Galileo "Message Generation Facility" – Safety-critical and Real-time

Antonio LATORRE[(1)], Adrián MORA[(1)], Pedro PALOMO[(1)], Tomás SUAREZ[(1)], Mike RENNIE [(1)]

[(1)] DEIMOS Space, Ronda de Poniente, 19, Edificio FiteniVI, 2-2, 28760 Tres Cantos, Madrid, Spain.
Email: Antonio.Latorre@deimos-space.com

ABSTRACT

1. CONTEXT OF THE MGF

Galileo will be an independent, global European-controlled, satellite-based navigation system. It will have a constellation of satellites monitored and controlled by a Ground Control Segment (GCS) providing also the capability to detect satellite or system malfunctions and broadcast real-time warnings (integrity messages).

GALILEO is a programme sponsored by the European Space Agency and the European Union.

The overall Galileo System is divided into two main segments:

- The Galileo Space Segment (SS) will comprise a constellation of 36 satellites in MEO. Each satellite will broadcast four ranging signals carrying clock synchronisation, ephemeris, integrity and other data, depending on the particular signal. A user equipped with a suitable receiver will be able to determine his position to within a few metres when receiving signals from visible Galileo satellites.
- The Galileo Ground Segment (GS) will control the whole Galileo constellation, monitor the satellite health and up-load data for subsequent broadcast to users. The key elements of this data such as clock synchronisation, ephemeris and integrity, will be calculated from measurements made by a network of Galileo receiving stations. The GS is split into:
  - The Ground Control Segment (GCS) in charge of monitoring and control of the Galileo constellation.
  - The Ground Mission Segment (GMS) in charge of the determination and dissemination of the navigation and integrity data and of the external components data (ERIS, SAR, CS, etc.). It is decomposed into several "elements", one of which is the Message Generation Facility (MGF).

The MGF is developed by an industrial consortium headed by DEIMOS Space, as "N-2" element prime. The GMS itself is the responsibility of TAS (Toulouse) as "N-1" segment prime.

2. OBJECTIVES OF THE MGF

The Message Generation Facility is one of several elements, along with the OSPF, IPF, Key Management facilities and MUCF, that comprise the "processing chain" of the GMS.

The MGF is in charge of multiplexing and routing navigation/integrity data to be sent for mission uplink. It also performs multiplexing of Search & Rescue (SAR), Commercial Service (CS) and external regional integrity data (ERIS). It is the meeting point for all the data streams that make up the C-Band Uplink Messages.

Its mission is to elaborate messages as specified in the GMS to Space ICD, by multiplexing those data that allow providing the Open Service (OS), CS, Safety of Life (SoL), Public Regulated Service (PRS), SAR, and ERIS services. The multiplexed streams are then forwarded to the Uplink Stations (ULS)/antennas according to pre-defined routing tables constrained by the ULS antennas tracking plan and by needs specific to each service.

Messages to be provided to the GCS (for the Degraded Navigation Service, through S-Band) are also prepared by the MGF. In particular, selection processes are implemented to handle the various redundancies of input streams. Also, the MGF has to manage the specific feature of the integrity messages in the Galileo SIS by building messages with integrity Tables (periodically) and with related Alarms (continuously) on the basis of input integrity data.

The handling of Integrity data is time and safety critical, and the MGF implements a Safety-Monitor (or "Safety Box") capable of

identifying, and isolating, failures in the processing. Safety Monitor is the main barrier included in the architecture to avoid integrity data corruption (Critical Event), and it does verifications on alert generation, HIT generation, Region Status, Alert counter management, AIT update, application CRC verification, IPF input data selection and ERIS data.

In summary, the MGF must ensure the following real-time processing functions, among others:

- Navigation, Integrity, ERIS, SAR, etc. data acquisition and processing
- Time/Safety-Critical Integrity alarms elaboration for SoL and PRS services. This data is composed of the safety-critical signal status alarms that indicate to the final users if the positioning data sent by a given satellite can be trusted.

  The integrity alarms produced by the GMS have to be sent to the final users in a limited time (TTA – Time To Alarm). Each element in the GMS has been assigned a particular portion of the TTA.
- Generation, at 1Hz, of dedicated message sub-frames for each Galileo satellite, containing:
  - OS navigation data in order to provide the satellites with orbit determination data, satellites clock correction data with respect to Galileo System Time (GST), Signal-in-Space Accuracy (SISA) data, ionospheric products and other services
  - SoL navigation data, providing the same products as above
  - SAR data for given satellites and SAR beacons
  - Encrypted CS data for transmission to specific regions
  - Encrypted PRS integrity and navigation data, including integrity alarms for PRS, status of the Galileo constellation, SISMA data for the constellation, and a complete set of PRS navigation data
- Generation, at 1Hz, of dedicated integrity message for each Galileo satellite, containing integrity data for the SoL service. This data includes:
  - Integrity alarms for SoL
  - Status of the whole Galileo constellation
  - Hard Integrity Tables (HIT) containing the predicted SISMA data for the whole constellation
  - External Service Providers (ERIS) integrity data, dedicated to specific regions
- Generation of the Quality of Service (QoS) indicators to be used by the GMS to select the best messages to up-link
- Generation of the corresponding ground assets technical monitoring data
- Generation of mission monitoring data

3. ELEMENT INTERFACES

The figure below shows a basic representation of the external interfaces of the MGF Element, and includes the associated external data flows between external elements (inside and outside the GMS) and MGF.

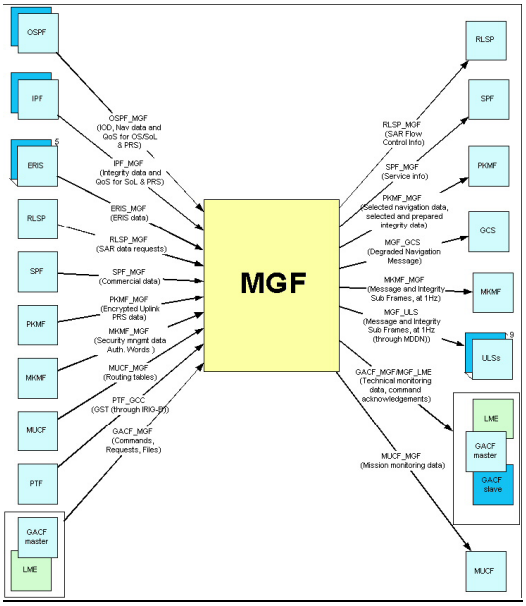


Figure 1. MGF Interfaces

The physical interfaces between the MGF and other elements are as follows:

- The GCC **Real-time LAN** (RT): Using the **UDP/IP** protocol.
- The GCC **Near real-time LAN** (NRT):
  - Using the **TCP/IP** protocol for non-real time data transfers.
  - Using the **FTP** protocol for transferring routing tables, configuration files and local storage data migration to the global archive.
  - Using the **SNMP** protocol for GACF / LME control and monitoring transfers.

- **xKMF Security Ports**: Dedicated Ethernet lines, using UDP/IP for real-time transfers.
- The **PTF time dissemination network** (IRIG-B): The time signal sent by the PTF facility is received in the MGF by a time acquisition card.
- **Local Maintenance Interface** (LMI): This is a local network port used for element maintenance and test.
- **Test ports**: Used in all modes except "Off" to send potentially – for test and observation purposes – all MGF products.
- **Front Panel LEDs**: give a visible indication of the current mode and status of the MGF.

4. ARCHITECTURE

Fig 2. illustrates the physical architecture of the MGF.

It comprises three processing boards, two of which have a SATA Hard Disk for file storage, and a VMEbus backplane that provides the means to implement inter-board communications (as well as power distribution). Two of the boards implement the external interfaces. All boards have a flash disk to support the boot process. The hardware has been integrated in a VME rack by Thales Computers.

One board contains two software partitions (CPU time and memory space partitioning), allowing the DAL-B Safety Monitor to be hosted in the same board as the DAL-C frame generation software that it protects. The underlying real-time operating system is a version of LynxOS-178B, from LynuxWorks, tailored for use in Galileo.

The software architecture has been decomposed at the highest level into three "components": Message Processing & Uplink Frame Generation (MPUFG), Monitoring & Control (M&C) and External Interfaces (EIF). MPUFG (which includes the Safety Box) is contained entirely within Board 1, while the other two components, M&C and EIF are present in all three boards. The relevant components are compiled and linked to produce the application software executables that run on each board/partition.

In addition, dedicated "loader" programmes have to be developed per board to support the boot process, and these are considered part of the M&C.

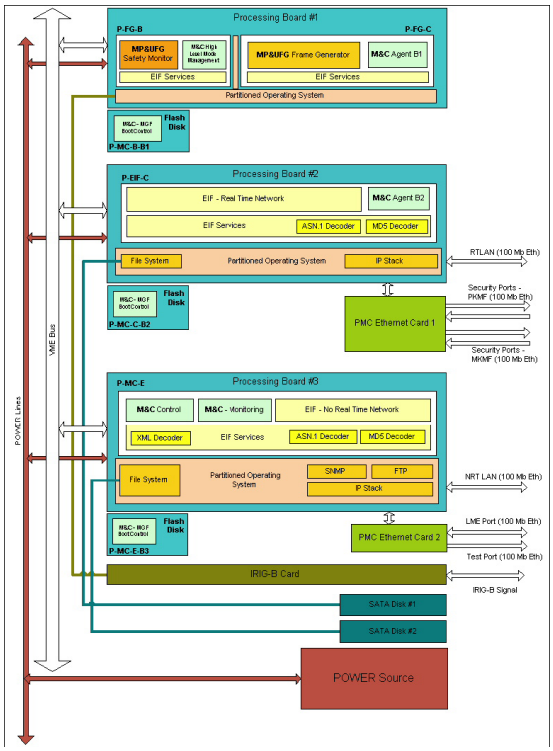


Figure 2. MGF Interfaces

The software architecture has been decomposed into around 280 terminal HOOD objects (not including OP_Control objects) across all three SW components, resulting in individual objects that are a manageable size and where each object is traced to a reasonably small number of requirements.

5. ANALYSIS AND DESIGN

The MGF element functional decomposition uses IDEF0 notation, allowing it to be presented in an ordered and hierarchical way. The principal functions are decomposed into sub-functions, the incoming and outgoing data flows, control commands, resources, etc.

At software-level, in the Software System Specification, the technique of "Functional chains" has been applied.

The software life-cycle caters for DALs "B" (Safety Box, including "agents" of M&C and EIF), "C" (the rest of MPUFG, a large part of EIF and another M&C agent) and "E" (a large part of M&C and another EIF agent). As can be seen, different pieces of each SW component have to be developed to different DALs.

The software architectural design has been performed using the Hard Real-Time Hieracrchical Object-Oriented Design (HRT-HOOD) methodology (supported by the "Stood" tool from Ellidiss).

The requirements are managed in a DOORS database, for easy exchange with the GMS prime, but in addition, the traceability between SRDs and the design model in Stood is supported by the "Reqtify" tool from TNI, which acts as a bridge between DOORS and Stood.

6. DEPENDABILITY AND SAFETY

As the MGF is a safety-critical element in the real-time processing chain in the GMS, a large effort has been made on the RAMS activities. Initial Fault Tree Analysis (FTA) was performed in order to allocate the SW DAL (i.e. demonstrate that the barriers defined in the design allow the DAL to be reduced). In parallel, initial hazard analysis (HA) was performed at the very beginning of the design (i.e. at functional level). Hazards identified at this level (e.g. possible Single Point Failures) were mitigated by injecting recommendations into the design.

Once the SW/HW was defined (taking into account the results of the initial RAMS activities) a complete set of RAMS analyses were performed: HW analysis (Reliability and Maintainability), FMECA, FTA (both qualitative and quantitative), HA for "personal safety", HSIA, CM&CCA and COA. All these analyses generate recommendations that are then injected into the design, or design process, in order to mitigate the identified critical items. Exhaustive follow-up of those recommendations is done in order to assure their correct implementation.

7. DIFFICULTIES AND PROBLEMS THAT HAD TO BE OVERCOME

7.1. Outline

The remainder of this paper addresses the main technical issues that presented some difficulties and interesting problems that had to be solved in the specification and design of the MGF. Briefly, they can be summarised as follows:

- At first, the MGF was expected to be implemented on a single board (single processor). However, for several reasons, the architecture is now the three-board solution presented above.

  The reasons include the evolving requirements regarding physical interfaces to the external elements, feedback from RAMS analyses (DAL-reduction, safety barriers and critical items) the need to separate the "soft" real-time processing (of asynchronous external inputs) from the truly "hard" real-time processing (associated mainly with message processing and frame generation), and even the raw processing power (with margin!) that it requires (particularly in the time-critical segment of the processing that contributes to the Time-To-Alarm).

  The three-board design solved the problems inherent in the original one-board design, but created other problems of its own, that required creative and novel solutions.

- Time synchronisation across three boards, using the VME to distribute the time received by a single IRIG-B on one of the boards, presented quite a few headaches.

  Careful time synchronisation is essential to ensure that the VMEbus is shared by all three boards correctly (they all refer to the same time schedule for exchanging messages via the bus).

- The automatic boot procedure for the three boards required a considerable amount of thought and effort to develop.

  It is only partially supported by the operating system on each board. The application software (including the "loaders") have to manage and control the boot and initialisation of the three boards, in a fully co-ordinated manner, and including the commanded (from the GACF) mode changes.

These problems are explored in more detail below:

7.2. Three Processing Boards

RAMS analyses at segment (GMS) level determined that the MGF should be assigned software DAL-B. However, as a result of the element-level RAMS process, different software components or functions have been given different DALs. A condition for this is that the software parts with different DALs must be separated from one another such that runtime errors are prevented from being propagated from lower to higher DAL software.

Our initial approach was based on using a DO-178B ("ARINC653-like") compliant operating system that supports time (CPU) and space (memory) partitioning, and all of the software executing on the same board.

The MGF has many external interfaces to the rest of the GMS. It is connected to: the real-time (RT) LAN for receiving and sending data at 1Hz (e.g. input integrity data from the IPFs, output uplink sub-frames to the ULSes); the near real-time (NRT) LAN for exchanges at a

lower frequency, including the command and monitoring interface with the GACF; and the dedicated Ethernet interfaces for exchanging messages with the security elements (xKMF).

A single board could not support the required number of Ethernet ports, and that is one reason for extending the design to several boards.

Another important reason is that the processing of received UDP/IP and TCP/IP messages, performed in the RTOS, is done in periods of time that cannot be controlled or guaranteed by the application software. That cannot be tolerated in a safety-critical hard real-time system. The sequence of functions that generate the uplink sub-frames form a kind of "critical path", where the CPU margins (between the deadline for one step and the latest start of the next step) are as short as a few milliseconds. These margins could be consumed if extraneous activity, namely the RTOS handling messages arriving from the LANs, were allowed to pre-empt the application just at the worst moment. That would make it impossible to guarantee the MGF's critical TTA-related deadlines for the generation of the uplink sub-frames. Therefore, the processing of the network traffic, both input and output, has been put onto boards 2 (RT) and 3 (NRT), leaving board 1 to perform uplink the message generation unmolested by sporadic interruptions coming from the external interfaces.

The 3-board architecture solved the problem of how to support the many external interfaces of the MGF, but it raised another problem, namely the internal interfaces. Input data to be used in the production of the uplink sub-frames arrives in boards 2 and 3, but the sub-frames themselves are built in board 1. Those sub-frames are sent to the mission data distribution network (MDDN) via the RT LAN through board 2. This implies that a lot of data has to be moved about between boards, in real-time.

The first part of the solution was to use the VMEbus to exchange data between boards. However, that solution comes at a cost. We still have to guarantee those critical deadlines for the uplink sub-frame generation in board 1, so the VME traffic has to be carefully controlled, such that it is deliberately scheduled to occur at allotted times. That means the times for the VME exchanges have to be known by design, and have to fit into the overall schedule of all the activities that occur during each 1-second cycle. In particular, there is a certain span of time (around 100ms) leading up to the critical deadline for sending the generated sub-frames from board 1 to board 2, where the margins are very short, and almost no other VME traffic can be done in parallel (VME traffic is not instantaneous, it implies a CPU load on the origin and destination boards because of DMA transfers robbing CPU cycles).

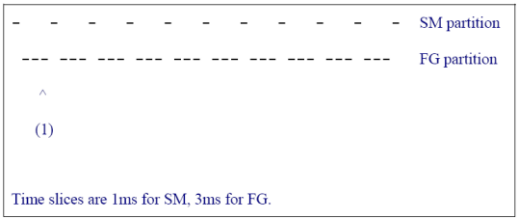
So, the next part of the solution was to organise design an application-level "scheduler" that initiates VME transfers according to a pre-defined, fixed plan. That plan has to accommodate the worst-case traffic scenario, and include margins (for error, or for minimising the impact of future changes to the scenario).

The design of the VME bus access plan had to cope with some other problems:

- Partition time-slicing in Board 1

  The time-slicing between the two partitions in B1 introduces jitter in the initiation of VME transfers, because when one partition is programmed to start a transfer at time *t*, the other partition might be scheduled by the RTOS, so the first partition might not actually be able to start its transfer until it gets the CPU again, at the next time slice, i.e .at *t+(time slice)*. Furthermore, if a transfer takes longer than the time slice of the initiating partition, the transfer will be suspended while the other partition is scheduled, resuming when the initiator gets its next time slice.

  This is illustrated below:



  FG at (1), which is during the 3rd ms of its 1st time slice, initiates a transfer with a CPU execution time of 7ms. During that time, SM will be scheduled three times (for a total of 3ms), so we say that the time-slice overhead is 3ms.

- Ensure that the processes contributing to the TTA have exclusive access to the VME when they need it.

  Other VME traffic that has nothing to do with TTA is allocated VME time slots before the TTA-processing cycle or after the last TTA deadline, so that they don't interfere with the TTA.

The fixed "VME bus access plan" is implemented in all three boards by a high-priority cyclic task executing at 500Hz, i.e. the schedule is composed of 500 slots of 2ms each.

The same, common schedule is used in all three boards. When a transfer needs more than one slot, then consecutive (contiguous) slots are allocated in the schedule.

In most cases, the entire transfer of data can be performed in one or more VME slots in the same 1-second cycle. However, in a few cases, where a large amount of data has to be transferred, the approach is to spread the transfer over several 1-second cycles. In each cycle, a reasonable amount of data is transferred. The duration of this type of transfer is defined with the aim of making good use of the VME (by transferring data in large chunks, so that the transfer of the entire data does not take an unnecessarily long amount of time).

The VME bus schedule is synchronised with the GST epoch, i.e. with the 1-second time signal coming from the IRIG-B. It therefore relies on that time synchronisation between boards to ensure that the VME bus usage is synchronised. The schedule

says when to read, when to write, and what type of message/data is to be transferred.

The VME schedule allocates time slots for all possible transfers that can occur each second. Some transfers are not necessarily performed every second. When there is no such transfer to perform, there will simply be no data available to transfer, so the allocated time slots on the VME will be "idle".

There is another important aspect of this inter-board communication point that needs to be highlighted here. When data is exchanged across boards, it is being exchanged between software with different DALs. It is imperative to prevent the propagation of faults from lower to higher DALs, so the inter-board messages are protected by several means:

- They are CRC-protected
- The transfers via VME between boards, regardless of the direction of the data flow, are always initiated from the board containing the highest DAL software. B1 is DAL-B/C, B2 is DAL-C, and B3 is DAL-E. So, transfers between B1 and either B2 or B3, are performed under the control of B1. Transfers between B2 and B3 are performed under the control of B2. This means that a B3 never writes to the RAM of B2, and B2 never writes to the RAM of B1.
- Each type of message has a unique, dedicated area in the exchange memory, and they each have an associated counter. The counter is used to make sure that old data (that should have been updated with a new message, but got left behind because of a fault in the originating board) cannot be mistaken for new data.

The tasks that produce the data to be sent over the VME place their data into a buffer in the originating board. The VME scheduler task, when the allotted time slot is reached, triggers the processing of the relevant buffer, fetching the stored data and managing it's transfer.

Similarly, when data is scheduled to be read, the VME scheduler task on the higher-DAL board, at the allotted slot time, initiates the transfer and deposits the received data into a buffer from where it can be picked up by the destination task (the one that actually uses it for something). This ensures that the VME traffic happens in the allotted slots, rather than being determined by when the producer/consumer tasks perform their work. Note that when the VME scheduler in a lower-DAL board triggers a read or write, it does not actually initiate any traffic on the VMEbus, but instead merely access the VME-mapped RAM in its own board, and moves data to/from that area and the local buffers (that are accessed by the other producer/consumer tasks).

Of course, all of this relies on accurate time synchronisation between all three boards, which is another tricky problem, as we shall see next.

7.3. Inter-Board Synchronisation

There are three boards in the MGF, but only one IRIG-B card, which is connected to Board 1. Therefore, the other two boards do not have direct access to the GST.

Nevertheless, all three boards to set their clocks to be synchronised to the external time from the PTF, not just for the purposes of generating or verifying time-stamps on messages, but also in order to ensure that their respective VMEbus schedulers are synchronised with respect to each other, as explained previously.

This is performed by a task in board 1 (DAL B partition) that acquires the external time from the IRIG-B signal, updates the system time of the Board 1, immediately writes that time into a reserved VME-mapped location in the other two boards, and then generates interrupts in the those boards in order to trigger them to go and synchronize their own system clocks using the time provided by B1. This is done once every second, so that there is no significant drift between boards in the meantime.

However, there is another little problem. Board 1 has two partitions, and jitter caused by the partition time-slicing (as already mentioned).

The boards would be synchronised badly if the procedure was unlucky enough to set the system time just before a partition time-slice came along, and then interrupt the other boards just after the other partition time-slice finished, because 3ms (enough to invalidate the VMEbus traffic plan) would have elapsed in the meantime!

Therefore, the synchronisation task in Board 1 (DAL-B) has to be "uninterruptible".

The synchronisation task is very high-priority, and when it starts (once each second), it goes into a continuous loop, reading the time, and comparing it with the previous time it read. When it sees that there has been an (approximately) 3ms jump in the time (instead of the expected, very short time difference), it deduces that there must have been a time-slice in the middle. Therefore, it also knows that it must be at the start of its own partition's time slice, and can therefore go ahead now with the synchronisation, in the knowledge that it won't be interrupted before it has finished.

One further point of interest, regarding timing, is that the MGF also has to tolerate up to 4 hours without the external time reference (the PFT time via the IRIG-B), and still meet its deadlines. In four hours, the internal clocks could drift away (in either direction) from the GST. Therefore, the deadlines that are allotted to the tasks in the MGF software are actually set pessimistically, assuming the worst case. So, when a task deadline is X milliseconds with respect to the GST, the MGF task is given a deadline of X minus 7ms to cope with the potential drift.

7.4. Automatic Boot Procedure

One of the most complex problems that had to be solved in the MGF is the procedure for booting up the element.

There are three boards, and on each board, there is a KDI, an RTOS, loader software to cope with the protocol for starting up the boards in a step-wise fashion, and of course the application software in each board. In fact, there is not just one application software board, because each one has a default (nominal) executable, a backup executable (in case the default, perhaps newly installed, image fails), and possibly a test executable (to be loaded instead of the operational application software, in order to perform trouble-shooting).

Furthermore, the application is not just one executable file, but may be several (the SNMP processes in Board 3 are separate executables), and are accompanied by their associated configuration parameter files.

Here's how it works:

1. At power-on, or after performing a reset in the three processing boards, the RTOS (KDI) is launched by the LynxOS pre-boot application in each processing board, from the image stored in the flash memory.
2. Each LynxOS-KDI starts automatically one pre-configured main application per VM. B1 has two VMs (one for each partition), while B2 and B3 have one VM each. The KDIs also launch a small "loader" application in each partition.
3. The multi-board boot-up is co-ordinated and controlled from B3. It protects the MGF from becoming inoperable as a consequence of loading a corrupted application software or one that fails to start correctly:
   - The B3 loader checks the MD5 digest of the KDI in B3 It writes a special file (to disk), that the application will delete when it starts. If the loader finds the file still there the next time it executes, it will know that the application failed to start correctly, and so will try instead to launch the backup application.
   - The set of files to be loaded into memory and executed, as well as the associated set of configuration files, are listed in "SW package files", and the loaders perform version consistency checks as well as MD5 file digest checks to ensure that everything is correct, before going ahead and loading the software. If there is a version or MD5 error, the loader reverts to the backup application package.
   - Failures detected by the loader are written to file, so that they can be read by the application and/or by the operator.
4. The boot process continues in the B3 application, once it has started. It checks certain disk files to see if there is any unfinished work to do, that was started before a reset, and has to be finished during the boot. This includes unfinished mode change and software installation commands. It also reads another file to get the results of the power-on built-in tests (PBIT) performed by the loader, and writes them into the board's MIB. If the PBIT results indicate a failure, the application automatically transits to FAILED mode.
5. The B3 application loads its own configuration data from the associated disk

files. This too is protected against inadvertent errors in the configuration files. If an error is detected, the application loads hard-coded (fully validated) parameters that allow the MGF to communicate with the GACF.

6. Meanwhile, the loaders in B1 and B2 send a message, once each second, to B3 via VMEbus, to show that they are LOADED and awaiting further instructions. As soon as they get a message from B3 (and so they know that B3 is alive), they proceed to verify their KDIs (versions and MD5 digests), informing B3 (via VME message) of the results.
7. The next step is for the operator (GACF) to command the MGF to go to INITIALISED mode. B3 responds to it by starting the load process on the other two boards.
8. The application files for those boards are actually stored in the disk that is accessible only to B3, to B3 has to provide them (via VMEbus) to the other two boards. Once the loaders in the other boards have successfully received and verified the files, they proceed to launch them.
9. B3 implements a timeout on the software launch in B1 and B2. It starts a timer when the files are sent. The application software in the other boards have to report back to B3 before the timeout expires, otherwise B3 treats it as an error.
10. Once the application software is executing in all three boards, the process continues with the exchange of configuration parameters, also sent from B3 to the other boards via VME. This is another quite complex sequence, also with many verifications to detect potential errors (such as mismatches between applications and configuration parameters). It is further complicated by the need to implement the "Install" command, which changes the default configuration parameters.
11. If all works properly, at the end of this process all the components will have been started in all boards/partitions and the transition to INITIALISE mode continues (this isn't the end of the story!).

When the B3 application software is commanded to transit to TEST mode, it writes another special file to disk to indicate to the B3 loader (which will execute after the board is reset) to load the Test software instead of the application software.

8. CONCLUSIONS

The MGF started out as a single-board design, with its software entirely at DAL-B. The final design is based on three processing boards with four software partitions, at a variety of DALs.

The MGF does use software partitioning in one board, where the message processing and the safety box reside, and that has strong benefits. On the other hand, further software partitioning would not be at all appropriate for the other functions (mainly I/O) in the MGF, and they are better suited to being implemented in a single partition in separate boards.

The consequences of the multi-board architecture are: a lot of internal traffic on the VMEbus; further complexity in order to synchronise the boards and to ensure that they cooperate correctly; and the boot (and mode transition) process is quite complex.

There are many verifications that act as safety barriers to prevent fault propagation between different DALs and to protect against corruption of data (especially important with regard to integrity-related data) within the MGF, in particular when travelling between boards.

Nevertheless, the multi-board architecture is the best solution to cope with the problem of the many physical external interfaces, the CPU load induced by the processing of large amounts of data, the need to guarantee the hard real-time deadlines in the TTA-related portion of the message generation process, and it also strongly supports the DAL reduction (which benefits the software developers).

Identifying the tricky problems in the design, finding the potential fault propagation paths, and then inventing smart solutions to place adequate safety barriers, to ensure the hard real-time deadlines, and so on, has been one of the most interesting aspects of the MGF development project. Ultimately, we are confident that the MGF will be a reliable and safe element of the GMS.

9. ACKNOWLEDGEMENTS

The authors wish to express their gratitude to the following contributors to this paper:

Robert Pagnot, *EADS Astrium-F*

Gérard Bulsa, *ESA/ESTEC*