GERICOS: a generic Framework for the Development of On-Board Software

Philippe.Plasson@obspm.fr
1. Introduction
2. GERICOS framework overview
3. GERICOS::CORE layer
4. GERICOS::BLOCKS layer
5. GERICOS::DRIVERS layer
6. Conclusion and roadmap
■ Space science missions
  - Solar Orbiter, BepiColombo, JUICE, PLATO, Euclid...

■ The payload flight software are often developed by the research labs or science institutes in charge of the instruments.

■ Developing a flight software is complex and costly.

Heavy development and qualification process (coding rules, metrics, robustness analysis, schedulability analysis, code coverage, unit tests, ...)
Starting the development of a flight software from scratch is a huge and risky effort.

Reuse often limited to the adaptation of software pieces inherited from previous projects.

Lack of space-qualified generic software building blocks, available off-the-shelf.

Not a new issue, but an issue more and more significant:
- Continuous increase of the space instrumentation complexity
- Growth of the needs in terms of on-board payload data processing
- Ex.: PLATO with its network of 20 on-board computers

The need of qualified generic frameworks and tools for the development of on-board software is now an obviousness.
By promoting reference architectures and generic software platforms, the space agencies are well aware of this strong need to improve and to ease the development process of the on-board software.

<table>
<thead>
<tr>
<th>Lead Space Agency</th>
<th>Project</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESA</td>
<td>COrDeT, SAVOIR, OSRA</td>
<td>On-board Software Reference Architecture for Platform Software</td>
</tr>
<tr>
<td>ESA</td>
<td>OSRA-P</td>
<td>On-board Software Reference Architecture for Payloads</td>
</tr>
<tr>
<td>CNES</td>
<td>LVCUGEN</td>
<td>Generic solution for the development of payload on-board software based on the time and space partitioning paradigm</td>
</tr>
<tr>
<td>NASA</td>
<td>Core Flight System (cFS)</td>
<td>Platform and project independent reusable software framework and set of reusable software applications</td>
</tr>
</tbody>
</table>
In the context of its space science projects, LESIA has developed and qualified its own generic platform for the development of science payload software (category C software):

- The GERICOS platform (GEneRIC Onboard Software).

The GERICOS studies started in 2007.

At this time, no off-the-shelf solution was available.

The first version of the GERICOS framework is now operational and qualified.

The RPW DPU on-board software (Solar Orbiter) has been built and qualified successfully with the GERICOS framework.
GERICOS framework overview

- GERICOS framework = set of reusable and customizable software components, written in C++, for the rapid development of payload flight software.

- The GERICOS framework is made up of three layers:
  - GERICOS::CORE (1)
  - GERICOS::BLOCKS (2)
  - GERICOS::DRIVERS (3)
GERICOS::CORE layer:

- Extremely lightweight, optimized and space qualified implementation of the active object paradigm on top of a real-time kernel
- Layer including the concepts related to real-time and embedded systems: interrupts, synchronized objects, shared resources, circular buffers, …
- Goal = to be able to quickly build a real-time application using the oriented-object concepts while being independent from a specific RTOS and from a specific H/W target
- A real-time application is built as a set of active objects, each active object having its own message queue and computational thread which processes incoming messages one by one by executing the corresponding methods.
GERICOS::BLOCKS layer:
- Set of reusable software components for building flight software based on generic solutions to recurrent functionalities:
  - Telecommand management
  - Telemetry management
  - ECSS PUS services
  - Mode managements
  - Time management
  - CCSDS protocol
  - Etc.

Application Software (mission specific)
GERICOS::BLOCKS
Building Blocks for Space Software
GERICOS::CORE
Active Object Layer for Embedded Software
RTOS (ThreadX, RTEMS, FreeRTOS…)
Board Support Package Libraries
Hardware target
GERICOS framework overview

- GERICOS::DRIVERS layer:
  - Software drivers corresponding to COTS IP cores:
    - Example: GRSPW IP core (GRLIB SpaceWire IP core used in SoC such as the UT699 LEON3-FT processor).
With GERICOS, a flight software is designed as a system of interacting objects, each object containing encapsulated data and procedures grouped together to form an entity.

For implementing generic components and offering the capability to define deeply modular architecture, the GERICOS framework widely relies on:
- Data abstraction and encapsulation
- Messaging
- Templates mechanisms (generic programming, template metaprogramming)
- Polymorphism
- Interface mechanisms allowing to define a set of services which can be provided by an object A and required by an object B
  - implemented in C++ with classes containing only pure virtual methods

GERICOS framework overview
Object-oriented design

GERICOS UML profile
Safe subset of the C++ language
- No dynamic memory allocation / deallocation with the new / delete operators
- No exceptions (throw, catch and try)
- Restrictions on:
  - multiple inheritances
  - operator overloading
  - type conversion
  - ...

Joint Strike Fighter Air Vehicle - JSF C++ coding standards
- Published in December 2005 by Lockheed Martin Corporation.
GERICOS::CORE layer
Objectives

- Hide the specificity of the real-time kernels.
  - GERICOS::CORE is an abstraction layer built on top of the RTOS.
  - RTEMS 4.10, ThreadX 5.4, FreeRTOS 7.0.2
- Hide the specificity of the hardware targets.
  - Support of LEON processor family fully operational.
  - LEON2-FT AT697, LEON3-FT UT699, LEON3-FT GR712RC, LEON3-FT RTAX
- Use the C++ and the power of the object-oriented programming to develop real-time embedded applications.
GERICOS::CORE layer
Objectives

- Promote the following best practices:
  - modularity
  - reusability
  - loose coupling
  - separation of concerns
  - component based development
  - use of design patterns

- Make the transition from the modeling activity to the implementation activity almost immediate via a Model Driven Engineering approach.
GERICOS::CORE layer
Overview of the GericosCore library

- GericosCore is a C++ library implementing the active object paradigm and providing a set of components for managing the various aspects of a real-time software.
- These components can be used to build the real-time structure of any embedded application.

- Active objects with message-based communication
- Task response time recorder
- Interrupt handlers and interrupt management
- Synchronized objects
- Shared resources
- Timers
- System time retrieving (tick, µs and ms)
- Time span recorder
- Circular buffers and FIFO with locking capabilities
- Single buffers with locking capabilities
- CPU load recorder
- Data rate recorder
- Hardware register management
- Cache controller management
- Power down mode management
With GERICOS::CORE, a flight software is designed as a set of active objects communicating together using
- Asynchronous marshalled operations
- And/or synchronous non-marshalled operations (i.e. standard methods)

A GERICOS active object (called a “task”) encapsulates its own execution thread and an input message queue.
- The GERICOS tasks communicate with the other tasks by the way of their message queues.
- The asynchronous method calls are serialized and posted as messages in the task message queues.

Real-time scheduling model = priority preemptive scheduling.
- A task A calls a service \texttt{m1()} implemented by a task B \textbf{(1)}.
- The method \texttt{m1()} called by the task A is a proxy method implemented by the stub of the task B \textbf{(2)}. The method \texttt{m1()} is exposed in the task B interface.
- The call to this method is marshalled and asynchronous: the stub of the task B processes the method call and transforms it in a serialized message (marshalling process) put in the task B message queue \textbf{(3)}.
- This message will trigger the task B thread \textbf{(4)}.
- Once the message is deserialized by the stub of the class B (unmarshalling process) \textbf{(5)}, the service \texttt{m1()} (a concrete method) is executed in the context of the task B thread \textbf{(6)}.
The active objects are implemented thanks to the proxy design pattern.

The developer can focus on the service implementation while all the marshalling and unmarshalling mechanisms are gathered in the stub class.

With the proxy design pattern, the call to an asynchronous marshalled operation is written in the same way than the call to a standard C++ method.

- Object1->method1()

Example: task implementing the Housekeeping Management Service (PUS service 3).
The GERICOS::CORE tasks offer a functionality for recording and reporting task response time statistics. This functionality is particularly useful for:
- computing the worst response times of each task
- verifying the whole schedulability of a real-time application
- for adjusting the task priorities and the message queue sizes
A specific UML profile has been created for modeling the different concepts provided by the GERICOS::CORE framework.

This UML profile has been used to fully document the design of the Solar Orbiter RPW DPU application software architecture.

The GERICOS UML profile defines for each GERICOS::CORE component:
- an UML stereotype: task, shared object, synchronized objet, timer, interrupt handler, etc.
- a set of tags allowing to specify the real-time and embedded features of the component.

Ex.: tags associated to the task components:
- task priority
- stack size
- message queue size

With the GERICOS UML profile, the UML composite structure diagrams can be enriched with all the information related to the real-time properties of the components.
GERICOS::CORE layer
GERICOS UML profile

- UML composite structure diagram illustrating, with the Solar Orbiter RPW DPU software HK Manager component, how the GERICOS UML profile can be used.
  - The hkManager task is responsible for managing the PUS service 3 (Housekeeping Management).
  - A timer component (hkTimer), is responsible for triggering the hkManager task every 5 ticks, i.e. every 125 ms.
  - The hkManager task needs to know the system time: that’s why it is connected to the timeManager task which maintains the system time.
  - Concrete housekeeping producers whose responsibility is to collect the housekeeping parameters and to gather them in telemetry packets are plugged in the hkManager component.
The architecture of an application built with GERICOS is defined thanks to a specific XML language called **GERICOS::XAL**.

This language defines a set of keywords in the form of XML elements allowing to describe the instantiation of the various components depending on their types.

The features of each component instance are managed via XML attributes.

The GERICOS::XAL language defines also keywords for describing the connections between the components.
The C++ code performing the declaration, the definition and the initialization of the different component instances is entirely generated automatically thanks to:
- The GERICOS::XAL description of the architecture.
- A set of XSL transformation style sheets.
GERICOS::CORE layer
Code generation

- All the objects defined in the GERICOS::XAL descriptions are not instantiated on the heap dynamically at startup:
  - Defined as global variables.
  - Located in the .bss sections.
  - Easily identifiable in the linker memory maps.

- All the code making the connections between the various component instances of an application is also fully generated from the GERICOS::XAL descriptions.

- The auto-generated initialization and connection routines are automatically called during the application startup.
  - The user can still add its own extra initialization code in an extension routine intended to this use.
GERICOS::BLOCKS layer
Overview

- **GericosBlocks** is a C++ library containing a set of interfaces, templates and classes implementing, in a generic and reusable way, services which are recurrent in the domain of the flight software.

Standard PUS services commonly required in the science payload instruments + Other services not directly linked to the PUS standard = GericosBlocks
The PUS standard defines a set of generic services whose associated requests (telecommand packets) and reports (telemetry packets) are standardized. Theses services correspond to operational concepts related to the monitoring and control of on-board segments. Each service (ex.: Housekeeping and diagnostic reporting service) is itself made up of several sub-services (ex.: Enable report generation, Disable report generation, etc.). The GericosBlocks library offers an implementation of the PUS services and sub-services focused on what is usually required at payload level in the ESA missions:

- **PUS service 1**: Telecommand verification service.
- **PUS service 3**: Housekeeping and diagnostic reporting service.
- **PUS service 5**: Event reporting service with event flow regulation mechanism.
- **PUS service 6**: Memory management service supporting local and remote memory.
- **PUS service 9**: Time management service.
- **PUS service 17**: Test connection service.
For each service, the GericosBlocks library provides an implementation based on a consistent set of *templates*, *classes* and *interfaces* which can be easily instantiated, assembled and incorporated in any flight software whose architecture relies on the GERICOS framework.

**Example: service 3 (Housekeeping management)**

- The flight software developer has only to focus on the development of the housekeeping producer component which are very simple components collecting data and gathering them in packets.
- The housekeeping producers are plugged in a generic HK manager component thanks to a standardized interface.
- All the other mechanisms required for implementing the PUS service 3 are offered by the GericosBlocks library.
### What is part of the framework?

<table>
<thead>
<tr>
<th>Class/Interface</th>
<th>Description</th>
</tr>
</thead>
</table>
| GsbHkManager    | Interface defining the sub-services of the PUS service 3: `enableHk(pid, sid)`, `disableHk(pid, sid)`, `requestHk(pid, sid)`, `changeHkPeriod(pid, sid)`, `processPeriodicHk()`.
| GsbHkManagerStub, GsbHkManagerImpl, GsbHkTimer | Classes implementing the active object and the timer responsible for processing the requests defined in the `GsbHkManager` interface. |
| GsbTmControlEntry, GsbTmWriter, GsbTmGenerator, GsbTmProducer, GsbTmProducers, GsbHkProducers | Set of templates, interfaces and classes providing generic mechanisms for requesting, enabling, disabling the HK packet generation and for changing the generation period. |
| GsbTcDisableHk, GsbTcEnableHk, GsbTcUpdateHkPeriod, GsbTcRequestHk | Templates defining the telecommand packets corresponding to the PUS service 3 requests. |
| GsbHkReportingTcParser | Template defining a generic PUS service 3 telecommand parser. |
GERICOS::BLOCKS layer
Example: PUS Service 3 implementation

■ What shall be done at user level?

Implement concrete housekeeping producers responsible for collecting the data to be reported and to build the telemetry packet.

Instantiate the PUS service 3 TC packet templates and the PUS S3 TC parser template.

Declare and configure the GsbHkManager task and timer in the GERICOS::XAL file.

Plug the concrete housekeeping producers into the GsbHkManager task via the GERICOS::XAL file.

Plug the PUS S3 TC parser into the GsbTcRunner task via the GERICOS::XAL file.
Packet handling (the concepts of generic packet and protocol layer are modeled thanks to C++ templates).

TC management (reception, queuing, parsing, execution).

Telemetry management (multi TM FIFO, TM flow regulation against the modes, time-stamping, sequence counter management, TM filtering by process ID and category, etc.).

CCSDS protocol management (via C++ templates).

SpaceWire protocol management.

RMAP protocol management.

RMAP boot process management.

Time management supporting the CUC format (CCSDS Unsegmented Time Code).
<table>
<thead>
<tr>
<th>Mode management (with the concept of mode observers, mode transitions and actions upon mode transitions).</th>
<th>Lossless compression based on the Rice algorithm.</th>
<th>Persistent data block management with save and restore capabilities in non-volatile memory or in remote memory via RMAP.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic EEPROM driver.</td>
<td>Generic idle task managing the processor power down mode.</td>
<td>Basic mathematic functions (exp, power, ln, ...).</td>
</tr>
</tbody>
</table>
The GERICOS::DRIVER layer is made up of several C++ libraries providing software drivers for IP cores belonging to the LEON processor ecosystem (GRLIB IP core library).

Built using the GERICOS::CORE components.

Compatible with the GERICOS::BLOCKS interfaces.

Fully independent of any RTOS or any specific BSP library.
<table>
<thead>
<tr>
<th>Library Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GericosDriversAmba library</td>
<td>• Components managing the GRLIB AMBA plug’n’play areas.</td>
</tr>
<tr>
<td></td>
<td>• Mapping addresses and the interrupt assignment.</td>
</tr>
<tr>
<td>GericosDriversGrspw library</td>
<td>• Very efficient implementation of a SpaceWire software driver compatible with the GRSPW IP core.</td>
</tr>
<tr>
<td></td>
<td>• Advanced functionalities (SpaceWire communication failures, connection and transmission timeouts).</td>
</tr>
<tr>
<td>GericosDriversAhbstat library</td>
<td>• Components managing the registers defined in the AHBSTAT IP core.</td>
</tr>
<tr>
<td></td>
<td>• Used for reporting AHB bus error statistics and for building memory scrubbers.</td>
</tr>
<tr>
<td>GericosDriversIrqmp library</td>
<td>• Classes mapping the registers defined in the IRQMP IP core (IP core managing the interrupts).</td>
</tr>
</tbody>
</table>
The GERICOS framework V1 is operational and will flight on Solar Orbiter.

With GERICOS, creating the core of a new payload flight software implementing the basic PUS services requires only a couple of days.

- Only a matter of configuring, assembling and connecting together generic pieces of software.

The GERICOS framework V2 is under development.

- Improvement of GERICOS V1
- R&D action funded by the Labex ESEP (network of nine French labs involved in planetary exploration space missions).
- The GERICOS framework v2 will be used by LESIA to build the PLATO N-DPU flight software.
GERICOS framework
Conclusion and roadmap

- **Goal 1**: supporting the multi-core processor architectures
  - AMP approach.
  - Extension of the GERICOS::CORE active object layer to the multi-core architectures.
  - Extension of the GERICOS::XAL language.
  - Building chain making transparent the AMP approach.
  - Work in progress.
  - The first demonstrator is operational.

- **Goal 2**: porting the GERICOS::CORE layer to the ARM processors
  - Objective = to use the GERICOS framework in the context of the CubeSat projects.

- **Goal 3**: making the framework available to other labs and institutes
GERICOS framework

- Any questions?