A Component-based Framework for Space Flight Software

Guillaume Veran, Gerald Garcia
A Component-based Framework for Space FSW

- SEMS Component model
- Process overview
- Code structure
- First feedback from GB2
- Perspectives
- Conclusion
A Component-based Framework for Space FSW
Component based software engineering

A FSW is a set of components
- Components encapsulate application “business” code
- Components are managed by containers
- Underlying runtime provides execution and communication mechanisms

Separation of concern
- Functional: Algorithms are inside the components
- Non-functional: Runtime resources are implemented in the container
System Engineering and Middleware for Space domain (SEMS):

- Based on LwCCM
  - Subset of the Corba Component Model
  - Designed for embedded domain
  - Official OMG specification

- Components are defined by:
  - Provided and required services
  - Published and consumed events
  - Attributes

- Components provide services to:
  - Other components
  - The ground station/operators
Some “space-oriented” extensions have been added:

- Data types extensions (engineering data types, constrained types…)
- Specific communication standard (CCSDS)
- Standardized identification of spacecraft services (PUS)
- Customized communication schemes between components to optimize resources

```
<<interface>>
I_SpacecraftCommands
setMode(in mode_t mode)
```

**Ground-Board interface**

*Offering a PUS service to change the spacecraft mode*

- The component interface and attribute definition is used to automatically produce:
  - The ground/board Interface Control Document (ICD)
  - TM/TC/Event handling code
SEMS Process

1. Component types definition:
   - Definition of specific types (arrays, ranges, structure…)
   - Definition of provided & required services
     - Identification of services callable from ground (PUS)
   - Definition of component attributes
     - Including visibility rules (visible from ground, adjustable from ground)
   - Published and consumed events

2. Component implementation:
   - Implementation language, RTOS, ports implementation
   - One component type can have several implementations

3. Component instantiation:
   - Creation of component instances

4. Component deployment and configuration
   - Allocation of the component instances on tasks
   - Definition of the connections between component instances
   - Definition of properties on components and ports
A Component-based Framework for Space FSW

SEMS Code structure

Application code:
- Functional code of the component
- Hand-coded or generated from functional modeling tools
  - Simulink
  - ...

Containers code:
- Adapt component user code to the actual platform
- Implement communication between components
- Mediate between user code and technical services

Deployment code:
- Connect component instances
- Initialize component attributes

SEMS runtime:
- Provides tasking (RTOS) and communication mechanisms (SwBus) for the container
A Component-based Framework for Space FSW
SEMS Editors

Supports of several diagrams
- Component type definition diagram
- Component implementation diagram
- Instantiation & Configuration diagram
- Deployment Diagram

Ada 95 files

LwCCM model

Container API generator

Component deployment generator

LwCCM
Code generation

- Generated code
  - TM/TC/Event handling code
  - Component skeletons
  - Component containers
- Based on Acceleo M2T transformation
  - ADA 95 or C language
  - Targeting LwCCM execution platform
- Good code quality level
  - designed by C/Ada experts
- Very homogeneous design
- Easiness to introduce new capabilities common to all the components:
  - Checks, behaviours, …

Documentation generation

- Generation of the Interface Control Document (ICD)
- Generation of the Satellite Database Files
Globalstar 2 return on experience

- 48 LEO satellite constellation
  - Personal phone/multi-media communications
  - To replace Globalstar 1 constellation

- CBSE process has been applied on GBS 2
  - OBSW based on TAS Generic OBSW design
  - Software bus and component model

- This process have lead to productivity gains
  - The design phase is not significantly longer
  - Reduction of the coding and UT phases
  - Reduction of the number of anomalies reported on software interfaces

- Code size: No difference with hand-written solution

- Little runtime overhead compared to non-modular applications
Extension of the component model to support extra-functional properties

- Annotation of the component models with extra functional properties:
  - RT properties: deadlines, periods, WCET, bus load…
  - Dependability properties: Safety, integrity, reliability…
- Integration of tools for the analysis of the extra-functional at the model level
- Preservation of the extra-functional properties at runtime

**Functional code generation**

- Generation of the components internal code from heterogeneous functional models:
  - Simulink, Scylab, UML
  - Other domain specific modelling tools
- Automatic integration in the component containers
- Definition of a unified process to make sure that the component designs and their implementations are consistent
Functional code is independent from deployment code:
- Ease the reuse of code (consequence of the application design)
- Components can be redeployed easily

SEMS Component model improves the code structure:
- Productivity improvement (automatic code generation)
- Makes the analysis of applications easier (scheduling, etc.)

SEMS and MDE approach is the baseline for all our future projects:
- Sentinel-3
- O3B
- Iridium Next
- ...
Questions