INTEGRATED SOLUTION FOR RAPID DEVELOPMENT OF COMPLEX GNC SOFTWARE

Jean-Sébastien Ardaens [(1)], Gabriella Gaias [(2)]

[(1)] German Space Operations Center (DLR/GSOC), 82234 Wessling, Germany, jean-sebastien.ardaens@dlr.de
[(2)] German Space Operations Center (DLR/GSOC), 82234 Wessling, Germany, gabriella.gaias@dlr.de

ABSTRACT

The paper describes the integrated software solution retained for the design and development of the AVANTI experiment, a challenging on-board autonomous formation-flying endeavour to be conducted in 2016. This solution aims at enabling rapid prototyping by providing a powerful development, validation and testing environment, able to support simultaneously the design and validation of novel Guidance, Navigation and Control algorithms, the definition and documentation of the interfaces with the ground segment, the implementation of the onboard software using space quality standards, the integration into an existing satellite bus and all related testing activities.

1. INTRODUCTION

The possibility to conduct a spaceborne experiment is a rare and precious opportunity that has to be taken even in the presence of severe development constraints such as time pressure or limited human resources.
The Autonomous Vision Approach Navigation and Target Identification (AVANTI) experiment envisioned by the German Aerospace Center belongs to such unique challenging endeavours with limited resources. AVANTI aims at demonstrating fully autonomous approach to a non-cooperative object in a safe and fuel-efficient way using a simple camera. It is implemented onboard the DLR's BIROS [1] satellite, scheduled for launch in 2016, and takes advantage from the fact that BIROS embarks a third-party picosatellite which will be ejected in-orbit prior to the start of AVANTI and which will thus serve as non-cooperative target for the sake of the experiment
AVANTI is a complex experiment, developed by a small team (two researchers) in a limited time frame (3 years), making use of several satellite components (thruster system, attitude control, onboard camera) and undergoing many constraints (thermal and power requirements, visibility constraints of the target satellite, limited onboard resources, safety of the formation, limited manoeuvring capabilities, ground communication, telemetry budget).
In addition, the nature of the experiment requires extensive tests and validation of the algorithms and software on-ground before the ejection of the picosatellite. Dealing with a formation of satellites at low altitude (500km) with very different ballistic coefficients, it has to be avoided to fix problems - which could have been detected before - after establishing the formation because of the high propellant cost required to maintain the formation.
After a brief description of the envisioned experiment, the paper proposes some ideas to reduce the development and validation efforts of such a GNC experiment while ensuring a smooth integration in the satellite bus to augment the chance of successful completion of the experiment.

2. OVERVIEW

2.1. The AVANTI Experiment

As mentioned in the introduction, AVANTI will demonstrate the capability to perform rendezvous and receding approaches with respect to a noncooperative client satellite making use of vision-based angles-only measurements. The experiment focuses on far- to mid-range separations (several hundred meters to 10 km). The capability to approach and rendezvous a non-cooperative orbiting object in a safe, fuel efficient, and accurate manner is in fact a key requirement for future on-orbit-servicing and debris-removal missions. In this context, the exploitation of angles-only navigation is appealing since it relies on simple passive low-cost sensors (e.g., optical or infrared cameras) able to provide the line-of-sight direction to the target object. To that end, the star trackers usually employed for attitude determination can be advantageously used also to track a space object, if properly oriented [2]. At sufficiently large separations, where it is acceptable to approximate the center of mass of the client satellite with its intensity centroid, angles-only navigation represents a sufficiently accurate methodology to accomplish the first phases of the approach. This leads to a simpler and cheaper design of the servicer satellite, restricting the sensor complexity to close-proximity operations, for which more accurate, costly and power-demanding sensors might be required.
The experiment poses many non-trivial challenges:

- It is not easy to distinguish the target spacecraft at far range from the surrounding stars.
- The angles-only relative navigation problem is weakly observable, so that maneuvers have to

be planned and executed to solve the ambiguity in the range determination.

- These maneuvers are part of a rendezvous guidance profile, which has to be done in a safe a fuel-efficient way and which has to be computed autonomously onboard. This guidance profile has to be robust in case of failure of the thruster system and degraded onboard navigation.
- Since BIROS does not have 3D maneuvering capability, the spacecraft needs to rotate to execute a maneuver, in which case the target spacecraft exceeds the field of view of the camera.
- Eclipses and camera blinding due to the Sun affects also the visibility.
- Orienting constantly the star tracker towards the target picosatellite has some impacts in terms of power and thermal budget.
- The high differential drag encountered at low altitude results disturbs greatly the navigation and control algorithms.
- The BIROS onboard computer has limited resources which make it not really suited for image processing.

Further details about the mission can be found in [3].

2.2. Software Architecture

Figure 1 depicts the different components of the AVANTI software and their interaction with other systems of the spacecraft. First, the images collected by the star tracker are processed by the Image Processing (IMP) module, which identifies the target spacecraft among all the luminous spots comprised in the image and delivers line-of-sight measurements to the Relative Orbit Determination (ROD) module. For simplicity, IMP uses the knowledge of the onboard attitude to identify the stars present in the image without the need of using any lost-in-space star identification algorithm. ROD implements an extended Kalman filter to estimate the current relative state of the formation.

The formation state estimate is delivered to the Maneuver Planing and Commanding module (MAP), which is in charge of establishing a rendezvous profile. To that end maneuver commands are sent directly to the Attitude and Orbit Control System. For simplicity, it has been decided that MAP takes over the spacecraft attitude profile throughout the entire experiment, in order to point the star-tracker toward the expected position of the picosatellite, to rotate the spacecraft during the execution of maneuvers and orient the communication antenna towards the Earth during the ground contacts.

The AVANTI experiment is implemented as independent application running on the BIROS onboard computer, which is based on a PowerPC architecture and makes uses of an in-house embedded real-time operating system, called RODOS (Realtime Onboard Dependable Operating System). The RODOS real-time kernel provides an integrated object-oriented framework interface (using C++) to multitasking resource management. Compared to other existing operating systems, RODOS focuses on offering the simplest and smallest interface to user applications, while still providing all the required functionality and flexibility (including time management, CPU and memory management) [4].

3. SIMULATION ENVIRONMENT

3.1. High-Fidelity Simulink-Based Simulator

Such a complex experiment requires an advanced simulation environment to model precisely the perturbations acting on the system, understand the interactions between the different spacecraft component,

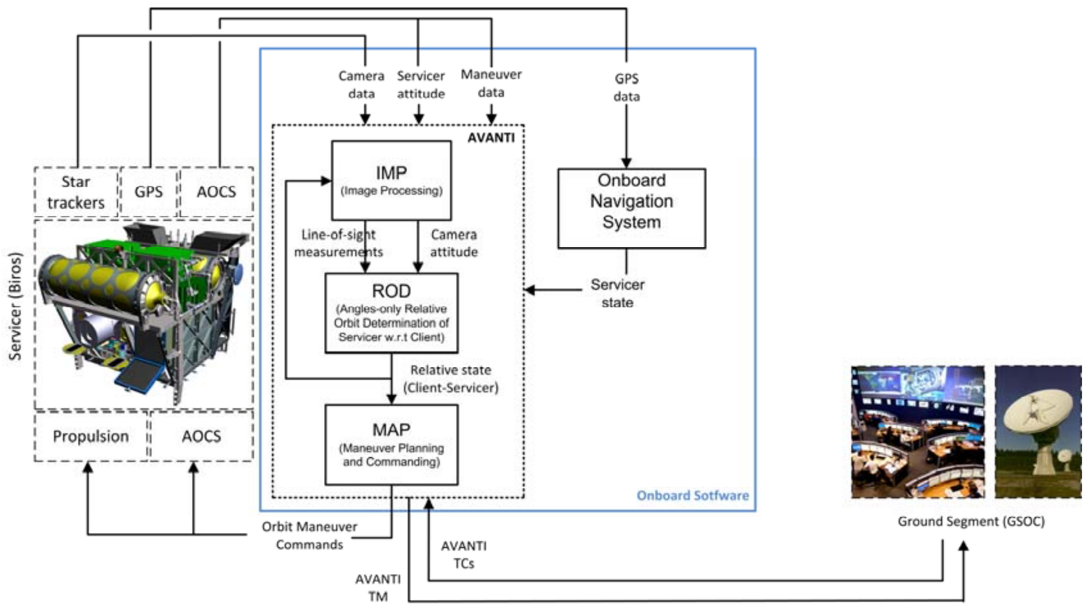


Figure 1: AVANTI software architecture and interfaces to space and ground segments.

assess the impact of the error sources, define and verify the behaviour in case of contingencies, etc. To that end, the German Aerospace Center can relies on its Multi-Satellite Simulator (MSS) [5], which was successfully used in the past to design advanced autonomous embedded formation-flying systems like the TanDEM-X Autonomous Formation Flying System [6] or SAFE experiment [7] using the PRISMA formation flying technology demonstrator [8]. MSS is based on a MATLAB/Simulink environment (which is widely spread in the aerospace community) and comprises a collection of in-house aerospace components to model precisely the environment (gravity field, orbital perturbations, position of the celestial objects, Earth orientation) as well as key sensors and actuators.

3.2. Hardware-in-the-Loop Testbed

In order to ease the final integration in the satellite bus, a hardware-in-the-loop testbed in currently under construction. The objective is to run the flight software with an engineering model of the star tracker. To that end, the field of the view of the camera is simulated using a monitor in a black chamber, so that the interfaces to the camera can be verified.

4. INTERFACING SIMULINK WITH THE FLIGHT SOFTWARE

4.1. Motivations

In order to reduce the development efforts, it is tempting to begin very early with the implementation of the flight software, while the underlying core algorithms are still under investigation and development. This approach presents the advantage of providing well in advanced relevant system information (telemetry budget, interface definition, system behaviour, etc.) to the other partners of the project.
On the other hand, the development of novel complex GNC algorithms might require the use of an advanced and mature Simulink-based development environment. This poses severe constraints in terms of software design, since Simulink does not support all the features of object-oriented programming (direct call of a class method, polymorphism, handling of events, etc.). Similarly, the RODOS operating system is not designed to be triggered by an external system like Simulink (in particular in what concerns the timing and threads handling).

4.2. Stub Components

The retained solution is to emulate the key functionalities of the onboard operating system (timing, threading, hardware communication, etc.) under on the host computer to create an image of the onboard software in the form of a dynamic library, which is then called by the different S-Functions composing the Simulink simulation. Functionalities which are not needed for the simulation are replaced by stubs.
Figure 2 provides a graphical representation of this approach. An almost empty early prototype of the flight software is created when starting designing the algorithms. This draft software is implemented using the target programming language (C++) and relies on the libraries provided by the target operating system (RODOS), which provides essential functionalities: access to the other devices of the satellites bus, timing and threading functions, handling of telemetry and telecommands, etc.
The flight software is then compiled as independent dynamic library on the host computer. At this stage, it is necessary to emulate all the functionalities of the operating system needed by the flight software and replace them by stubs if necessary.
Step by step, the flight software grows and becomes more mature while the algorithms are developed. The main advantage of this approach is that the flight software does not need to be adapted for the sake of the simulations. In fact, it uses the same C++ objects and methods as the ones available on the satellite bus. On the host PC, the stubs are implemented as gateways to the Simulink blocks. For example, a Simulink model of a GPS receiver computes a simulated GPS navigation fix, which is sent to the stub of the object instance interfacing the GPS receiver. The stub fed with simulation data provides them when requested by the flight software as if a data from real hardware components were read.

4.3. Task Execution Scheduling

The difficulty here is that the routines provided by the target operating system might be fundamentally

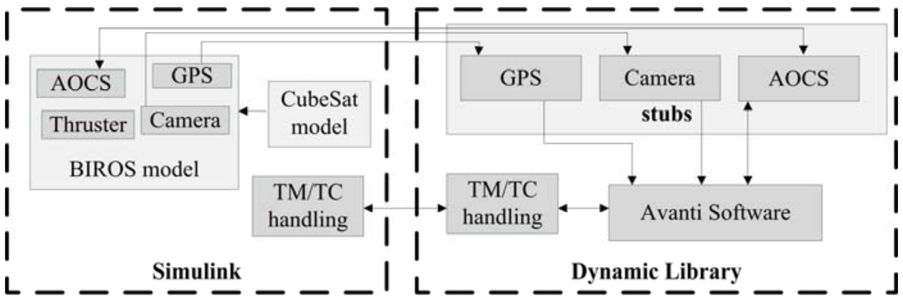


Figure 2: Integration of Simulink and exogen software

incompatible with a Simulink approach. In RODOS, an application is implemented as infinite loop whose time execution is controlled by the operating system:

```
class MyApplication: public Thread {
public:
  MyApplication (): Thread("name", priority) { }
  void init() { /*initialization*/}
  void run() {
    TIME_LOOP(offset, period) {
      /*work*/
    }
  }
}
```

Here, `TIME_LOOP` is a function provided by the operating system to execute the content of the loop at a desired frequency. The problem is that Simulink is designed to take over the task scheduling, i.e. Simulink decides when to execute the content of loop (called a step).
The solution which has been retained is to replace the RODOS thread object by a dedicated thread object running on the host platform which is completed by additional methods to control the execution of the loop externally. If `OSThread` represents a typical thread object of the host operating system, this translates simply into:

```
class Thread: public OSThread {
public:
  // executes the content of the loop
  void doStep() {step=true;}
  // returns the number of loop executions
  long getCount(){return count;}
protected:
  // returns true when the loop must be executed
  bool waitForTrigger()
    {while(!step){};step=false;}
  long count;
  bool step;
};
```

As a result, it is enough to redefine the function `TIME_LOOP` as:

```
#define TIME_LOOP(begin,period) for (count=0;
waitForTrigger();count++)
```

to control the thread execution from Simulink. Since Simulink and the image of the flight software are running in different threads, it is required to wait that the content of the loop has been completely executed to consider that a step has been done. If `application` stands for one piece of functionality of the flight software, the step function of the corresponding Simulink block contains simply:

```
long count=application.getCount();
application.step();
while (count == application.getCount()) {};
```

5. INTERFACE DEFINTION AS METADATA

Interfaces are one of the most important aspects for the successful integration of a subsystem into a more complex system. As a result, special care must be taken to define precisely the interfaces in the very early phases of the project. The interfaces definition is however subject to frequent updates throughout the development process. Automation can be of great help to reduce the efforts needed to reflect the interface changes in the documentation and software. It has been found useful to rely on a central data base on which all the development and documentation tools are based.
The data base contains:

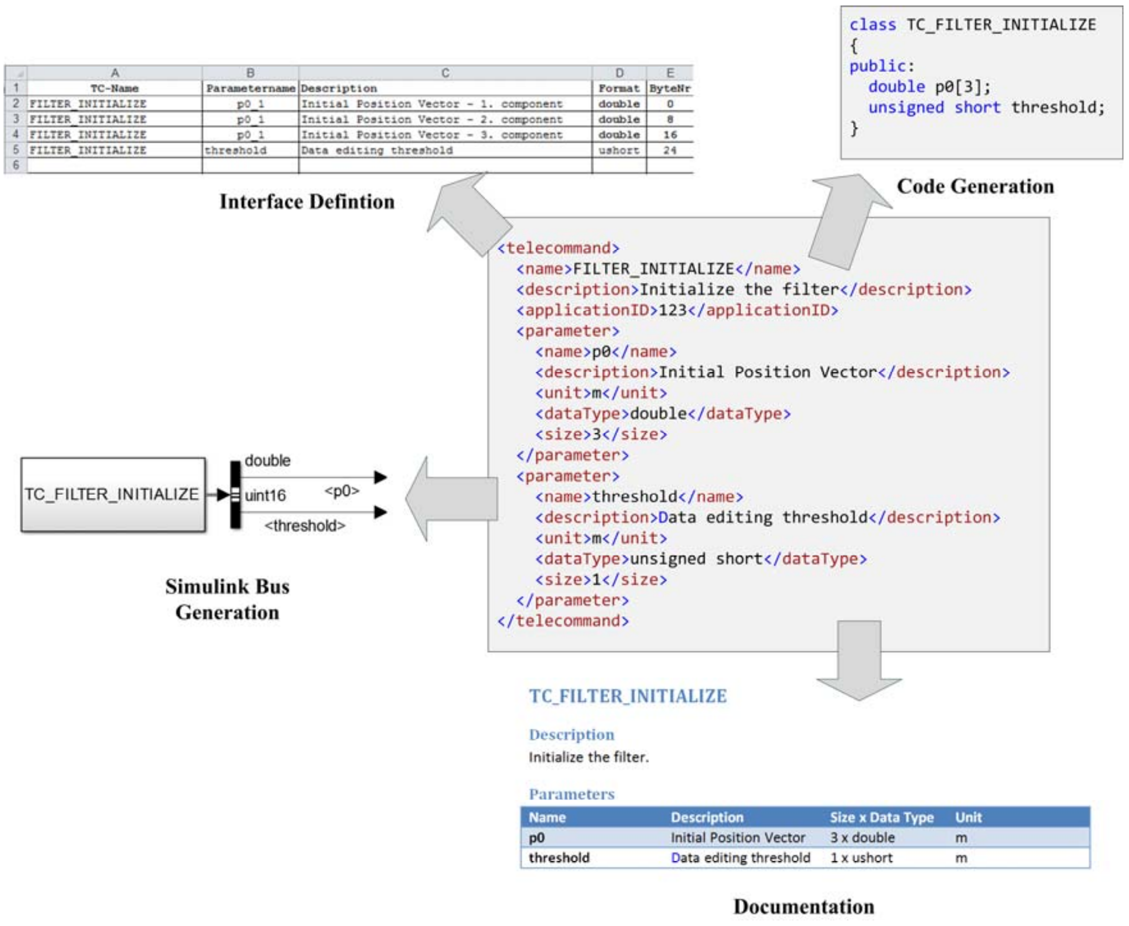
- The list of all telecommands, comprising Application ID, description, and list of parameters.
- The list of all outputs of each component of the GNC system.
- The list of all telemetry packets, comprising Application ID, description and content, which is made of a selection of the outputs of the GNC component.

As depicted on Fig. 3, the centralization of all interface information into a unique data base allows updating quickly the project and ensuring the overall consistency of the interfaces. In particular, the update process takes care of:

- Generating the C++ objects describing the inputs and outputs of every GNC component.
- Generating the C++ functions to generate telemetry packets by assembling the outputs of the GNC components and to read the content of a packet.
- Creating the Simulink bus describing the data flow.
- Updating the Interface Control Document and filling the interface database of the ground segment.

6. CONCLUSION

Bringing a complex GNC experiment on a real mission is a challenging task. Two solutions have been found to be particularly useful in the quest of optimizing the available human resource and reducing the development efforts. The direct interfacing of a mature and validated existing Simulink-based simulation environment with the flight version of a GNC experiment running in a different environment allows developing a prototype which can be immediately and effortlessly ported to the onboard computer. The centralization of all information related to the interfaces into a single data base allows instead significant gain of productivity when updating the software and the documentation.



	A	B	C	D	E
1	TC-Name	Parametername	Description	Format	ByteNr
2	FILTER_INITIALIZE	p0_1	Initial Position Vector - 1. component	double	0
3	FILTER_INITIALIZE	p0_1	Initial Position Vector - 2. component	double	8
4	FILTER_INITIALIZE	p0_1	Initial Position Vector - 3. component	double	16
5	FILTER_INITIALIZE	threshold	Data editing threshold	ushort	24
6					

```
class TC_FILTER_INITIALIZE
{
public:
  double p0[3];
  unsigned short threshold;
}
```

```
<telecommand>
  <name>FILTER_INITIALIZE</name>
  <description>Initialize the filter</description>
  <applicationID>123</applicationID>
  <parameter>
    <name>p0</name>
    <description>Initial Position Vector</description>
    <unit>m</unit>
    <dataType>double</dataType>
    <size>3</size>
  </parameter>
  <parameter>
    <name>threshold</name>
    <description>Data editing threshold</description>
    <unit>m</unit>
    <dataType>unsigned short</dataType>
    <size>1</size>
  </parameter>
</telecommand>
```

TC_FILTER_INITIALIZE

Description

Initialize the filter.

Parameters

Name	Description	Size x Data Type	Unit
p0	Initial Position Vector	3 x double	m
threshold	Data editing threshold	1 x ushort	m

Figure 3: Automated Interface Work using Centralized Data Base

7. REFERENCES

1. Reile, H., Lorenz E. & Terzibaschian, T. (2013), The FireBird Mission - A Scientific Mission for Earth Observation and Hot Spot Detection, Small Satellites for Earth Observation, Digest of the 9th International Symposium of the International Academy of Astronautics, Wissenschaft und Technik Verlag, Berlin, Germany.
2. Jørgensen, J. L., Denver T., & Jørgensen, P. S. (2004), Using an Autonomous Star Tracker as Formation Flying Sensor, Fourth Symposium on Small Satellites Systems and Services, European Space Agency, La Rochelle, France.
3. Gaias, G., Ardaens, J.-S. & D'Amico, S. (2014), The Autonomous Vision Approach Navigation and Target Identification (AVANTI) Experiment: Objectives and Design, 9th International ESA Conference on Guidance, Navigation & Control Systems, Porto, Portugal.
4. Montenegro S. (2005), RODOS, DLR-Network Centric Core Avionics TN 05-08, Deutsches Zentrum für Luft- und Raumfahrt, Germany.
5. Gaias, G., Ardaens, J.-S. & D'Amico, S. (2011), Formation, Formation Flying Testbed at DLR's German Space Operations Center (GSOC), 8th International ESA Conference on Guidance, Navigation & Control Systems, Karlovy Vary, Czech Republic.
6. Ardaens, J.-S. & D'Amico, S., Spaceborne Autonomous Relative Control System for Dual Satellite Formations (2009), Journal of Guidance Control and Dynamics, Vol.32, No.6, pp:1859-1870, doi: 10.2514/1.42855.
7. D'Amico, S., Ardaens J.-S. & Larsson R., Spaceborne Autonomous Formation Flying Experiment on the PRISMA Mission (2012), Journal of Guidance Control and Dynamics, Vol.35, No.3, pp:834-850, doi: 10.2514/1.5563.
8. Bodin, P., Noteborn, R. , Larsson, R., Karlsson, T., D'Amico, S., Ardaens, J. S. , Delpech, M. & Berges, J. C. (2012), Prisma Formation Flying Demonstrator: Overview and Conclusions from the Nominal Mission, No. 12-072, 35th Annual AAS Guidance and Control Conference, Breckenridge, Colorado, USA.