A General Mission Independent Simulator (GMIS) and Simulator Control Program (SCP)

Paul L. Baker (GST Inc.)
J. Michael Moore (NASA/GSFC)
John Rosenberger (CTA Inc.)

The Purpose of GMIS and SCP

GMIS is a general-purpose simulator for testing ground system software. GMIS can be adapted to any mission to simulate changes in the data state maintained by the mission's computers. GMIS was developed in Code 522 NASA Goddard Space Flight Center. The acronym GMIS stands for *GOTT Mission Independent Simulator*, where GOTT is the *Ground Operations Technology Testbed* . Within GOTT, GMIS is used to provide simulated data to an installation of TPOCC - the Transportable *Payload Operations Control Center* . TPOCC was developed by Code 510 as a reusable control center. GOTT uses GMIS and TPOCC to test new technology and new operator procedures.

Ideally, mission operations staff should have a variety of simulators to serve several purposes:

- Prediction - compute the future state of a system
  - Evaluate the effects of a proposed operational step, i.e., to answer "what if" questions.
  - Verify that the planned steps will cause operations that lie within safety and other operational constraints.
- Test – supply a time-variable system state to exercise subsystems.
- Training – create a realistic environment for training staff.

In practice, missions that use TPOCC have one or more simulators. Consequently, GMIS was not developed to fill a void; rather, it was developed to offer an alternative with certain advantages:

1) Convenience - GMIS is easy to setup and use.
2) Extensible - it is easy to add more simulation functions.
3) Speed - eventually, we expect GMIS to run very quickly.

In the present version, we have not achieved these goals in equal measure. The convenience factor is high, but the speed seems modest. The features that make GMIS extensible are useful, but there is room for improvement. In this report, we will relate some feedback from current GMIS users and indicate how we plan to improve the simulator in these three areas.

The GMIS manages the timing and external data links for optional simulation modules. It accepts any number of compiled or interpreted modules. Compiled modules are written in C or C++. The interpreted modules are written as procedures in TSTOL - *TPOCC System Test and Operations Language* . This language is familiar to flight operations team members, but it is not especially easy to use. In fact, programmers often find it difficult to use because it looks familiar but has a different syntax compared to programming languages. For this reason, the project developed the SCP as a convenience feature.

The SCP is a graphical, syntax-aware editor for TSTOL. Although SCP is really a simulation script editor, its name stands for *Simulator Control Program,* for historical reasons. SCP helps you write a correct TSTOL procedure and then lets you run it with a click of a button. SCP has an embedded copy of the TSTOL interpreter so that it can detect and report syntax errors locally. Finally, SCP reads and displays all the variable names in the data server's database. That feature helps the user find the correct spelling for system variable names.

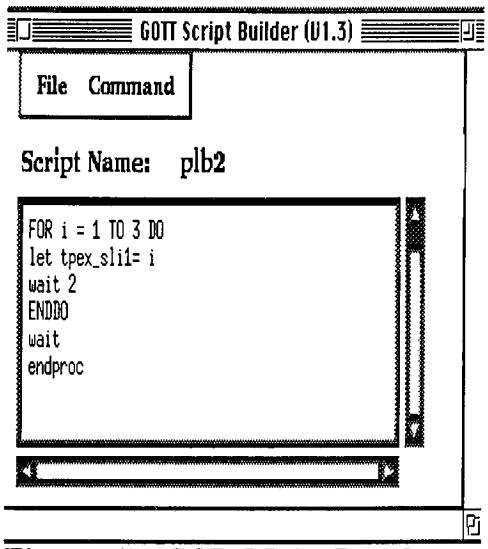

Figure 1: SCP Main Panel

SCP and GMIS Interaction Panels

Simple and easy to use Motif control panels are responsible for much of the convenience of GMIS/SCP. The panels strive for the same look-and-feel as the panels that are used in the TPOCC control centers.

The SCP has a main panel, called the Script Builder, that is used to edit TSTOL scripts. Figure 1 shows a copy of the panel at a point where the user has completed a simple script.

This script will loop three times with two seconds per loop, and it will set the value of the loop index, `i`, in the system variable, `tpex_sli1`.

The script appears within a Motif Text Widget. All of the Widget's editing commands are available. The File Menu has the usual options for saving and retrieving copies of the script. The Command Menu has only one option: Execute. When the user selects that option, the script runs and sends data values to GMIS. From GMIS, the updated values find their way to the data server.

GMIS always shows the updates to values when it receives them from SCP. The GMIS panel is shown in Figure 2 just after the SCP has executed the script in Figure 1. Compiled simulation modules are usually designed for a higher throughput and could swamp the display with output. Consequently, GMIS does not automatically show updates for such modules. However, compiled simulation modules can write progress messages to the display, if they wish.

The SCP has two panels that help a user write TSTOL. Suppose we want to add another statement to the procedure. We only need to click the mouse at the point where we want the new statement to appear. Then, we can go to the Statement Builder panel and pull down the Script menu. That menu shows the basic statements of the TSTOL language as illustrated in Figure 3.

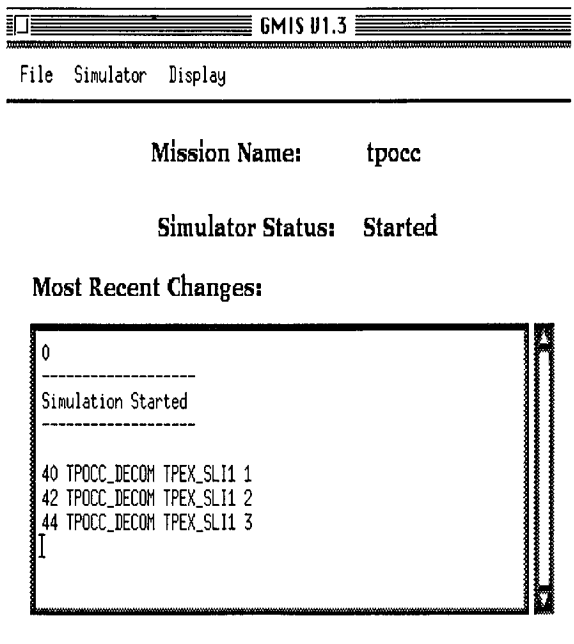

Figure 2: GMIS Main Panel

Many of the TSTOL constructs in the menu require multiple lines. For example, all the block structures have a starting and ending statement. In those cases, the statement builder will insert multiple lines and the user simply clicks within the block to add the statements that belong there. Moreover, some of the TSTOL constructs require parameters. That is indicated in the menu by a series of periods after the name. The Statement Builder helps with two of those: `Let` and `For`. When the user selects one of these constructs, the main area of the Statement Builder changes to display a fill-in form with the required parameters.

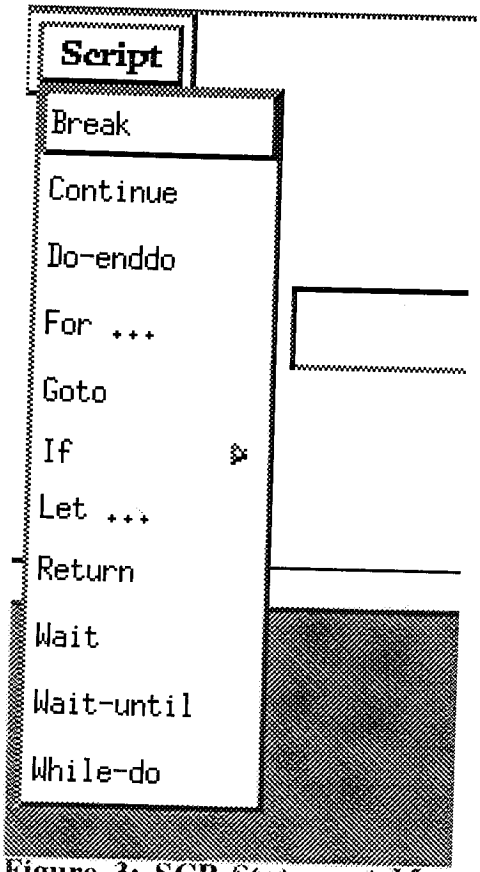

Figure 3: SCP Statement Menu

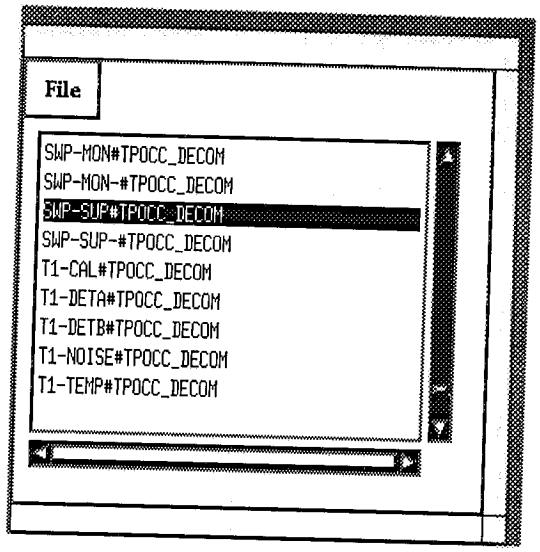

Figure 4: SCP Data Points Panel

When the script needs a TPOCC system variable name as a parameter, the user can type the name or click on the name in the Data Points panel. The Data Points panel lists all the current data server variables. For example, the user has just clicked `SWP-SUP` in Figure 4. Just before the name was selected, we started a `Let` statement in the Statement Builder window. When the name is selected, SCP copies it into the first entry field of the `Let` statement shown in Figure 5.

The statement that you construct in this window will be copied into the script when you click on the Apply option. In this case, we made an error - `foobar` is not a symbol TSTOL will recognize. We inserted this statement anyhow to produce error messages intentionally. When we execute a script, SCP brings up another window to show the dialog with the TSTOL process. Figure 6 shows the dialog for this script and the TSTOL error messages.

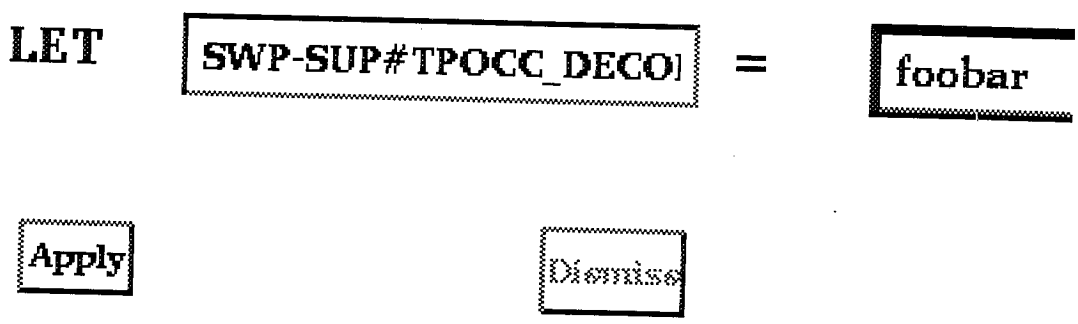

Figure 5: SCP Statement Builder - Parameter Form for "Let" Directive

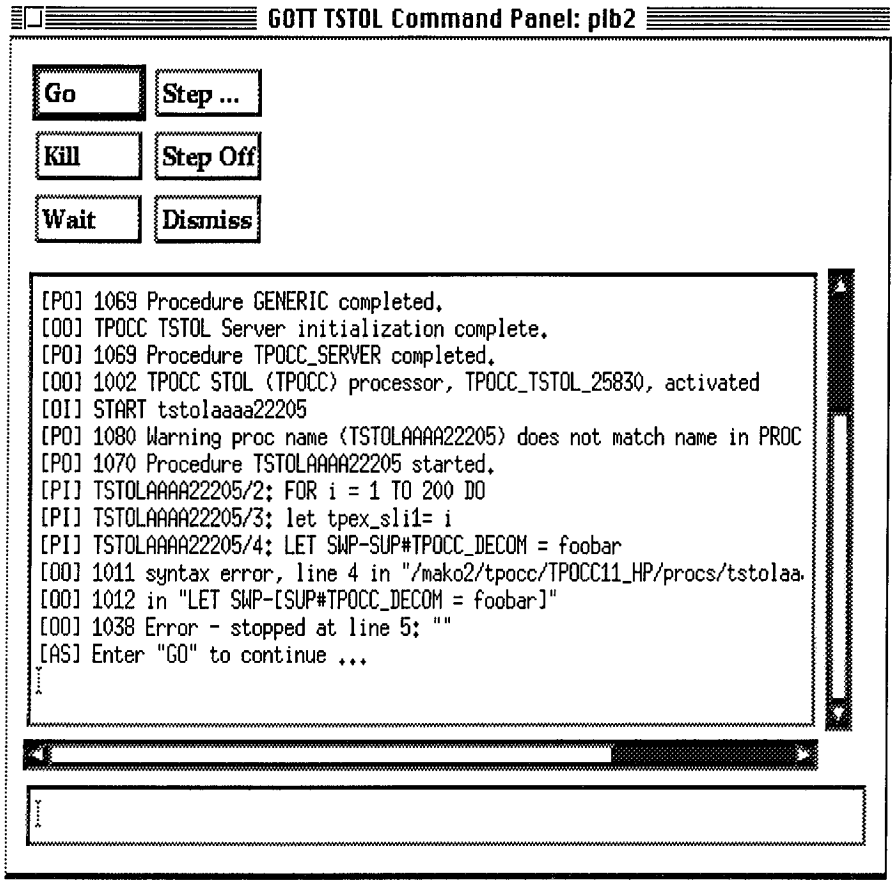

Figure 6: SCP - TSTOL Script Execution Panel

GMIS Software Design

The design for GMIS has several distinguishing features that are illustrated in Figure 7. A major design decision was to provide and maintain two copies of the system variables. The simulator modules read from one copy and write to a second. At the end of the time step, any variable that has changed is forwarded to the data server. At the same time, the first copy is updated from the second. The point of this feature is that it allows multiple modules to contribute to the next system state. The full state is written out at once after all the simulation modules have taken a turn.

The current version allows any number of internal, compiled simulation modules. In addition, the SCP can interpret TSTOL procedures and send new data values to the GMIS. The SCP is running asynchronously with the GMIS, but the internal modules are synchronized to a strict clock. The internal timing is maintained by examining the system clock, computing the amount of idle time between cycles, and then programming the Xt System software for the required delay. The basic timing cycle is illustrated in Figure 8.

The current version has the feature - or the implementation restriction, depending upon your viewpoint - that it is single threaded. Thus, the software cannot receive from the SCP when it is running a model, nor will it start its scheduled time step while a read operation is in progress. In future versions, we would like to be able overlap the data server communication with the other operations by providing two or more separate threads.

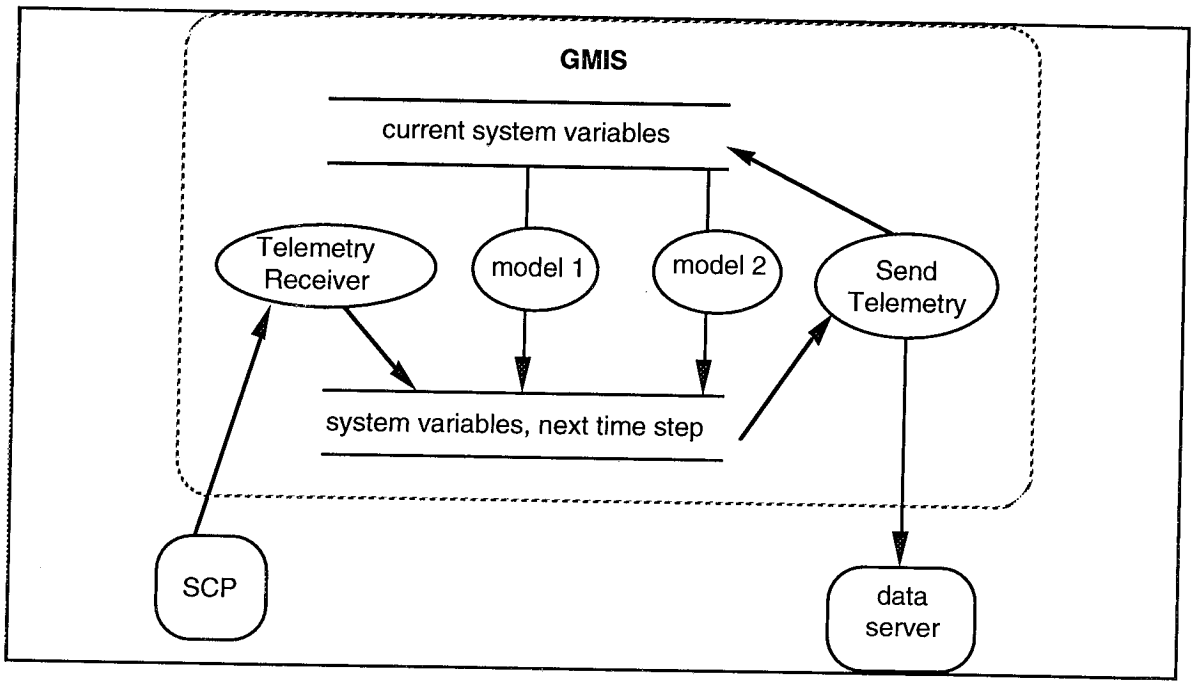

Figure 7 - The Data Flow in GMIS

A programmer who wants to extend GMIS with compiled modules must use C++ in the current version of GMIS. C++ classes provide most of the routine simulator functions such as timing and data communications. The functions are inherited by the simulator modules in the following way. To write a new module, the programmer defines it first as a class that inherits from the *ModelEngineInterface* class. Then the programmer makes a single instance of the class, which is the actual simulator module. When the instance is constructed, it will connect itself with the GMIS infrastructure automatically.

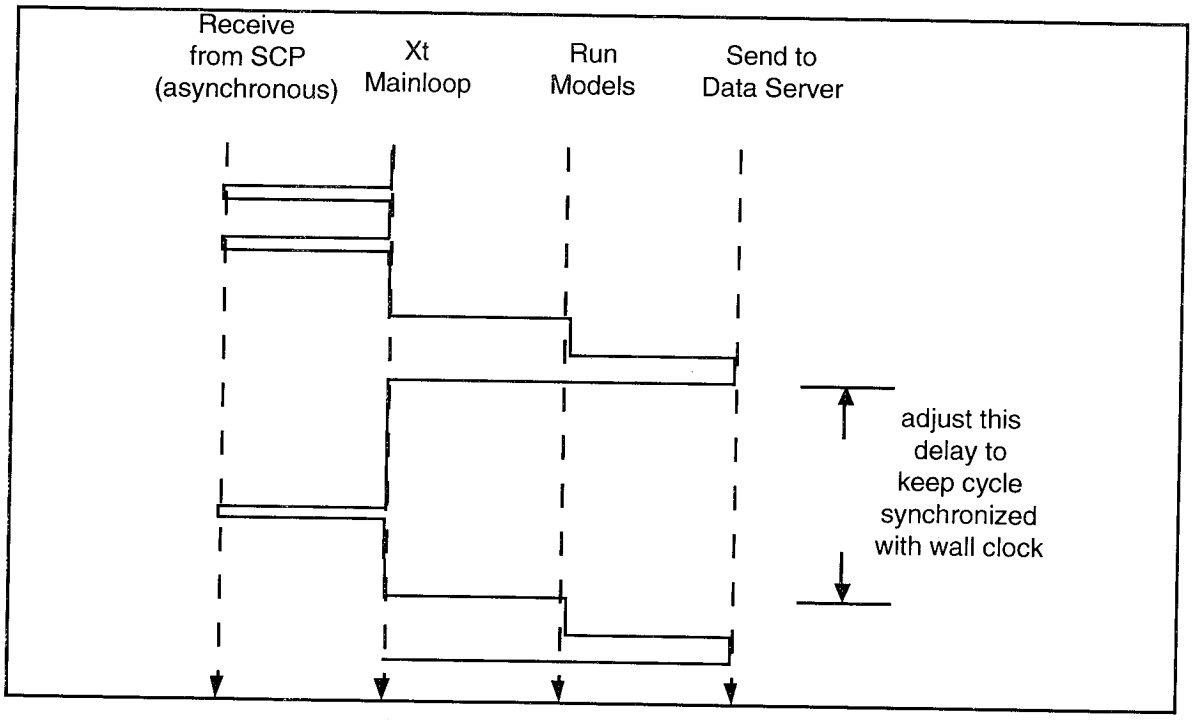

Figure 8 - GMIS Timing Diagram

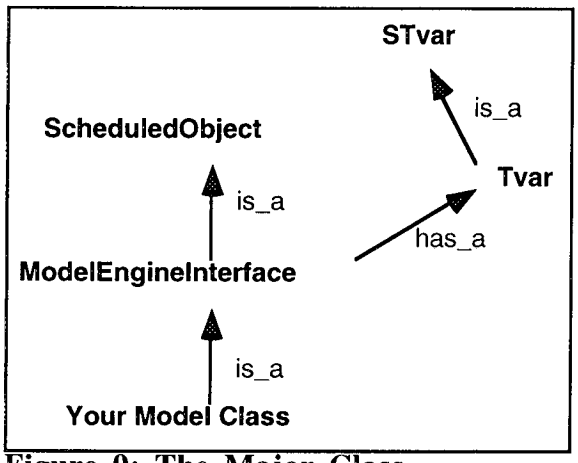

Figure 9: The Major Class Relationships in GMIS

The *ModelEngineInterface* class inherits its timing properties from a utility class, called *ScheduledObject*, and it acquires access to the system variables by a composition operation. In simple terms, it contains a pointer to a list of system variables. There is a full set of utility functions that come with the list as part of a class called *Tvar*. The GMIS effort borrowed the *Tvar's* from another project. Because the*Tvar's* didn't do everything we needed, we made *Tvar* a subclass of a new class *STvar* that added the new functions. This programming trick avoided any modifications to *Tvar*, initially anyhow. We will discuss software maintenance experiences later. The class relationships are summarized in the Figure 9.

Features Provided, Features Used

The GMIS simulator has been in operation for about 18 months. It has been used in experiments in the GOTT and by three other projects. The feedback has been surprising because the features that are used the most were not the most important during the development. Also, some of the most important features during development are used little, or tend to be in the way of use. Finally, we had to add features for two projects because there was a reasonable need that was not satisfied by the baseline version. As a result, there is currently a baseline version and two variations. The feature sets of these versions are summarized in the Table 1.

As we expected, the built-in connection to the data server has made GMIS a valued tool for testing TPOCC related software. The interpreted TSTOL modules have proven more valuable than expected. On the other hand, the class libraries have fared poorly in practice. The classes are not simple enough to encourage reuse, although a lack of familiarity with C++ may also be a factor. Moreover, the C++ class for data server access, `Tvar`, has proven very hard to maintain. Also, the `Tvar` class only handles asynchronous data and it proved impossible to extend it for synchronous data. Consequently, we had to write a special version of GMIS for a project that requires synchronous data simulation . Our conclusion is that it may not be a good idea to wrap a C++ class around a C library that is not well understood. This is certainly the case with `Tvar` and the TPOCC data services library.

A surprising requirement has been the need for simulations in which the simulation calculation is performed off-line and written to tape. The GMIS must then read the tape and supply the data at a steady pace controlled by the simulation clock. In principle, there is no reason that an off-line calculation could not be performed on-line. Indeed, one could keep the simulation software wherever it is developed and provide simulated data over a network on demand. In practice, software does not move freely and networks are not always connected to each other. For now, this requirement is a real one for NASA and it required a special version of GMIS.

Table 1: Summary of GMIS Features and Their Extent of Use

Feature	Version 1.0	Version 1.X
1. Asynchronous Data to Data Server	✪	
2. Synchronous Data to Data Server	○	✪
3. Local Copies of System Variables	●	
4. Compiled Simulation Modules, Static Binding	●	
5. Compiled Sim. Modules, Dynamic Binding	○	
6. Reusable C++ Classes	●	
7. TPOCC Data Server Classes	●	
8. Interpreted TSTOL Simulation Modules	✪	
9. Simulation Tape Playback	○	✪

Key:

- ○ feature not supported
- ● feature supported but not used
- ✪ feature supported and used extensively

Note:

Version 1.0 is the baseline version; Version 1.X is a notation for the several specialized variants.

The Future of GMIS/SCP

As the preceding summary shows, there are valuable features in GMIS that complement other simulation facilities. However, there is still room for improvement in the three qualities we deem important: convenience, extendibility, and speed.

In the area of convenience, it has long been our goal to have fast, compiled, simulation modules that we can start on demand while GMIS is running. These dynamically loaded modules would give the experimenters in the GOTT laboratory more flexibility in their tests. This feature should appear in the next version. The overall architecture of GMIS will then realize the design shown in Figure 10. Presently, all the compiled modules are linked statically, but that will change to dynamic loading in the next version.

In principle, the module that connects GMIS and SCP is just another producer module. Similarly, the module that connects to the data server is just another data consumer. In practice, it may be difficult to build a dynamically linked module without revising the extensive body of TPOCC code that is reused in these modules. Consequently, these modules will always be statically linked, but compilation options will determine whether the modules are present or not.

The extendibility of the current version is based on C++ classes that offer a variety of scheduling features as well as an encapsulated access to TPOCC data services. Only the simplest scheduling features are used, however, and the encapsulated access is often more of an obstacle than an advantage. For this reason, the next version of may be written in C, without classes.

The speed of GMIS should be improved considerably if we could overlap simulation calculations with system variable output. We plan to explore this possibility as DCE - Distributed Computing

Environment - is phased in. DCE has a built-in capability for multiple threads that should support simultaneous network communication and computation.

For the future, the GOTT laboratory is not limited to TPOCC control center software. In the past, the laboratory has hosted OASIS software and we are currently experimenting with software from Storm Technology, Inc. For this reason, the GMIS should be independent from a particular control center.

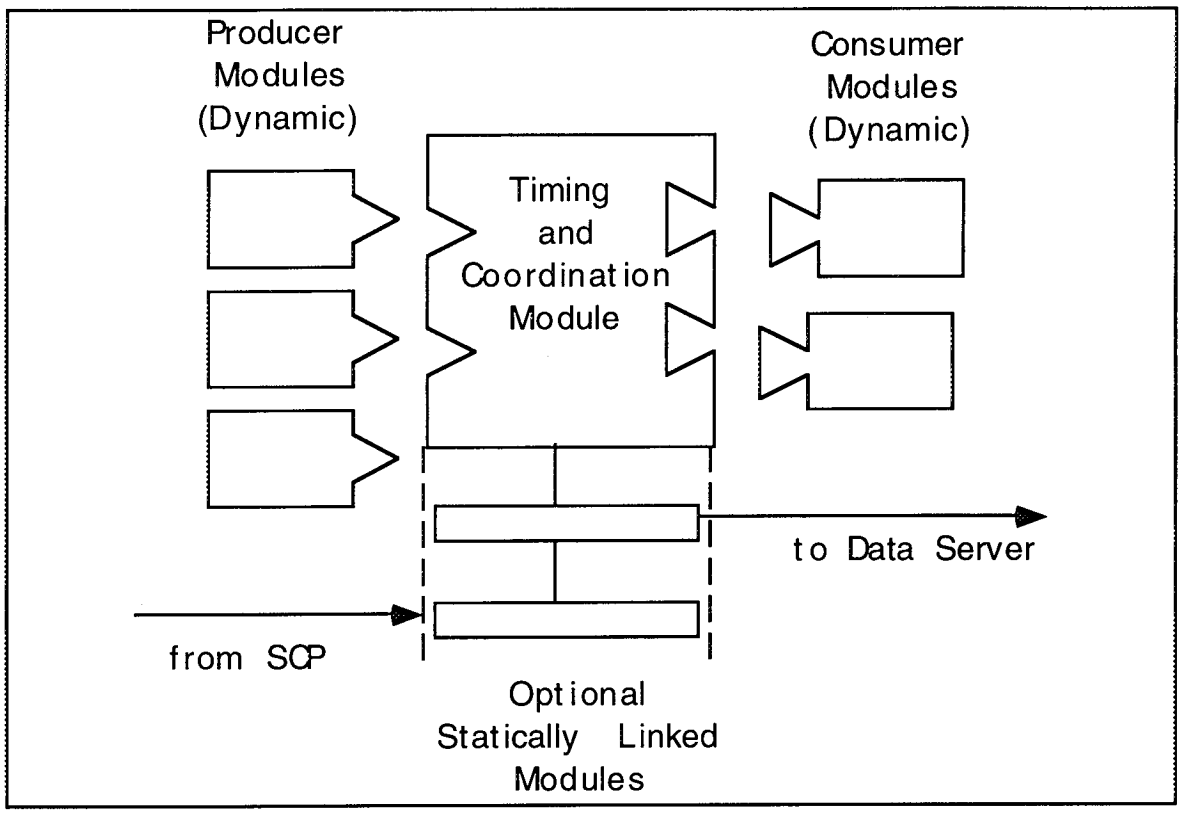

Figure 10: Overall Architecture for GMIS

Contact Information

Paul L. Baker
Global Science and Technology, Inc.
6411 Ivy Lane, Suite 610
Greenbelt, MD 20770

John Rosenberger
CTA Incorporated
6116 Executive Blvd.
Rockville, MD 20852

J. Michael Moore
Software Automation Systems Branch
Code 522
NASA Goddard Space Flight Center
Greenbelt MD 20771

Point of Contact

Paul L. Baker
Email: pbaker@gst.gsfc.nasa.gov
Tel. (301) 474-9696