generated Flight Code

Results

There are a few drawbacks to the automatic code, but none have proven to be fatal. The automatic code is large and less efficient than its manual equivalent. The MAP team has not had the luxury of duplicating the coding effort manually, but there have been a couple of cases when a fair comparison between the manual and automatic code could be made[4]. In these cases the automatic code has been two to three times larger than the manual code. This code bloat is primarily due to the fact that AutoCode does a lot of data copying before and after calling a procedure. Declaring procedure blocks inline can reduce this overhead, but this doesn't allow SuperBlocks to reside in separate source files. Since MAP has a 1Hz ACS task execution rate and 4 megabytes of EEPROM, resources have not been a concern. Therefore, the code was designed for readability and maintainability rather than for performance. Version 6.0 of AutoCode includes optimization capabilities that may address concerns stated above for missions where those issues may be of more serious concern.

There are a few non-resource issues that did create some trouble. The team wanted to treat %VARs as one or more FSW tables. A FSW table must contain physically contiguous data and this is typically achieved by defining a table as a C structure. Unfortunately the XMath variable partitions do not translate into C structures. The team was able to contiguously group the %VARs by defining the %VARs in a separate file. Using linker scripts, the %VAR object file was defined as a contiguous data structure.

There have been two cases when the generated code wasn't as expected. In both cases, default values were hard coded for %VARs instead of variables being used. Hard coded %VARs is unacceptable so workarounds had to be developed. In one case the team coded the block in BlockScript, a FORTRAN-like procedural language. In the other case the design was changed so that the offending blocks were not used. Aside from the %VAR problem, the generated code has properly implemented the HiFi design.

The last issue with the generated code concerns the time between valid control cycles, which is referred to as delta time. Nominally, delta time is one second, but there are situations in which this time can be greater than one second. The ACS FSW must use the actual delta time to safely control the spacecraft. However, AutoCode hard codes a one second delta time because the top-most SuperBlock is defined as a 1Hz procedure block. To correct the situation the automatically generated code must be manually changed to use the delta time passed to achifi_Execute(). Since the automatic code generation process used on MAP produces a separate source file for each SuperBlock, a manual code change has to be made only once, unless the SuperBlock changes in a future build.

AUTOMATED DOCUMENTATION GENERATION

ISI has an automatic documentation generator as an add-on module to the MATRIXx suite. DocumentIt runs a template file that extracts various pieces of information from each block or SuperBlock, and writes that information in ASCII format to the designated file. Included with the module is a standard, modifiable template file that controls both the information that is gathered from each block and the output file format. The template programming language (TPL) file can be modified to extract information the designers deem relevant, and the output can be formatted to produce html files, or other standard file formats. As with any new tool, reading the documentation is a requirement, and starting small is a must.

The team decided to do the documentation on the World Wide Web, which is a complete departure from previous practice. One analyst was now given the task of learning the template programming language, updating the HiFi with comments and data as needed, creating useful images of the SuperBlocks, and writing any other HTML that was required to fully document the algorithm design. Each Superblock would be given its own HTML file, complete with inline image, individual block documentation, and relevant links to navigate the entire documentation. Adding HTML tags to the DocumentIt output by adding them to the template file complicates the TPL file, but it makes the output file more legible, and creates web pages that can be easily traversed. The complexity comes about because both the template file and the resulting HTML code had to be debugged.

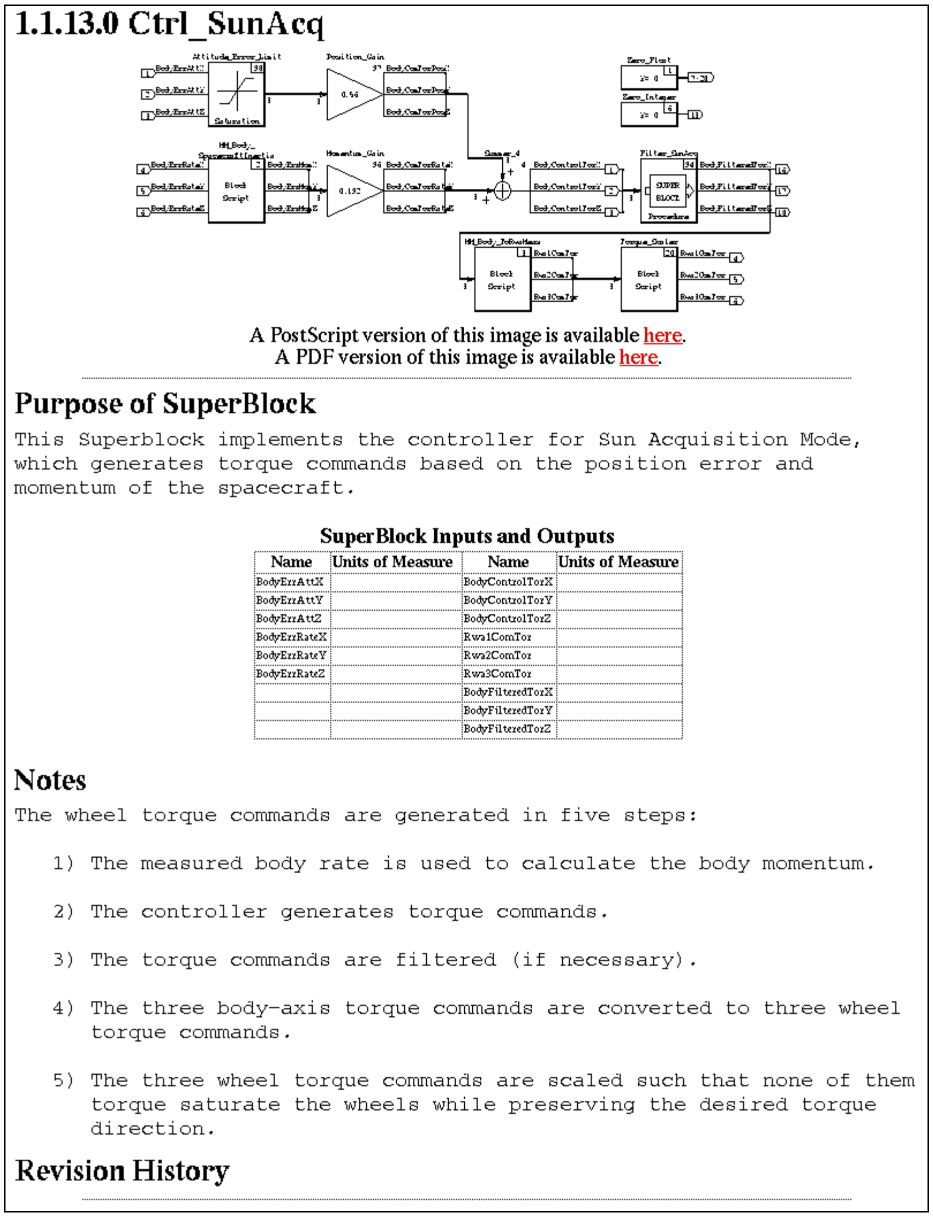
Part of the TPL file that creates a separate page for each SuperBlock is shown in Figure 9. The TPL is generally set off by @ or @@, and the html tags can be seen interspersed where needed.

```
@/ Open new file for each SuperBlock.  Format is name_section_numbers.html
@FILEOPEN(sb_name_s cat "_" cat ii cat "_" cat jj cat "_" cat kk cat "_" cat ll cat "_" cat mm cat "_" cat nn cat ".html","new")@@

<HTML>
<HEAD>
<TITLE>@sb_name_s@</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
@/ At each level, write section number and SuperBlock name as top header.
@IFF level_i eq 0@@
<H1>0.0  @sb_name_s@</H1>
@ENDIFF@@
@IFF level_i eq 1@@
<H1>@ii@.0  @sb_name_s@</H1>
@ENDIFF@@
@IFF level_i eq 2@@
<H1>@ii@.@jj@.0  @sb_name_s@</H1>
@ENDIFF@@
@IFF level_i eq 3@@
<H1>@ii@.@jj@.@kk@.0  @sb_name_s@</H1>
@ENDIFF@@
@IFF level_i eq 4@@
<H1>@ii@.@jj@.@kk@.@ll@.0  @sb_name_s@</H1>
@ENDIFF@@
@IFF level_i eq 5@@
<H1>@ii@.@jj@.@kk@.@ll@.@mm@.0  @sb_name_s@</H1>
@ENDIFF@@
@IFF level_i eq 6@@
<H1>@ii@.@jj@.@kk@.@ll@.@mm@.@nn@  @sb_name_s@</H1>
@ENDIFF@@
<CENTER>
<HR SIZE=5 WIDTH=85%>

@id = sb_id_i@@
@li = level_i@@
@/ Reference the gif version of the SuperBlock PostScript graphic to include in the
@/ HTML file.
<IMG SRC = "@sb_name_s@.gif" ALT = "@sb_name_s@ block diagram">
<center>A PostScript version of this image is available <a href = "@sb_name_s@.ps">here</a>.</center>
<center>A PDF version of this image is available <a href = "@sb_name_s@.pdf">here</a>.</center>
<HR SIZE=2 WIDTH=85%>

@/ List user-defined parameters (special comments) from the SuperBlock comment
@/ field.
</CENTER>
<H2>Purpose of SuperBlock</H2>
<pre>
@user_param("PURPOSE_s","SUPERBLOCK")@@
</pre>
@table_sb()@@
<H2>Notes</H2>
<pre>
@user_param("NOTES_s","SUPERBLOCK")@@
</pre>
<H2>Revision History</H2>
<pre>
@user_param("REVISIONS_s","SUPERBLOCK")@@
</pre>
<HR SIZE=5 WIDTH=85%>
```

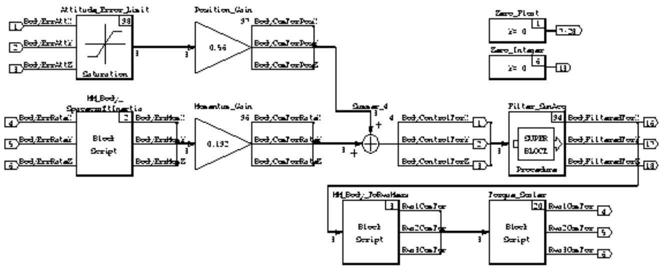
Figure 9: DocumentIt Template Programming Language Example

Each SuperBlock was given its own HTML file, which included an image. Every SuperBlock image was saved as PostScript, and a Unix utility written by a third party was used to convert them all to 72 dpi GIF images that were used as the image source in the HTML file. In addition, the original PostScript file and a manually converted PDF file were hyperlinked to from within the HTML code. Each type of block was given a separate segment in the TPL file; that segment would create documentation specifically for each block type. The SuperBlocks within each SuperBlock were also hyperlinked to, and every page had a link to the Table of Contents page, the cover page, the top level SuperBlock page, and its parent Superblock page. The Table of Contents page was created with a template file, but the cover page, mathematical appendix, and sensor data processing appendix were created independently on another platform, with HTML editors. This increased the amount of time that had to be devoted to algorithm documentation beyond learning and using DocumentIt.

Figure 10 shows a sample of portion of the resulting documentation[5].



1.1.13.0 Ctrl_SunAcq



A PostScript version of this image is available here.
A PDF version of this image is available here.

Purpose of SuperBlock

```
This Superblock implements the controller for Sun Acquisition Mode,
which generates torque commands based on the position error and
momentum of the spacecraft.
```

SuperBlock Inputs and Outputs

Name	Units of Measure	Name	Units of Measure
BodyErrAttX		BodyControlTorX	
BodyErrAttY		BodyControlTorY	
BodyErrAttZ		BodyControlTorZ	
BodyErrRateX		Rwa1ComTor	
BodyErrRateY		Rwa2ComTor	
BodyErrRateZ		Rwa3ComTor	
		BodyFilteredTorX	
		BodyFilteredTorY	
		BodyFilteredTorZ	

Notes

```
The wheel torque commands are generated in five steps:

   1) The measured body rate is used to calculate the body momentum.

   2) The controller generates torque commands.

   3) The torque commands are filtered (if necessary).

   4) The three body-axis torque commands are converted to three wheel
      torque commands.

   5) The three wheel torque commands are scaled such that none of them
      torque saturate the wheels while preserving the desired torque
      direction.
```

Revision History

Figure 10: Portion of Sample Documentation Web Page

Findings

DocumentIt allowed the team to eliminate the almost full time position of an analyst devoted to creating and maintaining an algorithm document. In addition, because the documentation and automatically generated code came from the same source, the HiFi simulation, the documentation has the same names and a similar format as the AutoCode output. Much less time was spent on algorithm documentation on MAP than on TRMM (and probably RXTE as well). The documentation benefited from the discipline imposed on the simulation by the flight software requirements.

However, the process was not without problems. There were three major bugs found in DocumentIt within five months of each other, and that gave the toolkit a black eye within the GNCC. All were officially recorded as bugs, and all were given call reference identifications. All the bugs were fixed, but not until the team had raised serious concerns about the product to ISI. The initial response had been that ISI would fix it in the next version, especially since MAP's version was on a platform that wasn't well supported by ISI. In addition, there were several typos in the documentation for DocumentIt that required calls to technical support to resolve. Some of the MAP algorithm documentation is incomplete because the units on all of the block inputs and outputs are not labeled. Doing this labeling requires some form of configuration management from the beginning of the design (such as the use of naming conventions) that was lacking in the documentation part of HiFi.

The resulting documentation was primarily block diagrams and brief descriptions of the block diagrams. The software engineers had to learn how to read block diagrams and how to interpret the block functions, in keeping with the philosophy of a smaller team and broader individual responsibility of the MIDEX program. A mathematical appendix had to be written that explained some of the math behind some predefined SystemBuild blocks.

LINEAR ANALYSIS OF SYSTEMS USING XMATH AND SYSTEMBUILD

As mentioned above, the mathematical capabilities of XMath allow the linear analysis of either a simple linear system or a more complex nonlinear model developed in SystemBuild. Systems can either be analyzed in a simple Single Input, Single Output (SISO) method or as Multi-Input, Multi-Output (MIMO) systems. ISI's Interactive Control Design Module (ICDM) allows an engineer to design robust controllers and digital filters graphically using one of several modern techniques. Also, there are modal reduction algorithms that can be used to determine the significance of specific eigenvalues (flexible modes) in a model. In general, these mathematical tools represent the capabilities one would expect in a controls analysis toolkit.

Although the ICDM allows for design of controllers of reasonably complex systems, the requirements of the MAP Attitude Control System did not warrant a complicated design. In fact, the relatively loose pointing requirements in each of the control modes and the lack of disturbance environment for MAP allowed the team to design low bandwidth controllers with robust gain and phase margins using conventional SISO controllers. Thus, the controllers employed on the MAP mission required only PID designs with additional filtering to suppress any control/structure interaction instabilities caused by the significant flexible modes of the spacecraft. The primary analyses required on the MAP mission included designing and determining the gain and phase stability of each of the controllers, selecting significant modes to include in the flexible body model, and designing appropriate filters, if needed, to suppress the control/structural interaction.

Conventional Analysis Method

The standard practice for performing the analyses described above would involve a combination of hand design and analysis using two in-house software packages. The first, the Modal Significance Analysis Program (MSAP) takes the output of a NASTRAN model and allows the analyst to use one of several selection methods (including Frequency, Modal Gain, Peak Amplitude, and Gregory's Method) to select four or more eigenvalues to include in the final flexible body stability analysis. Since higher fidelity NASTRAN models are typically not available during the initial design and analysis stages, the initial design is completed on a rigid body model with the controller bandwidth set a decade higher than the lowest expected first flexible mode.

Since a simple PID design is typically sufficient for the spacecraft controllers used, that design can be hand calculated with the knowledge of the controller's expected bandwidth and performance. That design is then analyzed using the second in-house software package, INCA (Interactive Controls Analysis). The analyst programs in the controller using INCA's simple S and Z domain language, after which a root locus, frequency domain, or time domain analysis can be performed. Since INCA allows users to develop their systems in external source files that are input to the program, it is quite easy to modify an existing system to analyze a new controller. The resulting gain and phase margins are typically fed back into the HiFi simulation (e.g. a 6 dB gain margin would be modeled by doubling the gain) to double-check the analysis.

Once the NASTRAN model is completed, if a digital filter is designed, both the flexible modes and filter can be input into the INCA model. Flexible modes are modeled as second order modes with a 0.1% damping, while digital filters are typically modeled in the Z domain. INCA includes some additional functionality that makes the analyst's job easier: an N* analysis tool that allows hybrid systems (part analog, part digital) to be analyzed without completely converting the S domain to Z domain, and describing functions that help analyze non-linear functions like deadbands or pulse-width modulators (PWMs).

Analysis Using XMath and SystemBuild

As one of the main goals was to use the MATRIXx suite for all of MAP's analysis and design tasks, some effort was made to use the state space tools available in XMath. The three axis controllers for Observing mode and Inertial mode were designed using the linear quadratic regulator function. The weights were picked to give good steady state performance with acceptable control torque. The resulting gain matrix was close enough to diagonal that there are, in effect, three SISO controllers on board. The control laws in the other spacecraft modes were designed using conventional practice. Once these control laws were implemented into HiFi, the ACS team attempted to perform a frequency response analysis on each of them.

Two problems were found in this analysis. The first was that the describing function capability found in INCA did not exist in XMath. This was resolved by using the INCA program to analyze the thruster modes' PWM, then comparing the frequency response of the linearized controller analyzed in XMath with the modified stability point described in INCA. Although cumbersome, this was an effective workaround.

On the other hand, the second problem was far more difficult to detect and work around. After performing the frequency analysis on each of the controllers, the analysts decided to verify them by adding back in the correct gain (or phase lag) to make the system unstable. The simulations turned out to be about 6 dB more stable than the frequency analysis had predicted. After a quick double-check on the analysis, analysts re-ran the controllers through the INCA program to find there was an error in the initial XMath results. After several calls to ISI, the problem was identified. In order to correctly analyze a hybrid system in Xmath, one must either fully discretize the entire system (including the continuous plant) or first analyze the continuous system, then use the resulting frequencies to perform the hybrid frequency domain analysis. Either approach works, but both are unnecessarily complicated for a linear analysis.

One other discovery occurred after flexible modes were added to the analysis. Unlike the functionality of INCA, the frequency analysis in XMath kept a fixed frequency step size, even as it approached an eigenvalue. The unintended consequence of this mathematical algorithm is to possibly mask potential stability implications of extremely lightly damped eigenvalues (damping ratio < 0.01). Discussion with analysts familiar with other analysis packages determined that this shortcoming exists elsewhere and that in INCA GSFC is fortunate to have the frequency step size change automatically.

OTHER CONSIDERATIONS WITH USING MATRIXX

Over the course of the development effort, specific business issues have arisen that have affected the use of the MATRIXx product. Although it was originally desired in the ideal development practice, Autocoding the Hybrid Dynamic Simulator for use in flight software testing was decided against early in the program. The reason may be obvious to the reader: there are significant risks that an error in the HiFi simulation would propagate itself into the flight code **and** the test code, allowing a possible mistake to go undetected.

For this reason, both the HDS and a separate Safe-Hold controller were independently designed and hand coded. There are cases where the approach called for in Figure 2 is warranted, but given the lack of maturity of this process, a more conservative approach was chosen this time.

The second major issue deals with functionality that does not currently exist in the MATRIXx toolkit. It was the team's experience that adequate configuration management (CM) tools do not exist within the toolkit itself on which to base any CM reliance. Instead, the maintenance of the HiFi simulation by a single engineer was found to be a prudent practice. Unfortunately, this has the undesired effect of limiting the simulation design and maintenance effort to a single engineer once the simulation has been CM'd.

The selection of that CM point should be carefully considered, as well. The MAP team discovered that the AutoCode generated in MATRIXx 6.0 is not backwards compatible with that generated in MATRIXx 5.0. Thus, teams wishing to maintain code generation capabilities across software builds have a choice to make: maintain older versions of the software for code generation or fully regression test the software after every build change. This question is made even more difficult by the fact that earlier versions of MATRIXx, including version 5.0, are not guaranteed to be Year 2000 compliant, necessitating a move to the current version. At this point, that decision is still under consideration by the MAP team.

Finally, a team should be aware of ISI's tier system before purchasing a MATRIXx license. The tier system effectively ensures that more product support is available for the popular platforms (Windows 95, Windows 98, Windows NT, H/P-UX, and Sun Solaris), with more limited support for other workstations. As users who've experienced MATRIXx support on tier-three IBM r6000 running AIX and DEC Alpha machines running DEC UNIX, the MAP team has found the product expertise on those machines to be limited. In fact, MAP team found many problems with the toolkit by simply being the first (or only) group to use that functionality on the tier three platforms.

CONCLUSIONS AND LESSONS LEARNED

Overall, the MAP ACS team considers this development effort to have been a success, with many lessons learned for future developments. Another in-house project, Triana, already has saved development time and effort by building on the MAP work. Upon reviewing the MAP effort, the following conclusions were drawn:

- **Xmath/SystemBuild is an excellent tool for nonlinear simulation development.** The tool has allowed a high fidelity simulation that is useful to an entire design team to be developed in a fraction of the time of a FORTRAN simulation. Its object-oriented design paradigm is key to further improvements by effectively allowing true simulation reuse. Its ease of use and plotting capabilities make it an appropriate tool for flight software testing comparisons, as well.

- **AutoCode effectively generates flight-quality software.** The use of AutoCode, when properly scoped, can be used to reduce the software design, code and review effort. However, one must be careful to note that the code is generated conservatively and it therefore produces larger and slower executables than does hand coding (though this issue may not be as true with the greater optimization capabilities available in later versions of AutoCode).

- **Documentation can be automatically generated from SystemBuild blocks.** The process of creating a single point of design enables the designer document the design in one place. Obviously, some care has to be taken to make sure the simulation is well documented (which was not necessarily the practice previously) and that templates are built to supply all of the relevant information. Nonetheless, this is a significant improvement over manual translation of algorithms between analysts and software developers.

- **The linear analysis capabilities of MATRIXx, although excellent in the field of interactive design, could stand some improvement in the field of frequency domain analysis.** As mentioned above, the process for performing a frequency domain analysis on a hybrid system is neither obvious nor well-documented. An improvement in this area and the addition of some functionality as described above is warranted for future releases.

- **The lack of backwards compatibility in AutoCode makes the determination of a time to put a product under configuration management important.** This determination should take into account upcoming releases of the toolkit, especially if the AutoCode functionality is used. Otherwise, an effort will have to be made to determine the effects of a version change or to maintain a version throughout a project. Effects of Y2K non-compliance must be taken into account with older versions currently being used.

- **As with many COTS software products, selection of the hardware platform plays a role in the quality of product service offered.** Although the MAP team has not experienced an unwillingness on ISI's part to support the MATRIXx software on the DEC Alpha workstation, the quality of that service suffers from a lack of customer demand. Given the costly lease-type licensing required by ISI, new users would be wise to select a license for a tier one platform.

The true benefit of this exercise had less to do with the specifics of the MATRIXx software and more to do with the change in philosophy regarding the development process. Engineers on the MAP team abandoned the specialization found on previous programs and developed generalized skills that were used in developing several of the deliverables. MATRIXx worked as an enabling tool to accomplish this philosophy change: it allowed analysts, flight software developers, systems engineers and testers to share a common design point. This increased team communication and allowed the team to be more tightly integrated in its design and development work. To that end, the use of this tool was a great success.

REFERENCES

[1] MAP Home Page: http://map.gsfc.nasa.gov.

[2] Andrews, S. F., et al., "MAP Attitude Control System Design and Analysis", Flight Mechanics and Estimation Theory Symposium, 1997.

[3] O'Donnell, J. R., et al., "Development and Testing of Automatically Generated ACS Flight Software for the MAP Spacecraft", International Symposium on Space Flight Dynamics, 1999.

[4] McComas, D. C., et al., "Using Automatic Code Generation in the Attitude Control Flight Software Engineering Process", 23rd Software Engineering Workshop, Goddard Space Flight Center, 1998.

[5] MAP ACS Algorithm Home Page: http://sandrews.gsfc.nasa.gov/mapdoc/cover.html.