PU/NE-98-10

SOFTWARE DESIGN AND IMPLEMENTATION DOCUMENT FOR THE GENERAL INTERFACE IN THE COUPLED CODE

Douglas A. Barber, Thomas J. Downar

School of Nuclear Engineering
Purdue University
W. Lafayette, IN 47907-1290

submitted to:

Division of Systems Technology
Office of Regulatory Research
Nuclear Regulatory Commission
Washington, D.C. 20555-001

Contract NRC-04-96-060, Task 5c
Performed Under Subcontract to SCIENTECH, Inc.

May 1998

I. Introduction

This document provides a description of the software design necessary to meet the Software Requirements Specification for the General Interface[1]. The General Interface code, which utilizes the FORTRAN-90 (F90) standard, is designed as a self-contained process which communicates with the thermal-hydraulic and neutronic processes using standard message-passing constructs. Specifically, the Parallel Virtual Machine (PVM) package is utilized to control all communication operations.

Section II of this document discusses the variables utilized by the interface. Section III then provides the software design description for the functional units and the error checking unit contained within the General Interface. Finally, Section IV discusses the process flow through the functional units, the input/output requirements, and the exit handling procedure.

II. Variable Definition

The scalar and array variables used in the General Interface are grouped into five separate classes, and are described in Appendix A. The first class, shown in Table 1, relates to the variables needed for process control with PVM. The second and third classes include those variables (e.g. the vectors and control buffers) which are received from the thermal-hydraulic and neutronic processes, as listed in Table 2 and Table 3, respectively. The vectors in these two classes, which contain the space-dependent property data, will utilize SI units.

Table 4 describes the variables associated with the two permutation matrices, both of which are received from either the thermal-hydraulic or neutronic process. In an effort to minimize the amount of storage required for the matrices used by the General Interface, the Coordinate Storage Format[2] is utilized. This format requires three arrays corresponding to the non-zero elements in the matrices (`matvalth`, `matvaln`), the row identifier for each element (`matrowth`, `matrown`), and the column identifier for each element (`matcolth`, `matcoln`). Finally, Table 5 describes the variables associated with the error checking.

These variables are declared in module `GI_Var_Decl`, which is used by each subroutine in the General Interface. For the utilization of PVM constructs, the header file 'fpvm3.h' is also included. In addition, for the declaration of variable arrays, advantage is taken of the dynamic memory allocation of F90, and thus static dimension parameters are not required.

III. Software Design

The General Interface code contains three functional units and one error checking unit. The first functional unit initializes the PVM process and establishes the communication with the thermal-hydraulic and neutronic processes. Following this setup, the initialization unit manages the communication of initial control information between the thermal-hydraulic and neutronic processes. In addition, both permutation matrices are received from either the thermal-hydraulic or neutronic process and stored for use in the two mapping units. The second and third functional units handle both the transfer of time-dependent control information and the mapping of property data between the thermal-hydraulic and neutronic processes. Finally, in the error checking unit, tests are performed to ensure that the information sent to the General Interface is accurate.

For organization of the subroutines, advantage is take of the F90 `MODULE` construct. As will be shown in the following subsections, modules are utilized to group subroutines similar in function. Specifically, the subroutines associated with the initialization are contained within module `GI_Init_Calc`, the subroutines associated with the time-dependent mappings are contained within module `GI_Time_Calc`, the subroutines associated with the communication of control buffers between processes are contained within module `GI_Comm_Buf`, and the subroutines associated with the error checking are contained within module `GI_Error_Check`. A brief description of the subroutines and modules, along with a calling tree, is provided in Appendix B.

III.A. Initialization

The initialization procedure is controlled by the subroutine, `GI_Init()`. The subroutines called by `GI_Init()` are contained within module `GI_Init_Calc`.

The first step of the initialization is to invoke the PVM process for the General Interface, and obtain the process ID. This process is then added to a dynamic process group, which also will include the thermal-hydraulic and neutronic processes. In order to ensure communication coherency, the process IDs are communicated between the three processes using predetermined message tags (`mtypegi, mtypeth, mtypen`). Subroutine `GI_Obtain_IDs()` performs these functions for the General Interface, and a similar routine is needed in the thermal-hydraulic and neutronic codes. As an example, the following depicts the logic needed in each code:

General Interface Process:

```
c Establish communication with the Thermal-Hydraulic
c and Neutronic processes.
      ntasks = 3
      mtypegi = 1
      mtypeth = 2
      mtypen = 3
      group = 'procs'
c Enroll General Interface process in PVM.
      CALL pvmfmytid( tidgi)
c Join the dynamic process group and wait for the thermal-hydraulic
c and neutronic processes.
```

```
      CALL pvmfjoingroup( group, inum)
      CALL pvmfbarrier( group, ntasks, info)
c Broadcast ID to the thermal-hydraulic and neutronic processes.
      CALL pvmfinitsend( PVMDEFAULT, info)
      CALL pvmfpack( INTEGER4, tidgi, 1, 1, info)
      CALL pvmfbcast( group, mtypegi, info)
c Obtain IDs from the thermal-hydraulic and neutronic processes.
      CALL pvmftrecv( -1, mtypeth, timeout, 0, info)
      CALL pvmfunpack( INTEGER4, tidth, 1, 1, info)
      CALL pvmftrecv( -1, mtypen, timeout, 0, info)
      CALL pvmfunpack( INTEGER4, tidn, 1, 1, info)
```

Thermal-Hydraulic Process:

```
c Establish communication with the General Interface process.
      ntasks = 3
      mtypegi = 1
      mtypeth = 2
      group = 'procs'
c Enroll Thermal-Hydraulic process in PVM.
      CALL pvmfmytid( tidth)
c Join the dynamic process group and wait for the General Interface
c and neutronic processes.
      CALL pvmfjoingroup( group, inum)
      CALL pvmfbarrier( group, ntasks, info)
c Receive ID broadcasted from the General Interface process.
      CALL pvmftrecv( -1, mtypegi, timeout, 0, info)
      CALL pvmfunpack( INTEGER4, tidgi, 1, 1, info)
c Send ID to the General Interface process.
      CALL pvmfinitsend( PVMDEFAULT, info)
      CALL pvmfpack( INTEGER4, tidth, 1, 1, info)
      CALL pvmfsend( tidgi, mtypeth, info)
```

Neutronic Process:

```
c Establish communication with the General Interface process.
      ntasks = 3
      mtypegi = 1
      mtypen = 3
      group = 'procs'
c Enroll Neutronic process in PVM.
      CALL pvmfmytid( tidn)
c Join the dynamic process group and wait for the General Interface
c and thermal-hydraulic processes.
      CALL pvmfjoingroup( group, inum)
      CALL pvmfbarrier( group, ntasks, info)
c Receive ID broadcasted from the General Interface process.
      CALL pvmftrecv( -1, mtypegi, timeout, 0, info)
      CALL pvmfunpack( INTEGER4, tidgi, 1, 1, info)
c Send ID to the General Interface process.
      CALL pvmfinitsend( PVMDEFAULT, info)
      CALL pvmfpack( INTEGER4, tidn, 1, 1, info)
      CALL pvmfsend( tidgi, mtypen, info)
```

The result of this example is that `tidgi` is associated with the General Interface, `tidth` is associated with the thermal-hydraulic module, and `tidn` is associated with the neutronics module. With the process IDs for each module known explicitly, the communication initialization is complete. It should be noted that utilizing this design to establish the inter-process communication restricts the number of simultaneous jobs to one. In addition, the `pvmfbarrier` call does not have an associated "time-out". Therefore, if all three processes are not started, the code will be stalled indefinitely.

The function, `pvmftrecv`, which is used throughout the code and was shown in the above example, is a non-blocking receive which terminates if no message has arrived after "`timeout`" seconds. `timeout`, which is described in Table 1 of Appendix A, is set to 600 seconds.

The next step is to receive the initial control information from the neutronic process, which is controlled by the subroutine `GI_Buf_Init()`. The routines used to communicate the buffers (i.e. `GI_Recv_Bufn` and `GI_Send_Bufn`) are contained in module `GI_Comm_Buf`. The neutronic control information is sent to the General Interface as a structure composed of the buffers corresponding to each data type (i.e. `cbufn, lbufn, i2bufn, i4bufn, r4bufn, r8bufn`). Therefore, upon receipt of this structure, the data is unpacked and temporarily stored in memory. Most of the information transferred in the control buffers is needed only by the thermal-hydraulic process. However, the following locations are allocated for access by the General Interface:

`lbufn(1):`	Indication of an error in the Neutronic code.
`lbufn(2):`	Indication of a data error in the Neutronic-Specific Data Map Routine.
`lbufn(3):`	Indication of a PVM error in the Neutronic-Specific Data Map Routine.
`lbufn(4):`	Indication of a data error in the General Interface.
`lbufn(5):`	Indication of a PVM error in the General Interface.
`lbufn(6):`	Indication of normal calculation termination.
`lbufn(7):`	Indication of whether Neutronic process is sending the permutation matrices.

Once this information has been extracted, the buffers are packed back into a structure and sent to the thermal-hydraulic process. The following coding example depicts this procedure:

```
c Receive data structure from neutronic process.
      CALL pvmftrecv( tidn, mtypen, timeout, 0, info)
      istride = 1
c Extract buffer dimensions and allocate memory for the
c control buffers.
      CALL pvmfunpack( INTEGER4, dimbuf, 6, istride, info)
      ALLOCATE( cbufn(dimbuf(1)))
      ALLOCATE( lbufn(dimbuf(2)))
      ALLOCATE( i2bufn(dimbuf(3)))
      ALLOCATE( i4bufn(dimbuf(4)))
      ALLOCATE( r4bufn(dimbuf(5)))
      ALLOCATE( r8bufn(dimbuf(6)))
c Extract data type-dependent control buffers.
      CALL pvmfunpack( STRING, cbufn, 6*dimbuf(1), istride, info)
```

```
      CALL pvmfunpack( INTEGER4, lbufn, dimbuf(2), istride, info)
      CALL pvmfunpack( INTEGER2, i2bufn, dimbuf(3), istride, info)
      CALL pvmfunpack( INTEGER4, i4bufn, dimbuf(4), istride, info)
      CALL pvmfunpack( REAL4, r4bufn, dimbuf(5), istride, info)
      CALL pvmfunpack( REAL8, r8bufn, dimbuf(6), istride, info)


c extract any necessary information used by this routine

        ...

      istride = 1
      CALL pvmfinitsend( PVMDEFAULT, info)
c Pack buffer dimensions.
      CALL pvmfpack( INTEGER4, dimbuf, 6, istride, info)
c Pack data type-dependent buffers.
      CALL pvmfpack( STRING, cbufn, 6*dimbuf(1), istride, info)
      CALL pvmfpack( INTEGER4, lbufn, dimbuf(2), istride, info)
      CALL pvmfpack( INTEGER2, i2bufn, dimbuf(3), istride, info)
      CALL pvmfpack( INTEGER4, i4bufn, dimbuf(4), istride, info)
      CALL pvmfpack( REAL4, r4bufn, dimbuf(5), istride, info)
      CALL pvmfpack( REAL8, r8bufn, dimbuf(6), istride, info)
c Send structure to thermal-hydraulic process.
      CALL pvmfsend( tidth, mtypegi, info)
c Deallocate memory for the control buffers.
      DEALLOCATE( cbufn)
      DEALLOCATE( lbufn)
      DEALLOCATE( i2bufn)
      DEALLOCATE( i4bufn)
      DEALLOCATE( r4bufn)
      DEALLOCATE( r8bufn)
```

Once the neutronic initial control buffers have been communicated, the thermal-hydraulic initial control buffers (i.e. `cbufth`, `lbufth`, `i2bufth`, `i4bufth`, `r4bufth`, `r8bufth`) are received, the necessary information is extracted, and the buffers are then sent to the neutronic process. The coding needed to perform these tasks, which is contained in subroutines `GI_Recv_Bufth()` and `GI_Send_Bufth()`, is similar to that shown above and is not repeated here. Again, most of the information transferred in the control buffers is needed only by the neutronic process. However, the following locations are allocated for access by the General Interface

`lbufth(1)`: Indication of an error in the T/H code.
`lbufth(2)`: Indication of a data error in the T/H-Specific Data Map Routine.
`lbufth(3)`: Indication of a PVM error in the T/H-Specific Data Map Routine.
`lbufth(4)`: Indication of a data error in the General Interface.
`lbufth(5)`: Indication of a PVM error in the General Interface.
`lbufth(6)`: Indication of normal calculation termination.
`lbufth(7)`: Indication of whether T/H process is sending the permutation matrices.

The final step in the initialization process is to receive and store the two permutation matrices sent from either the thermal-hydraulic or the neutronic process, and subroutine `GI_Recv_Mat()` performs these tasks. The logical flags `recvth` and `recvn`, described in

Table 2 and Table 3 respectively, indicate which process sends the matrices. Since the Coordinate Storage Format is utilized for the matrices, two structures, each containing the matrix buffer dimension and the three arrays, are sent to the General Interface. Upon receipt, the matrix buffer dimension is retrieved first and used to unpack and store the matrix into the arrays `matval*, matrow*, matcol*`, which are described in Table 4. The following example depicts the receiving and unpacking of the matrices:

```
      IF (recvth) THEN
c Receive the structure from the thermal-hydraulic process.
        CALL pvmftrecv( tidth, mtypeth, timeout, 0, info)
      ENDIF
      IF (recvn) THEN
c Receive the structure from the neutronic process.
        CALL pvmftrecv( tidn, mtypen, timeout, 0, info)
      ENDIF

      istride = 1
c Unpack buffer dimensions for the permutation matrices.
      CALL pvmfunpack( INTEGER4, nmatth, 1, istride, info)
      CALL pvmfunpack( INTEGER4, nmatn, 1, istride, info)
c Allocate memory for the permutation matrices.
      ALLOCATE( matvalth(nmatth))
      ALLOCATE( matrowth(nmatth))
      ALLOCATE( matcolth(nmatth))
      ALLOCATE( matvaln(nmatn))
      ALLOCATE( matrown(nmatn))
      ALLOCATE( matcoln(nmatn))
c Unpack thermal-hydraulic/heat structure to neutronic matrix.
      CALL pvmfunpack( REAL8, matvalth, nmatth, istride, info)
      CALL pvmfunpack( INTEGER4, matrowth, nmatth, istride, info)
      CALL pvmfunpack( INTEGER4, matcolth, nmatth, istride, info)
c Unpack neutronic to thermal-hydraulic/heat structure matrix.
      CALL pvmfunpack( REAL8, matvaln, nmatn, istride, info)
      CALL pvmfunpack( INTEGER4, matrown, nmatn, istride, info)
      CALL pvmfunpack( INTEGER4, matcoln, nmatn, istride, info)
```

III.B. Thermal-Hydraulics to Neutronics Mapping

This functional unit, which is controlled by subroutine `GI_th2n()`, is composed of several tasks. The first step involves the receipt of time-dependent control information from the thermal-hydraulic process, which is managed by `GI_Recv_Bufth()`. This task is performed in much the same manner as that for the initial control information, in that the type-dependent buffers are sent as one structure and unpacked upon receipt. The difference here is in the information extracted from the control buffers, which is described below:

`lbufth(1)`: Indication of an error in the T/H code.
`lbufth(2)`: Indication of a data error in the T/H-Specific Data Map Routine.
`lbufth(3)`: Indication of a PVM error in the T/H-Specific Data Map Routine.

`lbufth(4)`: Indication of a data error in the General Interface.
`lbufth(5)`: Indication of a PVM error in the General Interface.
`lbufth(6)`: Indication of normal calculation termination.

Once the necessary information has been extracted, the next step is to receive the structure of unpermuted vector data from the thermal-hydraulic process. This structure includes the vector buffer dimension, along with the unpermuted vector described in Table 2. The following lines of code demonstrate the receipt of vector data:

```
c Receive unpermuted vector from the thermal-hydraulic process.
      CALL pvmftrecv( tidth, mtypeth, timeout, 0, info)
      istride = 1
c Unpack vector buffer dimension and allocate memory once (at the
c beginning) for the vector.
      CALL pvmfunpack( INTEGER4, nvecth, 1, istride, info)
      IF (.NOT.allocated(vecth)) ALLOCATE( vecth(nvecth))
c Unpack unpermuted vector of thermal-hydraulic/heat structure data.
      CALL pvmfunpack( REAL8, vecth, nvecth, istride, info)
```

This vector is then permuted using the thermal-hydraulic/heat structure to neutronic matrix (`mat***th`) obtained during the initialization. The coding necessary to perform the matrix-vector multiply using a matrix stored in Coordinate Storage Format is illustrated below:

```
c Allocate memory once (at the beginning) for the permuted vector
c and clear the buffer.
      nvecthp = MAXVAL(matrowth)
      IF (.NOT.allocated(vecthp)) ALLOCATE( vecthp(nvecthp))
      DO 10 i=1,nvecthp
        vecthp(i) = 0.0
  10  CONTINUE

c Perform MAT-VEC with matrix stored in Coordinate Storage Format.
      DO 20 i=1,nmatth
        vecthp( matrowth(i)) = vecthp( matrowth(i)) +
     &     matvalth(i)*vecth( matcolth(i))
  20  CONTINUE
```

Once the permutation is complete, the thermal-hydraulic control buffers are packed up into a single structure and sent to the neutronic process, as described in Section III.A. Again, these tasks are performed by `GI_Send_Bufth()`. The permuted vector of thermal-hydraulic data, along with the dimension of the vector, is then packed and sent to the neutronic process as demonstrated in the following example:

```
c Send the permuted vector to the neutronic process.
        istride = 1
        CALL pvmfinitsend( PVMDEFAULT, info)
c Pack vector buffer dimension.
        CALL pvmfpack( INTEGER4, nvecthp, 1, istride, info)
```

```
c Pack permuted vector of thermal-hydraulic/heat structure data.
        CALL pvmfpack( REAL8, vecthp, nvecthp, istride, info)
        CALL pvmfsend( tidn, mtypegi, info)
```

III.C. Neutronics to Thermal-Hydraulics Mapping

The tasks performed by this functional unit are controlled by subroutine `GI_n2th()`, and are similar to those described in Section III.B. The structure of control information sent from the neutronic process is received using subroutine `GI_Recv_Bufn()`, and the information shown below is extracted.

`lbufn(1)`:	Indication of an error in the Neutronic code.
`lbufn(2)`:	Indication of a data error in the Neutronic-Specific Data Map Routine.
`lbufn(3)`:	Indication of a PVM error in the Neutronic-Specific Data Map Routine.
`lbufn(4)`:	Indication of a data error in the General Interface.
`lbufn(5)`:	Indication of a PVM error in the General Interface.
`lbufn(6)`:	Indication of normal calculation termination.

Following this, the vector of data sent from the neutronic process is permuted using the neutronic to thermal-hydraulic/heat structure matrix (`mat***n`) obtained during initialization. The following lines of code demonstrate the necessary procedure:

```
c Receive unpermuted vector from the neutronic process.
        CALL pvmftrecv( tidn, mtypen, timeout, 0, info)
        istride = 1
c Unpack vector buffer dimension and allocate memory for the vector.
        CALL pvmfunpack( INTEGER4, nvecn, 1, istride, info)
        IF (.NOT.allocated(vecn)) ALLOCATE( vecn(nvecn))
c Unpack unpermuted vector of neutronic data.
        CALL pvmfunpack( REAL8, vecn, nvecn, istride, info)

c Allocate memory for the permuted vector and clear the buffer.
        nvecnp = MAXVAL(matrown)
        IF (.NOT.allocated(vecnp)) ALLOCATE( vecnp(nvecnp))
        DO 10 i=1,nvecnp
          vecnp(i) = 0.0
  10    CONTINUE

c Perform MAT-VEC with matrix stored in Coordinate Storage Format.
        DO 20 i=1,nmatn
          vecnp( matrown(i)) = vecnp( matrown(i)) +
     &       matvaln(i)*vecn( matcoln(i))
  20    CONTINUE
```

Once the permutation is complete, the neutronic control buffers are sent to the thermal-hydraulic process using subroutine `GI_Send_Bufn()`, as described in Section III.A. The per-

muted vector of neutronic data, along with the vector dimension, is then packed into a buffer to be sent to the thermal-hydraulic process.

```
c Send the permuted vector to the thermal-hydraulic process.
        CALL pvmfinitsend( PVMDEFAULT, info)
c Pack vector buffer dimension.
        CALL pvmfpack( INTEGER4, nvecnp, 1, istride, info)
c Pack permuted vector of neutronic data.
        CALL pvmfpack( REAL8, vecnp, nvecnp, istride, info)
        CALL pvmfsend( tidth, mtypegi, info)
```

III.D. Error Checking

The error checking module, `GI_Error_Check`, contains three subroutines, `GI_Data_Errchk()`, `GI_PVM_Errchk()`, and `GI_Proc_Errchk()`, which are called from each of the previously described functional units. The error checking operations performed by `GI_Data_Errchk()` are specific to each unit, and the call to `GI_Data_Errchk()` includes one argument, `errcode`, which is an integer relating to the specific error check to be performed (shown in parenthesis below). If an error is detected in this subroutine, the logical `errdatagi` (see Table 5 of Appendix A) is set to .TRUE., and the appropriate error message is displayed. In addition, the General Interface then communicates a "data" error, which indicates that the calculation should be terminated, to the thermal-hydraulic and neutronic processes using the space available in the logical control buffers.

Initialization:

(1) **[fatal error]** *Inconsistency in the indication of which process is sending the permutation matrices:*
result from: `recvth = recvn = .true.`
(or) `recvth = recvn = .false.`

(2) **[fatal error]** *Matrix elements outside prescribed range*:
result from: elements of `matvalth` or `matvaln` are outside the range specified in Table 4 of Appendix A.

(3) **[fatal error]** *Weighting factors in permutation matrix are inaccurate*:
result from: sum of row elements in thermal-hydraulic/heat structure to neutronic matrix do not sum to 1.0
(or) sum of column elements in neutronic to thermal-hydraulic/heat structure matrix do not sum to 1.0

Thermal-Hydraulics to Neutronics Mapping:

(4) **[fatal error]** *Inconsistency between matrix and vector dimensions*:
result from: `nvecth` ≠ `MAXVAL(matcolth)`

(or) `nvecth` ≠ `SIZE(vecth)` (This check is required because the memory for `vecth` is only allocated once at the beginning of the calculation. Thus, if the dimension, `nvecth`, is not consistent with the size of the previously allocated vector, the unpacking of the vector will be in error.)

(5) **[fatal error]** *Negative elements in unpermuted thermal-hydraulic vector*:
result from: `vecth` < `0`

Neutronics to Thermal-Hydraulics Mapping:

(6) **[fatal error]** *Inconsistency between matrix and vector dimensions*:
result from: `nvecn` ≠ `MAXVAL(matcoln)`
(or) `nvecn` ≠ `SIZE(vecn)` (See note above.)

(7) **[fatal error]** *Negative elements in unpermuted neutronic vector*:
result from: `vecn` < `0`

`GI_PVM_Errchk()` detects errors in the PVM calls, and includes three arguments, `errcode`, `iunit`, and `istat`. `errcode` refers to the PVM function which returned an error, `iunit` refers to where `GI_PVM_Errchk()` is being called from, and `istat` is the integer PVM status code which indicates an error if its value is less than zero. An association table is shown below for `errcode` and `iunit`. If an error is detected in this subroutine, the logical `errpvmgi` (see Table 5 of Appendix A) is set to .TRUE., and the appropriate error message is displayed. The General Interface then attempts to communicate a "PVM" error to the thermal-hydraulic and neutronic processes using the space available in the logical control buffers. Specifically, the General Interface only allocates space for five words in the logical control buffers if a PVM error is detected, where the fifth word is used to communicate `errpvmgi`. It should be noted that an error resulting from a PVM call may not allow for safe termination of all processes.

`errcode`:	associated with:
1	pvmfinitsend
2	pvmfpack
3	pvmfsend
4	pvmfbcast
5	pvmftrecv
6	pvmfunpack

`iunit`:	associated with:
1	PVM operations required in `GI_Obtain_IDs()`
2	PVM operations required in `GI_Recv_Bufth()`

iunit:	associated with:
3	PVM operations required in GI_Send_Bufth()
4	PVM operations required in GI_Recv_Bufn()
5	PVM operations required in GI_Send_Bufn()
6	PVM operations required in GI_Recv_Mat()
7	PVM operations required in GI_th2n() to receive unpermuted thermal-hydraulic vector
8	PVM operations required in GI_th2n() to send permuted thermal-hydraulic vector
9	PVM operations required in GI_n2th() to receive unpermuted neutronic vector
10	PVM operations required in GI_n2th() to send permuted neutronic vector

Finally, GI_Proc_Errchk() performs checks on the value of the error logicals sent from the thermal-hydraulics and neutronics codes. This subroutine requires just one argument, errcode, which can have only one of two values (1 or 2). If errcode is one, the value of the three error logicals sent from the neutronics code: errcalcn, errdatan, and errpvmn (see Table 5 of Appendix A) is checked. If errcode is two, the value of the three error logicals sent from the thermal-hydraulics code: errcalcth, errdatath, and errpvmth (see Table 5 of Appendix A) is checked. The detection of any error from either the thermal-hydraulics or the neutronics code results in an error message being displayed indicating the origin of the error. The control buffers containing the error logicals are then forwarded to the appropriate process to communicate the detected error.

IV. Process Flow and I/O

The flow through the three functional units and one error checking unit described in the previous section proceeds as shown in Figure 1. The entry points into the initialization and mapping units of the General Interface are controlled with subroutine calls, and the communication of data to and from these subroutines is controlled with the use of PVM *sends* and *receives*. Entry to the error checking module is also controlled with subroutine calls.

There is no explicit input processing performed by the General Interface. However, input to this process is treated with the use of the PVM *receive* protocol, where the information received is as described in the previous section. In addition, output processing is separated into two classes: calculational and error trapping. Calculational output refers to both the control and vector information which is transferred to the appropriate process with the PVM *send* protocol. Error trapping output is composed of fault messages which are dumped to either the screen or the disk, and is controlled by the error checking unit.

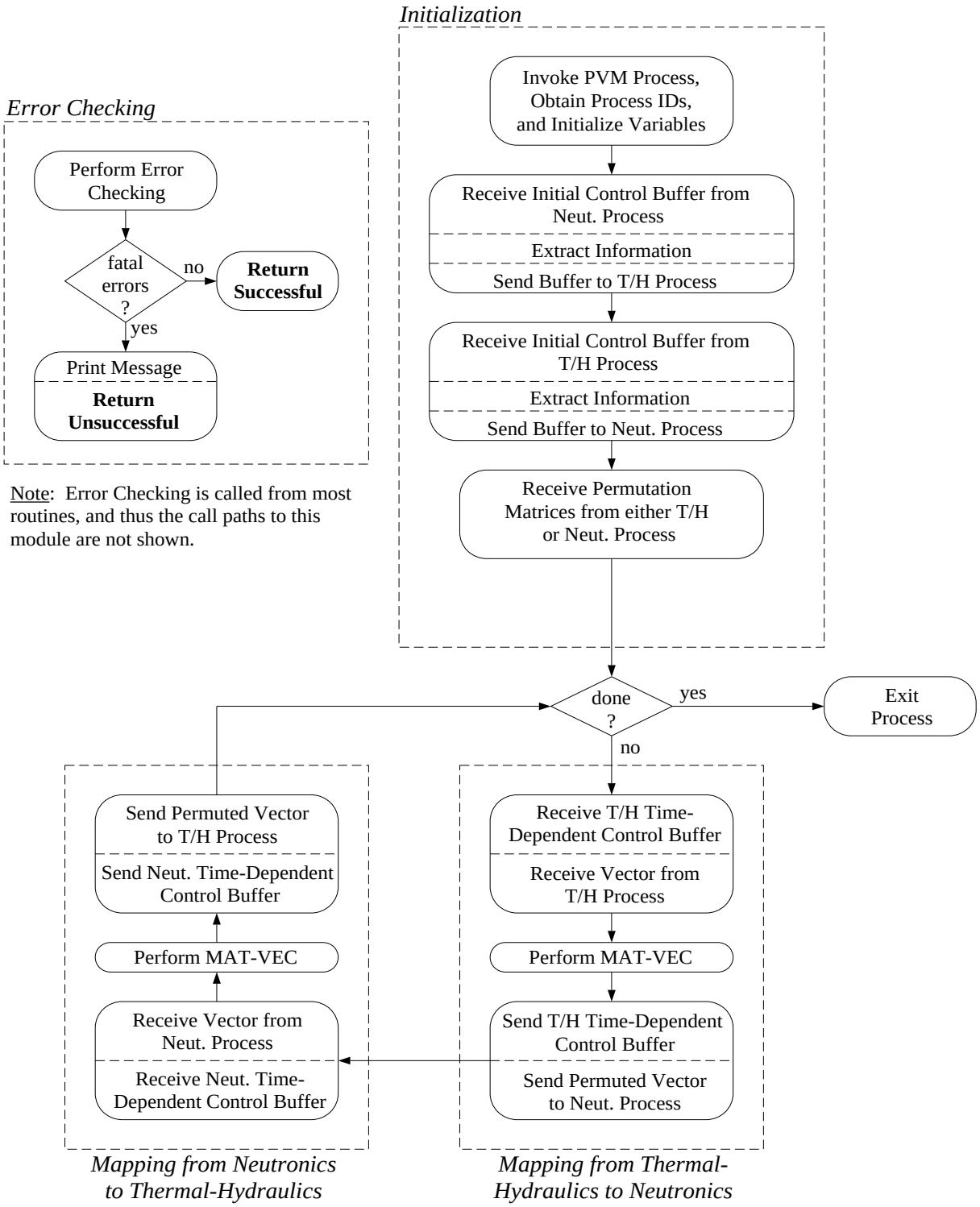


Figure 1: Calculational Flow for the General Interface Process

Exit handling in the General Interface process is invoked upon receipt of either a calculation- or fault-based signal. A fault-based signal results from a fatal error detected in the error checking module, as discussed in the previous section, and requires that all of the processes exit prematurely. The General Interface communicates the error to both the thermal-hydraulic and neutronic processes using the next available control buffer send. Once the error has been communicated to both processes, the General Interface initiates its exit procedure, which is described below.

A calculation-based signal (`done`) is controlled by the thermal-hydraulic and neutronic processes and is transferred to the General Interface through the control buffers. As an example, the thermal-hydraulic process sends a flag indicating that the current calculation should be the last. The interface then performs its calculation and transfers the required information to the neutronic process. At this point, the neutronic process has the ability to reject the thermal-hydraulic data, and request a new time step calculation, thus resetting the flag sent from the thermal-hydraulic process. The neutronic process then sends this flag to the interface, indicating that the calculation is not complete. This procedure, which can be inferred from Figure 1, will continue until both the thermal-hydraulic and neutronic processes agree on the condition of the flag.

Once termination of the General Interface process has been determined, the subroutines `GI_Clean()` and `GI_Exit()` are called. `GI_Clean()` frees up any memory not previously deallocated, and `GI_Exit()` removes the General Interface process from PVM.

The following lines of code depict the overall flow control for the General Interface:

```
      CALL GI_Init()
      DO 10 WHILE (.NOT. done)
        CALL GI_th2n()
        CALL GI_n2th()
 10   CONTINUE
      CALL GI_Clean()
      CALL GI_Exit()
```

V. Summary

This document described the software design necessary to satisfy the requirement specifications for the General Interface. The major components outlined here provide the basis for code development and relate to the variable requirements, the design of the functional units, and the process flow through the interface, which includes the handling of input/output requirements.

The General Interface is designed as an independent process, and the incorporation within the framework of the coupled thermal-hydraulics and neutronics code requires both a process control and message-passing protocol. The PVM package meets these requirements by both managing the interface task and controlling the communication between the separate processes. The

internal flow control through the functional units of the General Interface is treated logically, as described in the previous sections.

The computational requirements for the General Interface consist only of two matrix-vector multiplies, which serve to map property data between the thermal-hydraulic and neutronic code. The two mapping units described in the software design accomplish this task using the matrices and vectors sent from both the thermal-hydraulic and neutronic processes. This computational simplicity in the interface design is the basis for providing the necessary flexibility in coupling any thermal-hydraulic code with any neutronic code.

VI. References

1. D. Barber and T. Downar, “Software Requirements Specification for the General Interface in the Coupled Code,” Technical Report, PU/NE-98-8, Purdue University, (1998).

2. Y. Saad, “Numerical Methods for Large Eigenvalue Problems,” Manchester University Press, Manchester, UK., pp. 40-41 (1992).

Appendix A: Variable Description

The following tables provide a description of the variables used by the General Interface routines:

Table 1: PVM and Process Control Variables

Name	Type	Dimension	Description	Range
done	logical	1	logical flag: execution is stopped when done=.true.	.true. / .false.
ioutp	int*4	1	unit number for output	$0 < i < 100$
ntasks	int*4	1	number of processes	$i = 3$
group	char*6	1	6-character descriptor for the group of processes	“procs”
tidgi	int*4	1	process ID for the General Interface module	$0 < i < 2^{32}$
tidth	int*4	1	process ID for the Thermal-Hydraulic module	$0 < i < 2^{32}$
tidn	int*4	1	process ID for the Neutronic module	$0 < i < 2^{32}$
mtypegi	int*4	1	message tag associated with the General Interface module	$i = 1$
mtypeth	int*4	1	message tag associated with the Thermal-Hydraulic module	$i = 2$
mtypen	int*4	1	message tag associated with the Neutronic module	$i = 3$
inum	int*4	1	instance number in group	$0 \le i \le 2$
istride	int*4	1	striding of data in the buffer	$0 < i < 2^{32}$
info[(a)]	int*4	1	integer error flag for PVM calls	$-(2^{32}) < i < 2^{32}$
timeout	real*4	1	int*4 parameter indicating the max # of seconds to wait on a receive	$i = 600$
dimbuf	int*4	6	dimension of each of the 6 data type-dependent control buffers, in order, as shown in Tables 2 and 3	$0 \le i < 2^{32}$

[(a)] Values less than zero indicate an error has occurred in a PVM call.

Table 2: Thermal-Hydraulic Data and Control Buffers

Name	Type	Dimension	Description	Range
cbufth[(a)]	char*6	dimbuf(1)	control buffer of 6-character words	N/A
lbufth[(a)]	logical	dimbuf(2)	control buffer of logical words	.true. / .false.
i2bufth[(a)]	int*2	dimbuf(3)	control buffer of 16 bit integer words	$-(2^{16}) < i < 2^{16}$
i4bufth[(a)]	int*4	dimbuf(4)	control buffer of 32 bit integer words	$-(2^{32}) < i < 2^{32}$
r4bufth[(a)]	real*4	dimbuf(5)	control buffer of 32 bit floating point words	$-(10^{38}) < x < 10^{38}$
r8bufth[(a)]	real*8	dimbuf(6)	control buffer of 64 bit floating point words	$-(10^{308}) < x < 10^{308}$
recvth	logical	1	a value of .true. indicates that both permutation matrices are sent from the thermal-hydraulic process	.true. / .false.
nvecth	int*4	1	dimension of vecth received from thermal-hydraulic process	$0 < i < 2^{32}$
vecth	real*8	nvecth	vector of space-dependent thermal-hydraulic and heat structure data	$0 \le x < 10^{308}$
nvecthp	int*4	1	dimension of vecthp sent to neutronic process	$0 < i < 2^{32}$
vecthp	real*8	nvecthp	permuted vector of space-dependent thermal-hydraulic and heat structure data	$0 \le x < 10^{308}$

[(a)] Used for both initial and time-dependent control information

Table 3: Neutronic Data and Control Buffers

Name	Type	Dimension	Description	Range
cbufn[(a)]	char*6	dimbuf(1)	control buffer of 6-character words	N/A
lbufn[(a)]	logical	dimbuf(2)	control buffer of logical words	.true. / .false.
i2bufn[(a)]	int*2	dimbuf(3)	control buffer of 16 bit integer words	$-(2^{16}) < i < 2^{16}$
i4bufn[(a)]	int*4	dimbuf(4)	control buffer of 32 bit integer words	$-(2^{32}) < i < 2^{32}$
r4bufn[(a)]	real*4	dimbuf(5)	control buffer of 32 bit floating point words	$-(10^{38}) < x < 10^{38}$
r8bufn[(a)]	real*8	dimbuf(6)	control buffer of 64 bit floating point words	$-(10^{308}) < x < 10^{308}$
recvn	logical	1	a value of .true. indicates that both permutation matrices are sent from the neutronic process	.true. / .false.
nvecn	int*4	1	dimension of vecn received from neutronic process	$0 < i < 2^{32}$
vecn	real*8	nvecn	vector of space-dependent neutronic data	$0 \le x < 10^{308}$
nvecnp	int*4	1	dimension of vecnp sent to thermal-hydraulic process	$0 < i < 2^{32}$
vecnp	real*8	nvecnp	permuted vector of space-dependent neutronic data	$0 \le x < 10^{308}$

[(a)] Used for both initial and time-dependent control information

Table 4: Variables Associated with the Permutation Matrices

Name	Type	Dimension	Description	Range
nmatth	int*4	1	dimension of the arrays describing the permutation matrices: matvalth, matrowth, and matcolth.	$1 \le i \le n$; $n = nvecthp*nvecth$
matvalth	real*8	nmatth	non-zero elements in the permutation matrix used to map thermal-hydraulic and heat structure data to neutronic nodes.	$0 < x \le 1$
matrowth	int*4	nmatth	row number corresponding to each element in matvalth	$1 \le i \le nvecthp$
matcolth	int*4	nmatth	column number corresponding to each element in matvalth	$1 \le j \le nvecth$
nmatn	int*4	1	dimension of the arrays describing the permutation matrices: matvaln, matrown, matcoln.	$1 \le i \le n$; $n = nvecnp*nvecn$
matvaln	real*8	nmatn	non-zero elements in the permutation matrix used to map neutronic data to thermal-hydraulic zones and heat structure components.	$0 < x \le 1$
matrown	int*4	nmatn	row number corresponding to each element in matvaln	$1 \le i \le nvecnp$
matcoln	int*4	nmatn	column number corresponding to each element in matvaln	$1 \le j \le nvecn$

Table 5: Variables Associated with the Error Checking

Name	Type	Dimension	Description	Range
errdatagi	logical	1	indication of a data error in the General Interface	.true. / .false.
errpvmgi	logical	1	indication of a PVM error in the General Interface	.true. / .false.
errinit	logical	1	indication of General Interface error (errdatagi .or. errpvmgi) in the Initialization functional unit	.true. / .false.
errth2n	logical	1	indication of General Interface error (errdatagi .or. errpvmgi) in the Thermal-Hydraulic to Neutronic unit	.true. / .false.
errn2th	logical	1	indication of General Interface error (errdatagi .or. errpvmgi) in the Neutronic to Thermal-Hydraulic unit	.true. / .false.
errcalcth	logical	1	indication of a calculation error in the Thermal-Hydraulics code	.true. / .false.
errdatath	logical	1	indication of a data error in the T/H-specific data map routine	.true. / .false.
errpvmth	logical	1	indication of a PVM error in the T/H-specific data map routine	.true. / .false.
errth	logical	1	indication of error on T/H side (errcalcth .or. errdatath .or. errpvmth)	.true. / .false.
errcalcn	logical	1	indication of a calculation error in the Neutronics code	.true. / .false.
errdatan	logical	1	indication of a data error in the Neutronic-specific data map routine	.true. / .false.

Table 5: Variables Associated with the Error Checking

errpvmn	logical	1	indication of a PVM error in the Neutronic-specific data map routine	.true. / .false.
errn	logical	1	indication of error on Neutronics side (errcalcn .or. errdatan .or. errpvmn)	.true. / .false.

Appendix B: Subroutine Report

Subroutines Deleted:

none

Subroutines Added:

Main GI: Driver for the General Interface.
Uses Modules: GI_Var_Decl, GI_Time_Calc

Contains Subroutines:
Subroutine GI_Init(): Controls the Initialization functional unit.
Uses Module: GI_Init_Calc

Subroutine GI_Clean(): Frees up memory for arrays not previously de-allocated.

Subroutine GI_Exit(): Removes General Interface process from PVM.

Module GI_Init_Calc: Contains the subroutines used during the initialization stage.
Uses Module: GI_Var_Decl, GI_Error_Check

Contains Subroutines:
Subroutine GI_Obtain_IDs(): Establishes communication with the thermal-hydraulic and neutronic processes.

Subroutine GI_Buf_Init(): Communicates initial control buffers between the thermal-hydraulic and neutronic processes.
Uses Module: GI_Comm_Buf

Subroutine GI_Recv_Mat(): Receives the permutation matrices from either the thermal-hydraulic or neutronic process.

Module GI_Time_Calc: Contains the subroutines used for the time-dependent data mapping.
Uses Module: GI_Var_Decl, GI_Error_Check

Contains Subroutines:

Subroutine GI_th2n(): Communicates thermal-hydraulic control buffer to neutronic process and maps thermal-hydraulic data to the neutronic problem domain.
Uses Module: GI_Comm_Buf

Subroutine GI_n2th(): Communicates neutronic control buffer to thermal-hydraulic process and maps neutronic data to the thermal-hydraulic problem domain.
Uses Module: GI_Comm_Buf

Module GI_Comm_Buf: Contains the subroutines used to communicate control buffers between the thermal-hydraulic and neutronic processes.
Uses Module: GI_Var_Decl, GI_Error_Check

Contains Subroutines:

Subroutine GI_Recv_Bufth(): Receive thermal-hydraulic control buffer.

Subroutine GI_Send_Bufth(): Send thermal-hydraulic control buffer to neutronic process.

Subroutine GI_Recv_Bufn(): Receive neutronic control buffer.

Subroutine GI_Send_Bufn(): Send neutronic control buffer to thermal-hydraulic process.

Module GI_Error_Check: Contains the subroutines used to perform the error checking for the General Interface.
Uses Module: GI_Var_Decl

Contains Subroutines:

Subroutine GI_Data_Errchk(): Perform error checking on data used by General Interface.

Subroutine GI_PVM_Errchk(): Perform error checking on PVM status variables.

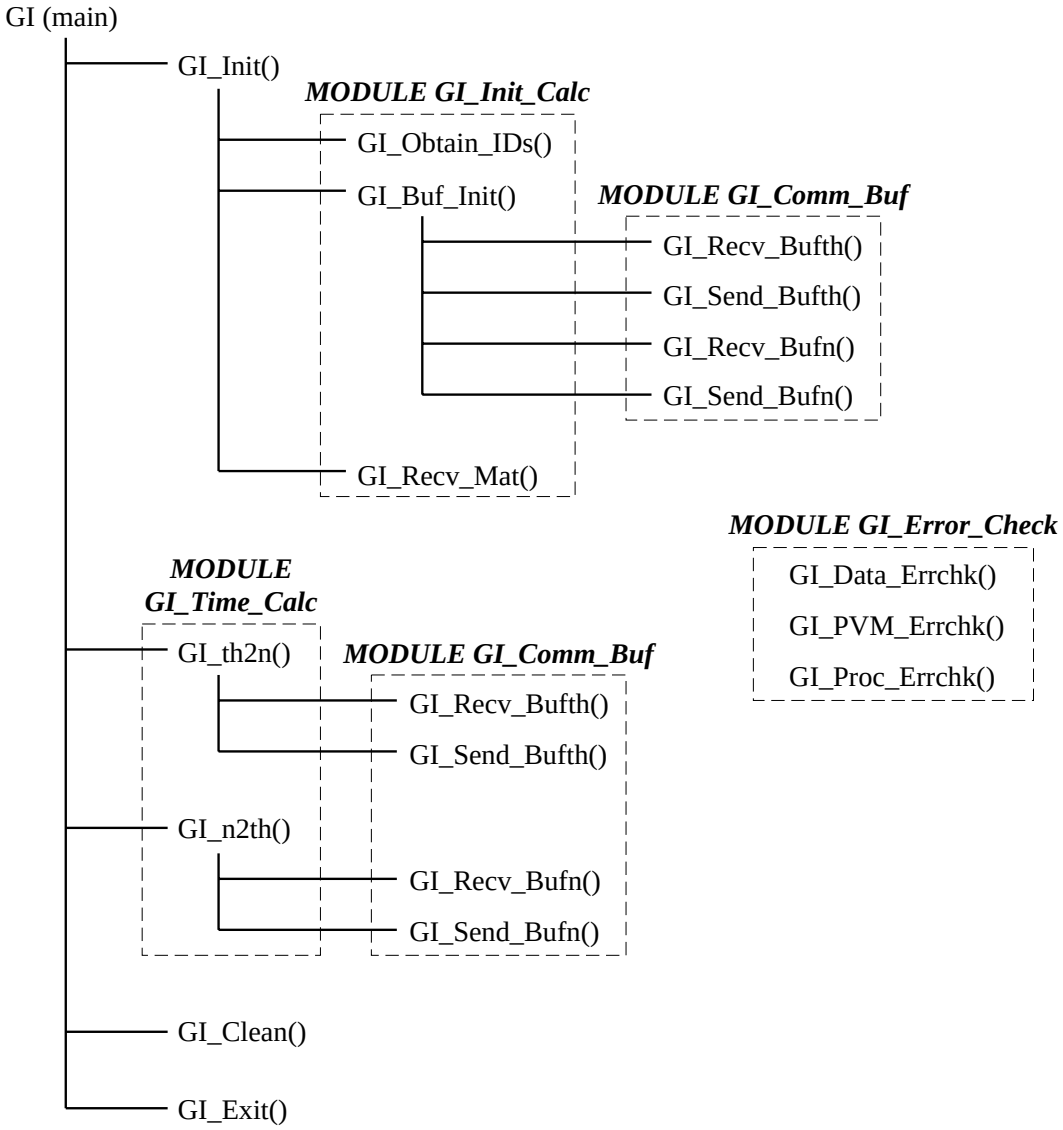
Subroutine GI_Proc_Errchk(): Check the value of error logicals sent from the thermal-hydraulic and neutronic codes.

Module GI_Var_Decl: Declares the variables used by the General Interface routines.

Subroutines Modified:

none

Subroutine Calling Tree:



Note: The subroutines contained in module `GI_Error_Check` are called from most subroutines, and thus, the paths of entry into these error checking subroutines are not shown.