

Deploy deliverable D20 – Pilot Deployment in the Space Sector



Project DEPLOY
Grant Agreement 214158
*“Industrial deployment of advanced system engineering methods for
high productivity and dependability”*



D20 D3.1 – Report on Pilot Deployment in the Space Sector

29.1.2010

<http://www.DEPLOY-project.eu>



List of Contributors

The following people contributed to the writing of D20 at SSF:

- Dubravka Ilić
- Timo Latvala
- Pauli Väisänen
- Kimmo Varpaaniemi

The following people contributed to the writing of D20 at Åbo Akademi:

- Linas Laibinis
- Elena Troubitsyna

Reviewers

This document has been reviewed by:

- Thai Son Hoang
- Kaisa Sere

Table of Contents

List of Contributors	2
List of Figures	5
List of Tables	5
1. Introduction and Aims of the Work	6
1.1 Goals of Pilot Developments	6
1.2 References	7
1.2.1 Applicable Documents	7
1.2.2 Reference Documents	7
1.3 Abbreviations	8
2. Pilot Descriptions	11
2.1 BepiColombo SIXS/MIXS OBSW	11
2.1.1 Overview	11
2.1.2 BepiColombo SIXS/MIXS OBSW Requirements	12
2.1.2.1 Mode Management Requirements	12
2.1.2.1.1 CSW	12
2.1.2.1.2 SIXS-P ASW	13
2.1.2.1.3 SIXS-X ASW	14
2.1.2.1.4 MIXS-T ASW	15
2.1.2.1.5 MIXS-C ASW	16
2.1.2.2 Compatibility between System-Level Modes and ASW Modes	16
2.1.2.3 Telecommands and Telemetry Packets	17
2.2 AOCS	18
2.2.1 Overview	18
2.2.2 AOCS Manager	19
2.2.2.1 Control Algorithms	19
2.2.3 Mode Manager	20
2.2.4 Unit Manager	20
2.2.4.1 Sensors	20
2.2.4.2 Actuators	21
2.2.4.3 Payload	21
2.2.4.4 Unit Status	21
2.2.5 Mode Transitions	22
2.2.5.1.1 Off to Standby	22
2.2.5.1.2 Standby to Safe	22
2.2.5.1.3 Safe to Nominal	23
2.2.5.1.4 Nominal to Preparation	23
2.2.5.1.5 Preparation to Science	24
2.2.6 FDIR	24
2.2.6.1 Handling Mode Transition Errors	24
2.2.6.2 Handling Attitude Errors	25
2.2.6.3 Handling Unit Errors	25
2.2.7 <i>A Simplified AOCS by Åbo Akademi</i>	26
3. Technical Descriptions	28
3.1 BepiColombo SIXS/MIXS OBSW Requirements Modeling	28

Deploy deliverable D20 – Pilot Deployment in the Space Sector

3.2.	AOCS Modeling by SSF.....	30
3.3.	AOCS Modeling by Åbo Akademi.....	32
3.4.	Comparison between SSF's and Åbo Akademi's AOCS Modeling Approaches	34
4.	Measurements	36
4.1.	Pilot on BepiColombo SIXS/MIXS OBSW	36
4.1.1.	Coverage in Requirements Modeling.....	36
4.1.2.	Proof Statistics.....	36
4.2.	Pilot on AOCS	37
4.2.1.	Architectural Coverage	37
4.2.2.	Coverage in Source Code Modeling	38
4.2.3.	Proof Statistics.....	38
5.	Lessons Learnt	40
5.1.	Assessment of Suitability of Event-B	40
5.2.	Proofs	41
5.3.	Model Checking.....	43
5.4.	Requirements Modeling.....	44
5.5.	AOCS Modeling	44
5.6.	Team Work	45
6.	Conclusions and Towards Enhanced Deployment	45
6.1.	Towards Enhanced Deployment.....	46

List of Figures

Figure 1 High-level architecture of BepiColombo SIXS/MIXS OBSW	11
Figure 2 System-level modes and mode changes in BepiColombo SIXS/MIXS OBSW	12
Figure 3 BepiColombo SIXS-P ASW modes and mode changes.....	13
Figure 4 BepiColombo SIXS-X ASW modes and mode changes	14
Figure 5 BepiColombo MIXS-T ASW modes and mode changes	15
Figure 6 AOCS top level schema	18
Figure 7 AOCS unit status change diagram	22
Figure 8 AOCS PLI reconfiguration of redundant unit.....	26
Figure 9 Architecture of a simplified AOCS	27

List of Tables

Table 1 System-level mode transitions in BepiColombo SIXS/MIXS OBSW	13
Table 2 BepiColombo SIXS-P ASW mode transitions.....	14
Table 3 BepiColombo SIXS-X ASW mode transitions	15
Table 4 BepiColombo MIXS-T ASW mode transitions	16
Table 5 Allowed ASW modes in BepiColombo SIXS/MIXS OBSW system-level modes	17
Table 6 TCs and TMs within the scope of the BepiColombo SIXS/MIXS OBSW pilot.	17
Table 7 Initial unit states in respective AOCS modes.....	23
Table 8 AOCS rollback rules upon attitude error.....	25
Table 9 Proof statistics of BepiColombo_Models_v5.0.....	37
Table 10 Proof statistics of DepSatSpec015Model000.....	38
Table 11 Proof statistics of Modes_v2	39

1. Introduction and Aims of the Work

The main goal of SSF's involvement in the DEPLOY project is the ultimate Rodin Platform integration chain with respect to current SSF processes. This complex goal is reflected in several more focused subgoals investigated in the two conducted pilots. Summarized below are the aims of the work carried out so far:

- Requirements engineering – investigate the support of the project tools and methods in improving requirements elicitation and management.
- Modeling approach – abstractions, managing complexity.
- Suitability – analyze (sub)systems particularly suitable to be modeled with Event-B.
- Getting to know how to model typical architectures.

The purpose of this deliverable is to report on the pilot deployment activities in WP3. This document is structured in the following way:

- Section 2 describes the two systems that have been modelled.
- Section 3 presents how the two systems have been modelled
- Section 4 presents statistics on the two pilot Event-B models.
- Section 5 discusses lessons learnt including feedback to methods and tools.
- Section 6 summarises the conclusions from the pilot deployment effort and discusses the enhanced deployment.

1.1 *Goals of Pilot Developments*

The method for evaluating the above stated goals was to perform two pilot developments. The first pilot development, a model of the BepiColombo MIXS/SIXS software, focused on analyzing basic issues related to using Event-B and the Rodin Platform in the space sector. In the end, the first pilot covered requirements engineering, general tooling issues, and partially suitability and architectural modeling. Findings from the first pilot are reported in Section 5.

Since it was felt that modeling the first pilot in more detail would not yield further insight a second pilot, a generic model of an Attitude & Orbit Control System, was defined. The second pilot allows evaluation of how well generic control system architectures can be modeled. In addition, since a detailed code level specification was available, it was also possible to evaluate how well Event-B and Rodin Platform could cope with the level of detail required for an actual implementation and eventually code generation. The second pilot also demonstrates the difficulty of handling complex control systems with many

Deploy deliverable D20 – Pilot Deployment in the Space Sector

non-deterministic inputs from the environment.

1.2 References

1.2.1 Applicable Documents

- [AD1] Information Society Commission of the European Communities, Information Media Directorate-General, and Communication Technologies. *Seventh Framework Programme Grant Agreement No 214158, Industrial Deployment of Advanced System Engineering Methods for High Productivity and Dependability. Collaborative Project*. December 2007.
- [AD2] Grant Agreement [AD1] *Annex I – Description of Work*. Approved by EC in November 2007.
- [AD3] DEPLOY Project. *Industrial Deployment of Advanced System Engineering Methods for High Productivity and Dependability. Consortium Agreement*. May 2008.
- [AD4] DEPLOY Project. *DEPLOY Deliverable D18 – D12.3 Project Refocus – Amended Description of Work*. August 2009.

1.2.2 Reference Documents

- [RD1] DEPLOY Project. *DEPLOY Deliverable D5 JD1: Report on Knowledge Transfer*. <http://www.deploy-project.eu/pdf/fv-d5-jd1-reportonknowledgetransfer.zip>, January 2009.
- [RD2] DEPLOY Project. *DEPLOY Deliverable D7 D11.1: Measurement Methodology Guide*. <http://www.deploy-project.eu/pdf/d7-revised-final.pdf>, August 2009.
- [RD3] ECSS Secretariat, ESA-ESTEC, Requirements & Standards Division. *Space Engineering: Ground Systems and Operations – Telemetry and Telecommand Packet Utilization (ECSS-E-70-41A)*. <http://www.ecss.nl/>, January 2003.
- [RD4] ESA Media Center, Space Science. *Factsheet: BepiColombo*. http://www.esa.int/esaSC/SEMNEM3MDAF_0_spk.html, January 2008.
- [RD5] Event-B and Rodin Platform Documentation Wiki. *Induction Proof*. http://wiki.event-b.org/index.php/Induction_proof, October 2008.
- [RD6] Event-B and Rodin Platform Documentation Wiki. *Modularisation Plug-in*. http://wiki.event-b.org/index.php/Modularisation_Plug-in, October 2009.
- [RD7] Alexei Iliasov, Elena Troubitsyna, Linas Laibinis, Alexander Romanovsky, Kimmo Varpaaniemi, Dubravka Ilić, and Timo Latvala. *Supporting Reuse in Event-B Development: Modularisation Approach*. Technical Report 947, TUCS (Turku Centre for Computer Science), <http://tucs.fi/publications/insight.php?id=tIITrLaRoVaIIIa09a>, June 2009.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

- [RD8] Alexei Iliasov, Elena Troubitsyna, Linas Laibinis, Alexander Romanovsky, Kimmo Varpaaniemi, Dubravka Ilić, and Timo Latvala. *Supporting Reuse in Event-B Development: Modularisation Approach*.
To appear in the proceedings (a volume in Springer-Verlag’s Lecture Notes in Computer Science series) of the ABZ 2010 Conference (on Abstract State Machines (ASM), Alloy, B and Z) that is to be held in Orford, Québec, Canada, in February 2010. See <http://deploy-eprints.ecs.soton.ac.uk/159/> for a draft of the paper.
- [RD9] Dubravka Ilić and Kimmo Varpaaniemi. *Event-B Project BepiColombo_Models_v5.0*.
<http://deploy-eprints.ecs.soton.ac.uk/136/>, May 2009.
- [RD10] Linas Laibinis and Elena Troubitsyna. *Event-B Project Modes_v2*.
<http://deploy-eprints.ecs.soton.ac.uk/165/>, December 2009.
- [RD11] Pauli Väisänen and Kimmo Varpaaniemi. *DEPLOY Satellite (an Attitude and Orbit Control System) Specification, Version 15*.
<http://deploy-eprints.ecs.soton.ac.uk/167/>, January 2010.
- [RD12] Kimmo Varpaaniemi. *Event-B Project DepSatSpec015Model000*.
<http://deploy-eprints.ecs.soton.ac.uk/168/>, January 2010.
- [RD13] Kimmo Varpaaniemi. *Rodin Platform Bug Tracker Item 2886026*.
http://sourceforge.net/tracker/?func=detail&aid=2886026&group_id=108850&at_id=651669, October 2009.

1.3 Abbreviations

Aabo	Åbo Akademi University
API	Application Programming Interface
APID	Application Process Identifier
ASRO	Aboa Space Research Oy
ASW	Application Software
BC	BepiColombo
CPU	Central Processing Unit
CSW	Core Software
Det. TC	Determined by Telecommand
DPU	Data Processing Unit

Deploy deliverable D20 – Pilot Deployment in the Space Sector

DSP	Digital Signal Processing
ECSS	European Cooperation for Space Standardization
ESA	European Space Agency
ESTEC	European Space Research and Technology Centre
FPA	Focal Plane Assembly
FPGA	Field-Programmable Gate Array
FID	Failure Identifier
FDIR	Fault Detection, Isolation and Recovery
FMI	Finnish Meteorological Institute
HW	Hardware
ID	Identifier
ISAS	Institute of Space and Astronautical Science
JAXA	Japan Aerospace Exploration Agency
LVDS	Low Voltage Differential Signaling
MIXS	Mercury Imaging X-ray Spectrometer
MIXS-C	MIXS Collimator
MIXS-T	MIXS Telescope
MMO	Mercury Magnetospheric Orbiter
MPO	Mercury Planetary Orbiter
N/A	Not Applicable
NASA	National Aeronautics and Space Administration
OBSW	On-Board Software
OIA	Oxford Instruments Analytical Oy
PSU	Power Supply Unit

Deploy deliverable D20 – Pilot Deployment in the Space Sector

PUS	Packet Utilization Standard
R&D	Research and Development
RTOS	Real-Time Operating System
SIXS	Solar Intensity X-ray and Particle Spectrometer
SIXS-P	SIXS Particle Detector Part
SIXS-X	SIXS X-Ray Detector Part
SSF	Space Systems Finland, Ltd.
SU	Sensor Unit
SW	Software
Sys. TC	System Telecommand
TC	Telecommand
TM	Telemetry
UNEW	University of Newcastle upon Tyne
WP	Work Package

2. Pilot Descriptions

2.1. BepiColombo SIXS/MIXS OBSW

2.1.1. Overview

The main goal of the BepiColombo mission [RD4] is the exploration of the planet Mercury. For the purpose of fulfilling its scientific goals, BepiColombo mission will send two orbiters. One of these is the Mercury Planetary Orbiter (MPO) responsible for carrying remote sensing and radioscience instrumentation. An important part of this element is MIXS/SIXS Data Processing Unit (DPU), which is used to control the power of a specific instrument and its operating states, monitor instrument operation, handle telecommand (TC) and telemetry (TM) communication.

The instruments controlled by MIXS/SIXS DPU are: **Solar Intensity X-ray** and particle Spectrometer (**SIXS**) – which records the radiation from the Sun at the position of the spacecraft, and **Mercury Imaging X-ray Spectrometer (MIXS)** – which records the fluorescent X-rays from the planet surface. Both instruments contain two sensor units: **SIXS-X** (X-ray spectrometer), **SIXS-P** (particle spectrometer), and **MIXS-T** (telescope), **MIXS-C** (collimator).

The MIXS/SIXS On-Board software (OBSW) running on the DPU's CPU consists of five different software components:

- The Core Software (**CSW**);
- **SIXS-P ASW** (Application Software) and **SIXS-X ASW** designate the software component controlling P and X sensor units;
- **MIXS-T ASW** and **MIXS-C ASW** designate the software component controlling T and C focal plain assembly.

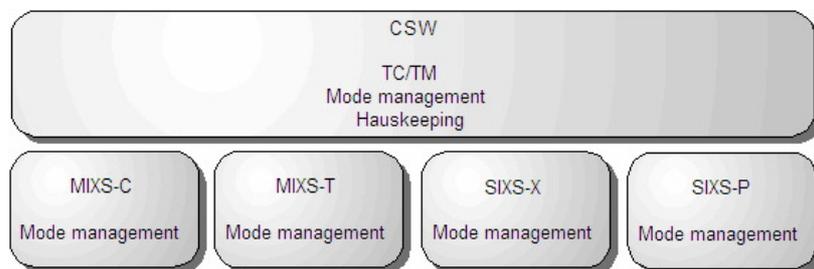


Figure 1 High-level architecture of BepiColombo SIXS/MIXS OBSW

CSW is common interface software for the MIXS ASW and SIXS ASW. It works as a TC/TM interface with the BepiColombo platform as shown in Figure 1, and on the other end, as an interface for the DPU and MIXS/SIXS Sensor Units. The MIXS/SIXS

Deploy deliverable D20 – Pilot Deployment in the Space Sector

instrument SW receives telecommands via a dedicated hardware link. Also, similarly as for TC reception, there is a TM delivery service.

The TC/TM-communication in BepiColombo is based on the ESA's Packet Utilization Standard (PUS) [RD3], even though it is not fully compliant with it. As far as SIXS and MIXS are concerned, telecommands only occur as inputs to the instruments whereas telemetry packets only occur as outputs from the instruments.

2.1.2. BepiColombo SIXS/MIXS OBSW Requirements

As the basis for modeling activities in DEPLOY, only a selected subset of BepiColombo SIXS/MIXS OBSW requirements is considered. These, mostly functional, requirements are focused on a subset of PUS services, some of which are dedicated to specific components, e.g., mode changing TCs for the CSW and instrument software.

For every component, mode commands are executed according to specified *transition diagrams*. Since we focus on modeling the components operating modes and mode transitions triggered either by dedicated telecommands or some failure condition, it is sufficient to present the transition diagrams for all the components without systematically including the actual requirement texts, which occur in the BepiColombo SIXS/MIXS OBSW Software Requirements Specification (that is not a public document).

2.1.2.1. Mode Management Requirements

2.1.2.1.1. CSW

The CSW should implement system-level modes and mode changes presented in the following figure:

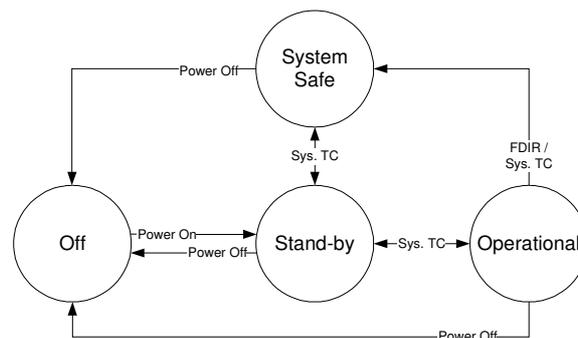


Figure 2 System-level modes and mode changes in BepiColombo SIXS/MIXS OBSW

The system-level mode can be seen as the main mode and the ASW modes as sub-modes of the entire system. The system-level mode can be changed/commanded from the spacecraft platform, though the CSW has to take care that the ASW modules are in modes compatible with the system-level mode.

There are three different system-level modes in addition to the off-mode:

Deploy deliverable D20 – Pilot Deployment in the Space Sector

1. **Stand-by** – all the ASW modules are in their respective Stand-by, Off or Safe modes (they need not to be in the same mode).
2. **Operational** – at least one ASW module is in a mode outside the set {Stand-by, Off, Safe}.
3. **Safe** – all the ASW modules are in their respective Off modes.

Allowed system-level mode transitions are given in the following mode transition table.

To\From	Stand-by	Operational	System Safe
Stand-by	-	TC	TC
Operational	TC	-	FDIR/TC
System Safe	TC	N/A	-

Table 1 System-level mode transitions in BepiColombo SIXS/MIXS OBSW

2.1.2.1.2. SIXS-P ASW

The operating modes and allowed mode changes in the SIXS-P ASW are as follows:

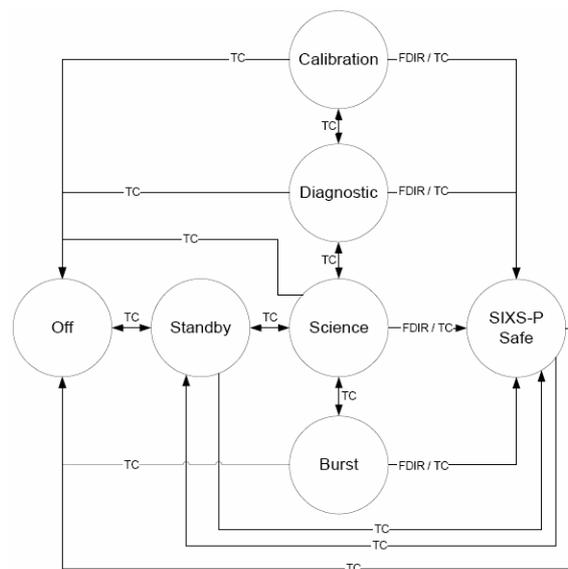


Figure 3 BepiColombo SIXS-P ASW modes and mode changes

The SIXS-P ASW has seven different operating modes:

1. **Off** – The SIXS-P sensor unit is not powered.
2. **Stand-by** – sensor unit is powered on but a detector firmware is not running. Only housekeeping functionality is enabled.
3. **Science** – a detector firmware is running. Science data and housekeeping data are normally transmitted. This is a normal operating mode of the SIXS-P ASW.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

4. **Burst** – a detector firmware is running. Science data is transmitted at an increased rate while housekeeping data is transmitted normally.
5. **Diagnostic** – a detector firmware is running. Raw event samples are transmitted instead of normal science data. Housekeeping data is transmitted normally.
6. **Calibration** – the same as diagnostic mode but calibrator is on.
7. **Safe** – sensor unit is in a safe state that is similar to stand-by. Only a restricted set of TCs (e.g. go to Off and Stand-by mode) are allowed.

The allowed mode changes are textually expressed in the mode transition table below.

From\To	Off	Stand-by	Science	Burst	Diagnostic	Calibration	SIXS-P Safe
Off	-	TC	N/A	N/A	N/A	N/A	N/A
Stand-by	TC	-	TC	N/A	N/A	N/A	TC
Science	TC	TC	-	TC	TC	N/A	FDIR/TC
Burst	TC	N/A	TC	-	N/A	N/A	FDIR/TC
Diagnostic	TC	N/A	TC	N/A	-	TC	FDIR/TC
Calibration	TC	N/A	N/A	N/A	TC	-	FDIR/TC
SIXS-P Safe	TC	TC	N/A	N/A	N/A	N/A	-

Table 2 BepiColombo SIXS-P ASW mode transitions

2.1.2.1.3. SIXS-X ASW

The operating modes and allowed mode changes in the SIXS-X ASW are as follows:

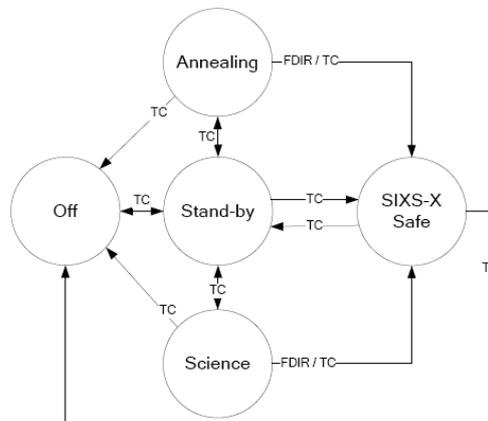


Figure 4 BepiColombo SIXS-X ASW modes and mode changes

The SIXS-X ASW has five different operating modes:

1. **Off** – The SIXS-X sensor unit is not powered.
2. **Stand-by** – sensor unit is powered on but a detector firmware is not running. Only housekeeping functionality is enabled.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

3. **Annealing** – the same as stand-by mode but annealing voltage is on.
4. **Science** – a detector firmware is running. Science data and housekeeping data are normally transmitted. This is a normal operating mode of the SIXS-X ASW.
5. **Safe** – sensor unit is in a safe state that is similar to stand-by. Only a restricted set of TCs (e.g. go to Off and Stand-by mode) are allowed.

The allowed mode changes are textually expressed in the mode transition table below.

From\To	Off	Stand-by	Science	Annealing	SIXS-X Safe
Off	-	TC	N/A	N/A	N/A
Stand-by	TC	-	TC	TC	TC
Science	TC	TC	-	N/A	FDIR/TC
Annealing	TC	TC	N/A	-	FDIR/TC
SIXS-X Safe	TC	TC	N/A	N/A	-

Table 3 BepiColombo SIXS-X ASW mode transitions

For both the SIXS-P ASW and the SIXS-X ASW all modes have a transition to the *Off* mode and all mode transitions are triggered by a telecommand, except for transitions to the *Safe* mode.

2.1.2.1.4. MIXS-T ASW

The operating modes and allowed mode changes in the MIXS-T ASW are as follows:

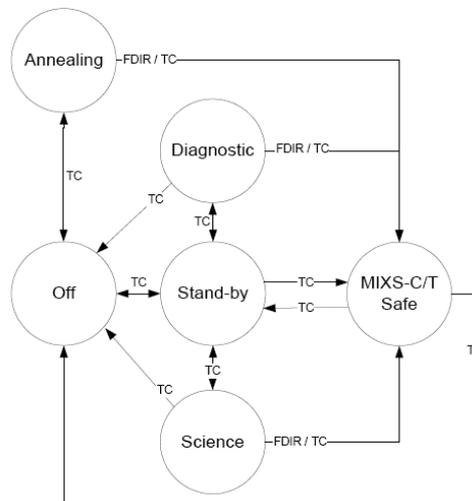


Figure 5 BepiColombo MIXS-T ASW modes and mode changes

The MIXS-T ASW has six different operating modes:

1. **Off** – The MIXS-T FPA is not powered.
2. **Stand-by** – The FPA is powered on and waiting for configuration. No data is

Deploy deliverable D20 – Pilot Deployment in the Space Sector

produced.

3. **Diagnostic** – The FPA is switched on and configured to operate in test mode. Raw image data are put into the telemetry in place of normal science data.
4. **Science** – Science data is normally produced. This is a normal operating mode of the MIXS-T ASW.
5. **Annealing** – The FPA is powered off and annealing voltage is on. No data is produced.
6. **Safe** – The FPA is in a safe state. Only a restricted set of TCs (e.g. go to Off and Stand-by mode) are allowed.

The allowed mode changes are textually expressed in the mode transition table below.

From\To	Off	Stand-by	Science	Diagnostic	Annealing	MIXS-T Safe
Off	-	TC	N/A	N/A	TC	N/A
Stand-by	TC	-	TC	TC	N/A	TC
Science	TC	TC	-	N/A	N/A	TC/FDIR
Diagnostic	TC	TC	N/A	-	N/A	TC/FDIR
Annealing	TC	N/A	N/A	N/A	-	TC/FDIR
MIXS-T Safe	TC	TC	N/A	N/A	N/A	-

Table 4 BepiColombo MIXS-T ASW mode transitions

2.1.2.1.5. MIXS-C ASW

The operating modes and allowed mode changes in the MIXS-C ASW are exactly as in the MIXS-T ASW (see Figure 5 and Table 4).

For both the MIXS-T ASW and the MIXS-C ASW it holds that all modes have a transition to the *Off* mode. Moreover, all mode transitions are triggered by a telecommand, except for transitions to the *Safe* mode.

2.1.2.2. Compatibility between System-Level Modes and ASW Modes

For each system-level mode, the following table expresses the ASW modes that are allowed in that system-level mode. An “X” in a cell indicates that for each ASW module that can have the ASW mode expressed by the title of the column, the system-level mode expressed by the title of the row allows that ASW mode for the module in question. Respectively, a blank cell indicates that for each ASW module that can have the ASW mode expressed by the title of the column, the system-level mode expressed by the title of the row disallows that ASW mode for the module in question.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

System-Level Mode	ASW Mode							
	Off	Stand-by	Diagnostic	Science	Annealing	Burst	Calibration	Safe
Stand-by	X	X						X
Operational	X	X	X	X	X	X	X	X
Safe	X							

Table 5 Allowed ASW modes in BepiColombo SIXS/MIXS OBSW system-level modes

Note: If the spacecraft platform switches off the power from the DPU box, all the instruments will also be powered off.

2.1.2.3. Telecommands and Telemetry Packets

In addition to telecommands handled by the CSW, the ASW modules should handle instrument-specific telecommands. The ASW modules should also collect and store science and housekeeping telemetry data from the instruments. The CSW is responsible for sending the data to the spacecraft platform. The telecommands and telemetry packets that are considered within the scope of the SIXS/MIXS OBSW pilot are given in Table 6.

Service Type	Subtype	Service Request (TC)	Subtype	Service Report (TM)
1			1	TC acceptance report – success
1			2	TC acceptance report – failure
1			7	TC execution report – success
3	5	Enable HK report generation	25	HK report
3	6	Disable HK report generation		
5			1	Event packet – nominal
5			2	Event packet – low severity error/anomaly
5			3	Event packet – medium severity error/anomaly
5			4	Event packet – high severity error/anomaly
6	5	Dump memory using absolute addresses	6	Memory dump using absolute addresses
6	9	Check memory using absolute addresses	10	Memory check using absolute addresses
17	1	Perform connection test	2	Link connection report
21	1	Enable science transfer	3	Science report
21	2	Disable science transfer		
21	128	Reset output buffer		

Table 6 TCs and TMs within the scope of the BepiColombo SIXS/MIXS OBSW pilot

In addition to these, there are also more specific telecommands and telemetry packets (private services) for commanding the SIXS and MIXS instruments.

2.2. AOCS

The purpose of this section is to describe a typical, yet simplified, Attitude & Orbit Control System (AOCS), which is widely used in the context of real satellites.

2.2.1. Overview

AOCS is a common name for a class of control systems whose main function is to control the attitude and the orbit of satellites. The attitude needs to be continuously controlled due to a tendency of a satellite to change its orientation by the disturbance of the environment. The attitude control is based on the information obtained from various sensors and corrective commands realized by actuators. An optimal attitude is also needed to support the needs of payload instruments and to fulfill the mission of the satellite. For example, attitude control may ensure that an optical system of the spacecraft will continuously cover the required area on the ground.

A typical AOCS is shown in Figure 6. Its main components are: the AOCS Manager, the FDIR (Fault Detection, Isolation and Recovery) Manager, the Mode Manager and the Unit Manager.

The *AOCS Manager* is the global orchestrator of the satellite attitude which is continuously controlled by a programmed control loop:

- Sensors measure the satellite's attitude.
- The OBC (On-Board Command) handler then processes these measurements and generates commands for the actuators, in order to ensure correct pointing.

Actuator commands are issued according to the *control algorithm* which is used for the computation.

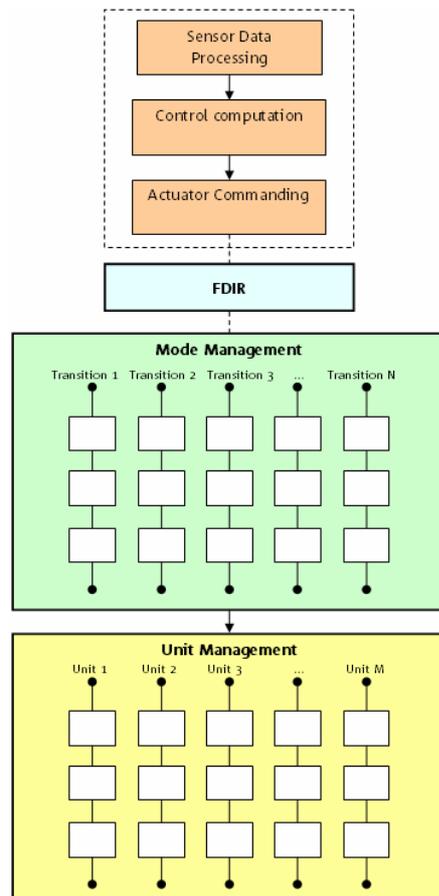


Figure 6 AOCS top level schema

The attitude control is based on control algorithms (controller) which are executed in a

Deploy deliverable D20 – Pilot Deployment in the Space Sector

closed loop where the feedback from the environment is also taken into account. In various mission stages different kind of controllers are required, and usually the different stages are served by specific satellite platform operational *modes*.

The highest level part of the AOCS represents the *Mode Manager*. It executes cyclically and provides different modes for attitude and orbit control of the spacecraft according to the specific spacecraft operational constraints. It is responsible for managing mode transitions and handling specific actions while a certain mode is active in order to supervise control algorithms and mode related *units*. These units are controlled as specified by the *Unit Manager*.

In order to detect failures in AOCS functioning and provide adequate failure recovery actions in response to detection of a failure, the *FDIR (Fault Detection, Isolation and Recovery) Manager* is considered as an important part of the AOCS system. In the execution cycle, the FDIR Manager executes right after controller has commanded the actuators and it can result in mode degradation or reconfiguration.

2.2.2. AOCS Manager

The AOCS Manager is responsible for controlling the overall execution cycle of the AOCS (sub)system, where first data obtained from the various sensors is processed in order for the control algorithm to be able to perform necessary computations and command the engaged actuators. The sensor data is collected only from those sensors which are active in the current mode, i.e., which have been previously locked to be used by the controller. Similarly, only locked actuators can actually be commanded. In addition, the AOCS Manager controls the processing of data from the payload which is actively used only in *Preparation* and *Science* modes (see 2.2.3).

2.2.2.1. Control Algorithms

There are two control algorithms (controllers) executed depending on the satellite operational mode. They are used for coarse and fine pointing control. The control algorithms are executed only when a certain operational mode is reached, meaning that in *Off* and *Standby* modes, no controller is executing. Consequently, while the CDMU (Central Data Management Unit) is rebooting, no controller is executing.

When during the transition to *Safe* mode all necessary unit configurations for this mode are completed, then the Mode Manager selects an adequate controller for this mode – *coarse pointing controller* and initiates its preparation for execution. After preparation phase reaches some predefined time limit, in case there are no ongoing unit reconfigurations, the controller is accepted as ready for execution. Similarly, when during the transition to *Nominal* mode all necessary unit configurations for this mode are completed, the Mode Manager selects *fine pointing controller* and initiates its preparation for execution, which starts after preparation phase reaches some predefined time limit, unless there is an ongoing unit reconfiguration.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

2.2.3. Mode Manager

A satellite can be in various operational modes, which determine its behavior. This behavior is mainly controlled by the Mode Manager, which is also responsible for execution of mode transitions.

The controlled modes are the following:

- *Off* – The satellite is typically in this mode right after CDMU boot.
- *Standby* – This mode is maintained until separation from the launcher is completed.
- *Safe* – Satellite is in this mode when the separation from the launcher is done. In the beginning the satellite acquires a stable attitude, and then it tries to preserve this coarse pointing mode.
- *Nominal* – In this mode the satellite has already achieved the coarse pointing and is trying to reach the fine pointing control which is needed in order to use the payload instrument.
- *Preparation* – In this mode the fine pointing control is reached and payload is getting ready for fulfilling its purpose.
- *Science* – The payload is in this mode ready to perform the tasks for which it was designed. The mission's goal is to reach this mode and stay in it as long as needed.

2.2.4. Unit Manager

The Unit Manager controls the internal state changes of seven different units. The Unit Manager is a low-level component commanded by the Mode Manager and by the FDIR Manager. Among the controlled units, there are 4 sensors, 2 actuators and 1 payload instrument.

2.2.4.1. Sensors

The following 4 sensors provide the cyclic input data for the AOCS software:

- Star tracker (STR) – is an optical device that measures the position of stars in its field of view and performs pattern recognition on these stars in order to identify the portion of the sky at which it is looking. The only possible STR operational states are *On* and *Off*.
- Sun Sensor (SS) – is a device that measures the direction to the Sun in the sensor's field of view. Its possible states are *On* and *Off*.
- Earth Sensor (ES) – is a device that measures the direction to the Earth in the sensor's field of view. ES internal states are *On* and *Off*.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

- Global Positioning System (GPS) – a GPS receiver is a device used to determine the position and provide initial measurements for the calculation of the satellite attitude. GPS can operate in one of two possible states: *Coarse_Navigation* and *Fine_Navigation*.

2.2.4.2. Actuators

AOCS software controls the following 2 actuators via sending specific control commands:

- Reaction Wheel (RW) – is a rotating wheel that is used to apply a torque to the satellite. This is achieved by accelerating or breaking the wheel. This actuator can be either *On* or *Off*.
- Thruster (THR) – is a position actuator which is by emitting gas jets used to force the satellite to change its position and its orbit. THR can be either *On* or *Off*.

2.2.4.3. Payload

The Payload Instrument (PLI) is an instrument which provides specific mission measurements. It can operate in one of two states: *Standby* or *Science*.

2.2.4.4. Unit Status

In order to avoid conflicts in unit usage and to handle units which are used during reconfiguration, each unit always has a unique status: either it is *free*, *reserved* or *locked*. A unit (sensor, actuator or payload) is *free* when it is not used for control in a certain mode. Before a unit is used by the controller, it has to be switched on and prepared for use. During this time unit status is *reserved* to indicate that unit is going to be used but it is not yet ready. Unit becomes *locked* when it reaches the fully operational state, and then it is exclusively used by the closed loop control algorithm, or in case of a payload instrument, unit is locked for its intended operational purpose (see Figure 7).

Each “logical” unit actually consists of two hardware units, nominal and redundant. The redundant unit serves as a backup resource in case that the nominal unit fails. When an error is detected in the nominal unit, the unit becomes “reconfigured”, meaning that the nominal unit is switched off and the redundant unit is taken into use.

During a reconfiguration, a unit cannot be locked, since the procedure to switch Off and On takes time. Hence the reconfigured unit has status *reserved* indicating that the redundant unit is not yet ready.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

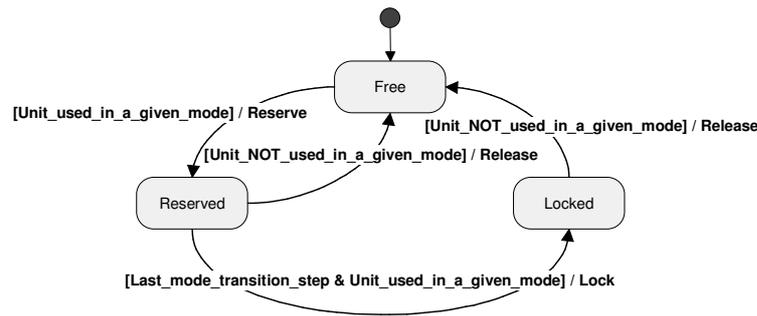


Figure 7 AOCs unit status change diagram

2.2.5. Mode Transitions

The Mode Manager controls the AOCs operation as specified by its modes and transitions between them. Each mode transition is described as a set of specific actions which should be performed in order for the mode to get changed. Transitions between modes are either *autonomous* (i.e., decided by the Mode Manager) or *commanded* by the FDIR Manager after an unrecoverable failure. In case of autonomous transitions, each transition criteria associated to a certain mode is cyclically checked in the active mode. If the condition for mode transition is satisfied, then the mode transition is autonomously performed by the supervision of the Mode Manager.

The following sections specify the *nominal mode transitions*, i.e., autonomous mode transition without any interference of the FDIR Manager.

2.2.5.1.1. Off to Standby

Transition from *Off* to *Standby* is an autonomous transition. As soon as CDMU boots, mode change is forced from *Off* to *Standby*.

This transition is performed in two steps:

- Since no controller is executing at the time, first all units are released and taken to their respective *Off* states.
- When all the units reach their *Off* states, the transition to *Standby* mode is done and the Mode Manager starts to monitor the completion of the separation from the launcher.

2.2.5.1.2. Standby to Safe

As soon as the separation is completed, the mode changes autonomously from *Standby* to *Safe* according to four steps as follows:

- First the payload data processing is disabled and all the units are released and taken to their respective *Off* states.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

- When all the units reach their *Off* states, specific units used in *Safe* mode are brought to their required operational states. The units are configured according to Table 7 . This means that only Earth and Sun Sensors together with Reaction Wheels are actively used in *Safe* mode, and hence these sensors are switched on.
- After the necessary units are switched on, the coarse pointing control algorithm is activated.
- When the selected controller is ready for execution, the used units are locked by the Unit Manager, the transition to *Safe* mode is completed, and both the sensor data processing and actuator commanding are started.

Mode Unit	Off	Standby	Safe	Nominal	Preparation	Science
ES	Off	Off	On	Off	Off	Off
SS	Off	Off	On	Off	Off	Off
GPS	Off	Off	Off	Coarse_Navigation	Fine_Navigation	Fine_Navigation
STR	Off	Off	Off	On	On	On
RW	Off	Off	On	On	On	On
THR	Off	Off	Off	On	On	On
PLI	Off	Off	Off	Off	Standby	Science

Table 7 Initial unit states in respective AOCS modes

2.2.5.1.3. *Safe to Nominal*

When a coarse pointing control is reached, an autonomous mode transition is performed with mode changing to *Nominal*. This transition is performed in three steps:

- First the payload data processing is disabled and specific units used in *Nominal* mode are commanded into their corresponding operational states. The units are configured according to Table 7 . Namely, only GPS, STR, RW and THR are actively used in *Nominal* mode, and hence STR, RW and THR are commanded to switch on, whereas GPS is commanded in *Coarse_Navigation* state.
- After all commanded units obtained their desired states, preparatory activities initialize launching of the fine pointing computations.
- When the preparatory activities are done and the selected fine pointing controller is ready for execution, all used units are locked by the Unit Manager and the transition to *Nominal* mode is completed.

2.2.5.1.4. *Nominal to Preparation*

When a fine pointing control is reached, an autonomous mode transition is performed with mode changing to *Preparation*.

This transition takes three steps to complete. The steps are defined as follows:

Deploy deliverable D20 – Pilot Deployment in the Space Sector

- As the first step, payload data processing is disabled.
- GPS and PLI units used in *Preparation* mode are configured into their corresponding states. The units are configured according to Table 7 . (Observe that also STR, RW and THR are actively used in the *Preparation* mode, but these are already in the desired states.) Thus, GPS is commanded in *Fine_Navigation* state, whereas PLI is commanded into *Standby* state.
- When both GPS and PLI have reached the commanded states, PLI is locked (observe that GPS has already been locked in *Nominal* mode). Now PLI can be engaged in necessary data processing and mode transition to *Preparation* thus completes.

2.2.5.1.5. *Preparation to Science*

When a payload is ready to be used for data processing, an autonomous mode transition is performed with mode changing to *Science*. This short transition to *Science* mode consists of only two steps:

- First, PLI is configured according to Table 7 so that it is commanded into *Science* state.
- Once the PLI reached *Science* state as commanded and after fine point control is obtained for at least the predefined time limit, PLI becomes ready to be functional and the transition to *Science* mode is completed.

2.2.6. **FDIR**

The FDIR (Failure Detection, Isolation and Recovery) mechanism in the AOCS framework relates to the functionality which aims at providing dependable AOCS operation. With respect to this, the AOCS FDIR is expected to handle:

- mode transition errors with the highest priority
- control algorithm related errors and
- unit errors with the least priority.

2.2.6.1. *Handling Mode Transition Errors*

At the beginning of each AOCS cycle, after sensor data is processed and actuators are commanded according to the control computations, the FDIR Manager evaluates whether there has been any flagged mode transition error.

A typical mode transition error is *Timeout*. Each mode transition is timed, meaning that if a step of a transition is not completed within a specified time limit, timeout occurs.

In case such an error is reported to the FDIR Manager, all possibly ongoing unit reconfigurations are aborted (units under reconfiguration are released and commanded to

Deploy deliverable D20 – Pilot Deployment in the Space Sector

their respective *Off* states). In case a mode transition error is detected during transitions aiming to *Standby* or *Safe* modes, then the emergency reboot is required and the Mode Manager is requested to initiate transition to *Off* mode. In the *Off* mode the AOCS Manager stops all activities and waits an externally controlled S/W reboot. Otherwise, if a mode transition error occurred during transition to *Nominal*, *Preparation* or *Science* modes, the recovery action is to move back to *Safe* mode.

In case an error is reported while AOCS is still recovering from a previous error, the recovery action depends on the target mode of the last FDIR recovery action. If this mode is *Off*, *Safe* or *Standby*, then the previous FDIR action is discarded and the new recovery action is to move to *Off* mode. Otherwise, the recovery action is to move to *Safe* mode.

2.2.6.2. Handling Attitude Errors

An attitude error may arise during the computations done by the selected controller. In a nominal case, an attitude error is handled on a principle of a mode rollback, according to Table 8 below.

From \ To	Off	Safe	Nominal	Preparation
Standby	✓			
Safe	✓			
Nominal		✓		
Preparation			✓	
Science				✓

Table 8 AOCS rollback rules upon attitude error

However, if an attitude error is detected while the FDIR is already performing some other recovery actions in *Nominal*, *Preparation* or *Science* mode, then attitude error is handled by requesting a transition to *Safe* mode.

2.2.6.3. Handling Unit Errors

Each unit has a redundant unit, operating in cold redundancy. Hence, there are two different scenarios for handling unit errors, depending on the availability of the redundant unit.

Redundancy not available. In case of a unit error when a redundant unit is not available, error handling depends on the type of unit error and the mode in which that error is detected.

In case when a specific PLI unit error – *Loss of Accuracy* – is detected in *Science* mode, the mode downgrades to *Preparation*, in which PLI will be configured to run in *Standby* state.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

Other possible unit errors are: *commanding failure*, *invalid data* or *timeout* error (similarly to mode transitions, each unit state change is timed; if a step of a unit state transition is not completed within a specified time limit, timeout occurs). All these unit errors are handled in the same manner: the mode is downgraded to the mode where the unit is not used according to Table 7 . For instance, in case of a GPS error, the mode is changed to *Safe*, since in *Safe* mode, GPS is not used, i.e., it is in the *Off* state.

Redundancy available. In case of a unit error when a redundant unit is available, error handling is based on a mechanism of *unit reconfiguration* by switching to a redundant unit. The redundant unit has to be commanded in the state in which the nominal failing unit was before the error was detected, as shown on an example of PLI reconfiguration in Figure 8.

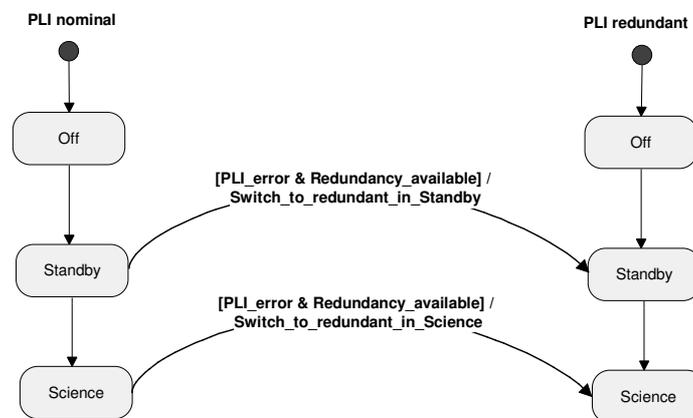


Figure 8 AOCS PLI reconfiguration of redundant unit

2.2.7. A Simplified AOCS by Åbo Akademi

The Åbo Akademi group has attempted an alternative development of an AOCS. The overall goal was to apply classical top-down modeling techniques, relying in particular on the previous experience on developing complex layered systems by refinement. Such techniques allow us to unfold system layers and thus its complexity gradually, while introducing at the same time communication and fault tolerance mechanisms between layers.

In the initial phase, the AOCS presented earlier was intentionally simplified to illustrate such style of modeling. However, the goal remains to gradually reach the described complexity of the AOCS by a number of dedicated refinement steps. The simplified architecture is shown in Figure 9.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

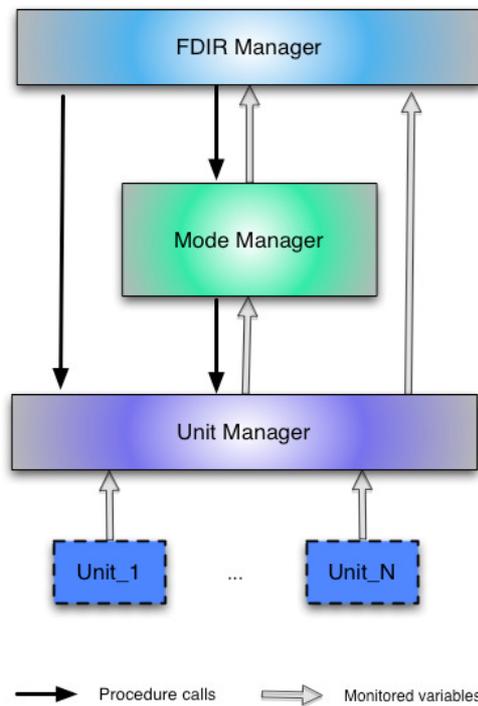


Figure 9 Architecture of a simplified AOCS

As seen from Figure 9, the formal development (in its initial phase) focuses on three system components – the FDIR Manager, the Mode Manager, and the Unit Manager – and their interactions. More specifically, it concentrates on mode management (both global and local) in the presence of system failures. The system tries to reach its most operational state (global mode) by following the pre-defined mode transition scenario. However, system errors may force the system to backtrack to some more degraded mode. The FDIR Manager is responsible for handling such erroneous situations and issuing the corresponding mode transition requests.

At the same time, global mode changes are tightly linked with the required local mode (unit configuration) changes. The consistency between global modes and the required unit configurations (in the presence of various failures) is one of the most important properties to be verified in this development.

3. Technical Descriptions

3.1. *BepiColombo SIXS/MIXS OBSW Requirements Modeling*

Here we concentrate on the Event-B project *BepiColombo_Models_v5.0* [RD9] that is the final version of the model of *BepiColombo SIXS/MIXS OBSW* requirements.

The general modeling strategy was reached after several attempts at different approaches. Finding the right approach clearly required experience with Event-B which SSF did not have at the beginning of the Deploy project. The chosen modeling strategy has the following main features:

- Refinement steps should be small. Every step should focus only on a few issues.
- Complicated behavioral refinement is difficult to use. Superposition and data refinement offer tools for decomposing the problem into steps in most cases.
- The refinements should proceed from a basic producer-consumer system towards a model of the TM/TC handling including mode management.

The machines in *BepiColombo_Models_v5.0* are as follows where every machine except the first one refines the previous machine:

- *obsw_M000* is devoted to TC/TM (telecommand/telemetry) traffic: reception and processing of TCs and production and sending of TMs is modeled on an abstract level. *obsw_M000* has a hardware input buffer for TCs, a hardware output buffer for TMs, a memory pool for TCs and another memory pool for TMs. In *obsw_M000*, TCs arrive in a hardware input buffer from which they are moved to the TC pool in which they stay at least as long as needed. Respectively, TMs are created in the TM pool from which they are either moved to the hardware output buffer or deleted as obsolete. The fact that some TMs are reports on TCs is modeled, too, as well as the needed special operation that atomically resets the hardware output buffer.
- *obsw_M001* is devoted to processing of TCs and TMs. In *obsw_M001*, processing of a TC includes an acceptability check and, in the case of an accepted TC, an execution of the TC. Respectively, processing of a TC includes creation of the TC, a decision on whether to move the TM to the hardware output buffer and, in the case of a positive decision, moving the TM to the buffer. Creation of a TM includes assigning a priority level to the TM and insertion of the TM in a FIFO (first-in-first-out) queue that is dedicated to that priority level. A TM is allowed to be moved to the hardware output buffer only when the TM pool has no higher-priority TM. A TM that is a report on a TC can be either a report on the check of the TC or on the execution of the TC. In *obsw_M001*, TCs are processed in the order they arrive in the hardware input buffer. (Several FIFO queues are used for that purpose.) Respectively, TMs are sent in the order in which they get moved to

Deploy deliverable D20 – Pilot Deployment in the Space Sector

the hardware output buffer, whereas the above-mentioned creation mechanism ensures that TMs on the same priority level are sent in the order of creation. All FIFO queues in obsw_M001 are instances of an abstract data type that has all the needed axioms and theorems in the context obsw_C008. All needed capacity constraints on queues are modeled in obsw_M001 in such a way that no refinement of the constraints is needed.

- obsw_M001continued is really just a continuation of obsw_M001 in the sense that the events are exactly as in obsw_M001. The contribution of obsw_M001continued consists of certain invariants, the handling of which inside obsw_M001 would have been an unnecessary challenge to Rodin Platform.
- obsw_M002 is devoted to TC/TM types. In obsw_M002, the execution of a TC depends on the type of the TC, whereas type-specific constraints are associated with TM creation. A TC without a valid type is not accepted for execution. Every created TM has an appropriate type.
- obsw_M003 is devoted to TC/TM pids (process identifiers). In obsw_M003, every TC has a pid that indicates the software module that is supposed to execute the TC. Respectively, every TM has a pid that indicates the software module that has created the TM. A TC without a valid pid is not accepted for execution. Every created TM has an appropriate pid.
- obsw_M004 is devoted to operational modes in the software modules. All possible mode change mechanisms are represented in obsw_M004. The allowed mode transitions are defined in the context obsw_C004.
- obsw_M005 is devoted to determinism in generation of TC check reports and TC execution reports. In higher-level machines, generation of such reports is abstractly optional. In obsw_M005, a valid reason is always needed for generating a report as well as for not generating a report.
- obsw_M006 is devoted to progress reports and to selected SIXS-X TCs. Progress reports are TMs with the type 5 and subtype 1. In obsw_M006, a progress report is generated whenever an operational mode changes in a software module and whenever the identity of the “current detector” of the SIXS-X instrument changes. Progress reports for other purposes are generated arbitrarily. obsw_M006 pays particular attention to execution of certain TCs that are dedicated to the SIXS-X instrument.
- obsw_M007 is devoted to “reliable” TCs and their “response” TMs as well as to error and anomaly reports. The “reliable” TCs are memory dump requests (type 6, subtype 5), memory check requests (type 6, subtype 9) and connection tests (type 17, subtype 1). The “response” TMs are memory dump reports (type 6, subtype 6), memory check reports (type 6, subtype 10) and connection responses (type 17, subtype 2). In obsw_M007, every successful execution of a reliable TC induces generation of the corresponding response TM. Moreover, obsw_M007 explicitly models the fact that a single memory dump request can induce many

Deploy deliverable D20 – Pilot Deployment in the Space Sector

memory dump reports and that the number of the needed memory dump reports then depends on the request. The error and anomaly reports include reports of severity “low” (type 5, subtype 2), “medium” (type 5, subtype 3) and “high” (type 5, subtype 4).

BepiColombo_Models_v5.0 is a model for a minority of BepiColombo SIXS/MIXS OBSW requirements only but still covers much of the essence of the requirements that concern TCs, TMs and mode management.

3.2. AOCS Modeling by SSF

The used AOCS specification [RD11] is an executable formal specification for “DEPLOY Satellite” in the form of Ada source code files. The decision to use a programming language as a specification language in the WP3 AOCS pilot was based on the desire to express algorithmic aspects unambiguously and to have an easy way to simulate a specification. Though the above Section 2.2 itself inevitably forms some kind of a specification for the AOCS, the WP3 team prefers to think that Section 2.2 is just a rough description of certain easy-to-describe features of the AOCS.

The specification [RD11] essentially consists of four Ada packages, i.e. AocsMgr, FdirMgr, ModeMgr and UnitMgr that correspond to the AOCS Manager, the FDIR Manager, the Mode Manager and the Unit Manager described in Section 2.2. Each one of these packages has its own Main procedure. For a program produced by a compiler from the source code, a run-time call stack of the program contains always at most one and almost always at least one of those four Main procedures. This property corresponds to the sequential nature of the AOCS cycle described in Section 2.2.

Let us then consider the Event-B project DepSatSpec015Model000 [RD12] that is a model of the specification [RD11]. In the model, essentially every event represents one or more source code statements in such a way that a single execution of the event corresponds to a single execution of one of the represented statements. DepSatSpec015Model000 fully models the non-debugging non-comment source code of [RD11] but contains only a few of the “behavioral invariants” that are discounted in comments and in debugging source code in [RD11]. Most of the discounted “behavioral invariants” are assertions concerning specific execution points. An Event-B proof for such an assertion typically requires auxiliary invariants where execution points are explicitly mentioned.

The machines in DepSatSpec015Model000 are as follows where every machine except the first one refines the previous machine. By a “value binding statement” we mean any assignment statement, function call statement, return statement or parameter-passing procedure call statement.

- ManagersMachine describes the systems execution cycle that is simply an infinite repetition of the call sequence “AocsMgr.Main; FdirMgr.Main; ModeMgr.Main; UnitMgr.Main;”.
- FocusOnVariablesUpdatedByManyMgrs describes every value binding statement

Deploy deliverable D20 – Pilot Deployment in the Space Sector

that is reachable from more than one Main procedure. For each such statement, the precise set of such Main procedures is identified. The action blocks of the describing events are final with respect to all further refinements.

- `FocusOnVariablesUpdatedByAocsMgrOnly` describes every value binding statement that is reachable from `AocsMgr.Main` but not from other Main procedures. The action blocks of the describing events are final with respect to all further refinements.
- `FocusOnVariablesUpdatedByUnitMgrOnly` describes every value binding statement that is reachable from `UnitMgr.Main` but not from other Main procedures. The action blocks of the describing events are final with respect to all further refinements.
- `FocusOnVariablesUpdatedByModeMgrOnly` describes every value binding statement that is reachable from `ModeMgr.Main` but not from other Main procedures. The action blocks of the describing events are final with respect to all further refinements.
- `FocusOnVariablesUpdatedByFdirMgrOnly` describes every value binding statement that is reachable from `FdirMgr.Main` but not from other Main procedures. The action blocks of the describing events are final with respect to all further refinements. Moreover, also the set of variables in the machine is final with respect to all further refinements.
- `FocusOnAocsMgrExecution` provides a complete model of the execution of `AocsMgr.Main`, by means of a detailed description that covers all non-debugging statements reachable from `AocsMgr.Main` and associates a precise control flow to all those statements.
- `FocusOnUnitMgrExecution` provides a complete model of the execution of `UnitMgr.Main`, by means of a detailed description that covers all non-debugging statements reachable from `UnitMgr.Main` and associates a precise control flow to all those statements.
- `FocusOnFdirMgrExecution` provides a complete model of the execution of `FdirMgr.Main`, by means of a detailed description that covers all non-debugging statements reachable from `FdirMgr.Main` and associates a precise control flow to all those statements.
- `FocusOnModeMgrExecution` provides a complete model of the execution of `ModeMgr.Main`, by means of a detailed description that covers all non-debugging statements reachable from `ModeMgr.Main` and associates a precise control flow to all those statements.
- `Invariants000` is devoted to “behavioral invariants” and has exactly the same events as `FocusOnModeMgrExecution`.
- `Invariants001` is devoted to “behavioral invariants” and has exactly the same

Deploy deliverable D20 – Pilot Deployment in the Space Sector

events as FocusOnModeMgrExecution.

- Invariants002 is devoted to “behavioral invariants” and has exactly the same events as FocusOnModeMgrExecution.
- Invariants003 is devoted to “behavioral invariants” and has exactly the same events as FocusOnModeMgrExecution.

As said above, DepSatSpec015Model000 contains only a few of the “behavioral invariants” that are discounted in comments and in debugging source code in [RD11].

3.3. AOCS Modeling by Åbo Akademi

This section presents some technical details of the attempted formal development of an AOCS model at Åbo Akademi. The Event-B project Modes_v2 [RD10], the current model, consists of the initial abstract specification and its two refinements. The model is “monolithic” though the events belonging to different managers can be easily identified. In future refinement steps, decomposition of the system (according to Figure 9) is planned, explicitly introducing communication mechanisms between components in the form of procedure calls and monitored variables.

The initial specification (Global_modes) describes the overall system as a simple state machine in terms of global mode changes. The system is always in some current global mode, while the current mode transition is defined by coupling the current mode with the target mode. The accompanying context component introduces the data types and constants representing global modes as a total order.

The operations in the machine abstractly describe the default scenario in which the system tries to achieve the top (most operational) global mode. The default mode scenario can be interrupted by the FDIR Manager, which non-deterministically issues mode transition requests, backtracking the system to the current or one of more degraded modes. The actual error handling and resolution is abstracted away until the second refinement step.

The proved invariant properties relate the ordering of global modes with the current mode transition. For example, the fact that the current mode transition is FDIR-initiated is logically equivalent to the one that the target mode is less or equal (according to the global mode ordering) to the current mode.

The first refinement (Unit_config) introduces a finite number of units that are monitored and controlled by the system. Each unit is in a certain local mode (configuration). The unit configurations should be always consistent with the current mode transition. At the moment when the target mode is reached (i.e., the current mode transition is successfully finished), the configurations of all units should be exactly as required by the target mode.

Similarly as global modes, unit configurations are introduced as a total order. However, as been pointed out during discussions with SSF, a total order is a too strict requirement for unit configurations, while a partial order is a much more realistic assumption for this kind of systems. The necessary modifications will be done for the next version of models.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

The relationship between global modes and the required unit configurations is introduced in the form of an abstract data structure (Mode_configs) in the accompanying context. The essential properties of Mode_configs are formulated as axioms. Mode_configs can be considered as a system parameter that should be supplied (instantiated) at some point of the system development.

One of the most essential properties is order preservation between global modes and unit configurations. In other words, it means that “increasing” (i.e., moving to a more operational) the current global mode corresponds to “increasing” or at least leaving the same unit configurations. There are a few rare exceptions when this property does not hold in the AOCS system. However, while taking these exceptions into account, this property can be considered as one of inherent properties of such types of systems.

In the refined model, unit configuration transitions are modeled in a very similar way as global mode transitions. Additionally, it is formally specified how global mode transition requests lead to generation of new unit configuration requests to keep the system in the consistent state (according to the given abstract function Mode_configs).

Additionally, this refinement step also models unit reconfiguration (i.e., switching, if possible, to an available redundant unit). Unit reconfiguration can be requested by the FDIR Manager in certain situations as a special error recovery procedure. In the refined model, such a FDIR request corresponds to one additional unit configuration step, where the main unit is switched to its spare, which is in its initial configuration Off. Then the spare unit will continue its operation by trying to reach its target configuration, defined by the target global mode.

The refinement step adds a number new event operations, essentially modeling the Unit Manager that (in a unit loop) monitors the units, detects their configuration changes, or sets new configuration requests.

One of the proved invariant properties essentially states the consistency requirement between global modes and unit configurations. Specifically, for all units, if a unit is not currently handling a FDIR request then its configuration is at least as good as required by the current global mode.

The second refinement (Error_handling) elaborates on the FDIR Manager, adding details about its error handling procedures. In the current version, two kinds of errors are handled by the FDIR Manager, namely, mode transition errors and unit errors. As a result of error handling, the FDIR Manager can send separate requests to the Mode Manager (requesting new mode transitions) or to the Unit Manager (requesting reconfiguration of some units) or both.

The intelligence of how the FDIR Manager makes its decisions is represented by abstract functions introduced in the accompanying context machine. These evaluation functions model the knowledge of how particular erroneous situations should be handled. The inputs for such functions include the current and target global modes, the failed unit and its error type. Like the abstract function Mode_configs, these abstract functions can be considered as system parameter that should be supplied at some point of the system

Deploy deliverable D20 – Pilot Deployment in the Space Sector

development.

The FDIR Manager handles the occurred errors in a pre-defined loop, considering first a mode transition error, if any, and then all unit errors. Each loop step produces a new estimation of FDIR resulting actions (i.e., the requests to be sent). Such a new estimation is dependant on the previous estimations. Thus the FDIR Manager iteratively re-assesses the situation, considering the occurred errors one by one. The axiom properties for the introduced evaluation functions require that the resulting verdict can only getting “stronger”, i.e., the FDIR decision in case of a particular set of the occurred errors is always more severe (by setting a more degraded mode as the new target) than for any of its subsets.

Future refinement steps would enforce a particular order (control flow) between managers, which should allow us to prove more interesting properties of such systems. At the moment, the control flow is intentionally modeled as very “loose”, permitting different interleavings between the events of the FDIR Manager, of the Mode Manager and of the Unit Manager.

At the same time, the additional refinement steps would allow us to unfold new layers of the AOCS system, modeling, e.g., sensor data processing and actuator commanding.

The main goal, however, is eventually to decompose the modeled system into smaller parts (components) and introduce communication mechanisms (such as procedure calls and monitoring of external variables) between them. That would allow us to better cope with overall complexity of the system. One of ways to achieve this goal that we are going to investigate is using the developed by Newcastle modularization plug-in [RD6], which allows us to decompose the system into components and describe precise interfaces for component interaction.

3.4. Comparison between SSF’s and Åbo Akademi’s AOCS Modeling Approaches

SSF’s AOCS model development stems from the detailed description of the AOCS system provided as Ada code. Its goal is to produce Event-B models as close as possible to the provided Ada description, reflecting both its overall complexity and organizational structure. As a result, the developed models put the Rodin Platform to a serious test by sheer number of variables and events needed to represent the level of detail present in the Ada description, which is a still somewhat simplified version of the real AOCS subsystem. It also highlights the lack of Rodin Platform features that are sorely needed to improve its scalability such as decomposition and modularization support, modeling of procedure calls, control flow management etc.

To faithfully reflect the control flow and organizational structure of the Ada description, many special variables (e.g., program counters or the variables to model formal parameters of called procedures) have to be introduced. This makes difficult to formulate more general properties of the system, for example, that some property which is true at certain point of execution is still true after a number of steps down the road of the modeled control flow.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

The formal development attempted by Åbo follows the standard techniques of top-down modeling by refinement. The provided Ada document is used more like a reference point or general requirements document. The goal of the development is to identify generic specification and refinement steps or patterns as well as general properties that are typical for this class of systems.

The development starts with a much simplified architecture of the AOCS system, which is then unfolded layer by layer by refinement. At the same time, the selected general properties about mode management are formulated and proved. This is possible because the level of complexity represented in models is still manageable.

The current development is still very incomplete. The refinement process will be continued, thus slowly approaching the level of detail that is already covered by SSF models. To avoid explosion of complexity, the use of newly developed decomposition and modularization techniques [RD7], [RD8] (e.g., the modularization plug-in) will be investigated.

4. Measurements

4.1. Pilot on BepiColombo SIXS/MIXS OBSW

4.1.1. Coverage in Requirements Modeling

BepiColombo_Models_v5.0 [RD9] is the final Event-B project in the first conducted pilot (see 2.1) and covers approximately 18% of mainly functional requirements. Detailed numbers below are showing the coverage of requirements with respect to their modeling status:

- 38 requirements (about 7.79%) have the status “fully modeled”,
- 48 requirements (about 9.84%) have the status “partially modeled”, and
- 402 requirements (about 82.38%) have the status “ignored”.

Even though this is only a small percentage of the original requirements, the resulting models are quite complex. Despite this, our experience suggests that the required modeling efforts are not very alarming. The only really time-consuming activity in the pilot development is proofs - a considerable amount of time is needed for producing a proof, even when it is needed only to reuse an existing proof. Hence our interest has moved during the project from quantitative goals such as requirements coverage to qualitative goals such as better understanding of the Event-B proof methodology.

Let us still pay attention to “ignored” requirements. The set of “ignored” requirements is too heterogeneous to have a simple content-based characterization. However, it can be observed that a vast majority of the “ignored” requirements is formed by requirements that do not directly concern telemetry/telecommand handling. The position of telemetry/telecommand handling as an inevitable part of many space software projects partially explains why the modeling effort in first pilot of WP3 has so far almost solely concentrated on requirements that directly concern telemetry/telecommand handling.

The emphasis on telemetry/telecommand handling has also the following pragmatic motivation: among the requirements that do not directly concern telemetry/telecommand handling, certain requirements specify somewhat massive collections of data structures to be handled in the instrument ASW modules of the OBSW. Sufficient modeling of the handling of all such structures would almost inevitably take a very long time.

There is still no good estimate for how long it would take to create an Event-B project that in some useful way would at least partially model every functional requirement and would also be complete with respect to proofs without strikingly compromising the collection of properties concerned in proofs.

4.1.2. Proof Statistics

BepiColombo_Models_v5.0 involves 8 refinement steps and 1000 generated proof

Deploy deliverable D20 – Pilot Deployment in the Space Sector

obligations. All these proof obligations have been discharged. The proof statistics is shown in Table 9. In the leftmost column, every context name has the prefix *obsw_C* whereas every machine name has the prefix *obsw_M*. In Table 9, every context except the first context extends the previous context, whereas every machine except the first machine refines the previous machine.

Context or Machine	Number of Events	Number of Proof Obligations	Number of Automatically Discharged Proof Obligations	Number of Manually Discharged Proof Obligations
obsw_C000	0	0	0	0
obsw_C001	0	0	0	0
obsw_C002	0	0	0	0
obsw_C003	0	0	0	0
obsw_C004	0	80	25	55
obsw_C005	0	0	0	0
obsw_C006	0	2	2	0
obsw_C007	0	7	1	6
obsw_C008	0	64	19	45
obsw_C009	0	2	0	2
obsw_M000	10	16	16	0
obsw_M001	19	181	162	19
obsw_M001continued	19	190	49	141
obsw_M002	33	24	24	0
obsw_M003	40	127	119	8
obsw_M004	44	31	25	6
obsw_M005	44	10	10	0
obsw_M006	57	155	127	28
obsw_M007	63	111	104	7
Total	63	1000	683	317

Table 9 Proof statistics of BepiColombo_Models_v5.0

4.2. Pilot on AOCS

4.2.1. Architectural Coverage

As described in Section 2.2, the AOCS used by SSF has 4 managers: the AOCS Manager, the FDIR Manager, the Mode Manager, and the Unit Manager. If “behavioral invariants” are not concerned, SSF’s Event-B model considered in Section 3.2 fully models the behavior of all the managers.

As described in Section 2.2.7, the AOCS used by Åbo Akademi has 3 managers: the FDIR Manager, the Mode Manager, and the Unit Manager. Åbo Akademi’s Event-B model considered in Section 3.3 models the behavior of all those 3 managers, but certain discounted details are missing as explained in Section 3.3.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

4.2.2. Coverage in Source Code Modeling

In the AOCS pilot, coverage in source code modeling is relevant only for models developed at SSF because only SSF uses source code as a specification.

Here we consider the same source code [RD11] and the same Event-B project DepSatSpec015Model000 [RD12] as in Section 3.2. As said in that section, DepSatSpec015Model000 fully models all non-debugging non-comment parts of the source code. This result holds essentially regardless of how source code coverage is measured.

4.2.3. Proof Statistics

DepSatSpec015Model000 involves 13 refinement steps and 1858 generated proof obligations. All these proof obligations have been discharged. The proof statistics is shown in Table 10 where ManagersContext and DataContext (an extension of ManagersContext) are the only contexts whereas every machine except the first machine refines the previous machine.

Context or Machine	Number of Events	Number of Proof Obligations	Number of Automatically Discharged Proof Obligations	Number of Manually Discharged Proof Obligations
ManagersContext	0	1	1	0
DataContext	0	6	0	6
ManagersMachine	12	13	13	0
FocusOnVariablesUpdatedByManyMgrs	37	32	27	5
FocusOnVariablesUpdatedByAocsMgrOnly	49	3	0	3
FocusOnVariablesUpdatedByUnitMgrOnly	68	20	3	17
FocusOnVariablesUpdatedByModeMgrOnly	119	22	20	2
FocusOnVariablesUpdatedByFdirMgrOnly	131	9	6	3
FocusOnAocsMgrExecution	148	762	633	129
FocusOnUnitMgrExecution	177	30	0	30
FocusOnFdirMgrExecution	203	19	0	19
FocusOnModeMgrExecution	230	19	0	19
Invariants000	230	230	170	60
Invariants001	230	230	121	109
Invariants002	230	230	158	72
Invariants003	230	232	157	75
Total	230	1858	1309	549

Table 10 Proof statistics of DepSatSpec015Model000

Åbo Akademi's latest AOCS model, the Event-B project Modes_v2 [RD10], involves 2 refinement steps and 121 generated proof obligations. All these proof obligations have

Deploy deliverable D20 – Pilot Deployment in the Space Sector

been discharged. The proof statistics is shown in Table 11 where every context name has the suffix _ctx. In Table 11, every context except the first context extends the previous context, whereas every machine except the first machine refines the previous machine.

Context or Machine	Number of Events	Number of Proof Obligations	Number of Automatically Discharged Proof Obligations	Number of Manually Discharged Proof Obligations
Global_modes_ctx	0	3	3	0
Unit_config_ctx	0	3	2	1
Error_handling	0	3	3	0
Global_modes	6	25	21	4
Unit_config	10	42	38	4
Error_handling	14	45	41	4
Total	14	121	108	13

Table 11 Proof statistics of Modes_v2

5. Lessons Learnt

5.1. Assessment of Suitability of Event-B

Event-B is essentially a guarded command style language where events and guards can be specified using a simply typed first-order logic. Since the language is event-based, it is eminently suited for modeling behavior that can be captured well with state machines. Capturing algorithmic computation requires more effort and is in some cases even impossible since real numbers are not currently supported by the Event-B type system.

An Event-B model has two parts: the context and the machine. Roughly speaking the context captures static definitions such as types while the machine models the behavior. Refinement can be used to split the modeling into several successive steps where each model provably is a behavioral refinement of the more abstract machines. This allows for some limited management of model complexity, but experience has shown that it mostly functions as an aid to manage proof complexity.

Type system. The type system in Event-B is inadequate:

- Event-B has no built-in support for record types. Though any record type can be simulated by means of functions and/or Cartesian products, such a simulation tends to complicate proofs. So, Event-B currently encourages “splitting” of records and then proceeding as if records had never existed.
- Event-B’s support to enumerated types is insufficient though has been considerably improved in 2009. (Before the introduction of the partition operator to Event-B in 2009, it was very uncomfortable to express that differently named elements are different indeed. In the bug report [RD13] it is still shown how a fatal non-response situation in Rodin Platform gets caused by a partition expression used for defining an enumerated type of 80 elements.) In particular, there is no built-in support for the enumeration associated with an enumerated type. One may argue that it is a poor practice to be dependent on numeric values or on a total or partial order. However, as long as Event-B does not provide operators for non-numeric basic types, users of Event-B are essentially encouraged to use numeric types instead.
- Event-B has no built-in support for real or rational numbers. Real numbers are compatible enough with the Event-B proof methodology, basically for the same reason why “non-Event-B” algebraic tools have for decades been used for handling real numbers without rounding. On the other hand, rounding errors are conceptually close to overflow errors that are already within the scope of Event-B. (When an expression contains constant integer numbers, an Event-B prover may do something that can be called a “concrete calculation”. An overflow may occur in such a calculation.)
- Event-B has no built-in support for abstract data types. Arbitrarily much time may

Deploy deliverable D20 – Pilot Deployment in the Space Sector

get spent on “mathematical foundations of Event-B” when abstract data types are defined from scratch. As mentioned in Section 3.1, an abstract queue data type was defined from scratch for the first pilot in WP3. As a somewhat necessary prerequisite for inductive proofs on the type, a fundamental induction principle on natural numbers was expressed as a theorem in the context `obsw_C008` in the Event-B project `BepiColombo_Models_v5.0`. The theorem is as recommended [RD5] by the Event-B and Rodin Platform Documentation Wiki but was still challenging to prove.

Modeling effort. Our current experience suggests that a trained person can construct a medium complexity model in a few weeks. However, spending a few weeks on writing an Event-B model tends to be more frustrating than spending the same amount of time on writing a program in high-level programming language and is likely about as frustrating as spending the same amount of time on writing a program in assembly language. One explanation is that any high-level programming language inherently provides more “interesting variation” than Event-B or any assembly language.

Model readability. Event-B does not include any way to decompose the model. In addition the pretty printed textual notation has several cryptic syntactic elements that present a clear risk for understandability.

Modularity. Lack of modularity is not only a readability problem but also a source of arbitrarily much work that could be avoided with proper utilization of modularity. Encouraged by the experience in WP3, the DEPLOY people at University of Newcastle and Åbo Akademi have designed a modular extension of Event-B and an associated plugin tool [RD6][RD7][RD8].

Model usefulness. A complete Event-B model presents a formal proof for the correctness of the model invariants. A detailed Event-B model is also comparable to a detailed architectural design, perhaps to some extent even a detailed design. However, mapping an Event-B model to source code is not always trivial, especially since the model does not currently take into account any real-time restrictions. These need to be analyzed from the actual source code architecture. The new work on code generation decided at the project refocus [AD4] will be essential to increase the usefulness of the model.

5.2. Proofs

Rodin Platform is the main tool used for Event-B modeling and proof activities in DEPLOY.

Theorems provide a good way to keep the amount of proof work in Event-B projects as small as possible. Unfortunately, regardless of how well theorems have been chosen, a lot of time in interactive proving tends to get spent in selecting of needed hypotheses and in deselecting of unneeded hypotheses. Rodin Platform tends to be unable to draw any obvious conclusion when there are many selected hypotheses.

It is apparent that the scope of automated proofs in Rodin Platform can be considerably

Deploy deliverable D20 – Pilot Deployment in the Space Sector

extended by means of straightforward improvements in the algorithms for automated proving. Respectively, the amount of human work in interactive proving in Rodin Platform can be considerably reduced by means of straightforward improvements in the algorithms for interactive proving.

Work in interactive proving can also be reduced by reusing work done in old proofs. The possibility to copy and paste a proof tree is a way for such reuse. Unfortunately, pasting a tree is not necessarily very helpful when the hypotheses and the goal in the new proof are not exactly the same as in the old proof. Rodin Platform should apparently have a direct way to modify an existing proof for a new purpose.

There is yet another way to reduce the amount of user hours spent in proving in Rodin Platform: it should be possible to non-interactively execute a sequence of actions that a user currently can execute during interactive proving only. Namely, a single sequence of actions is often usable in many proofs, whereas a user typically recognizes having used the same sequence over and over again. On the other hand, the facility for interactive proving in Rodin Platform tends to react so slowly to user actions that a user often spends much more time in waiting than in “productive work”. Successful systematic application of non-interactive executions of sequences would likely involve iterations, so the approach would be most beneficial in cases where the time spent in revising the sequences would be predictably less than the time needed for doing everything interactively.

The efficiency of Rodin Platform needs to be improved in many ways. One of the problems observed in WP3 is that Rodin Platform consumes memory in a very careless way. When BepiColombo_Models_v5.0, the final Event-B project for the first pilot, was created in a computer with a 1 GB RAM, Rodin Platform consumed so much (inherently slow-to-use) virtual memory that interactive proving was “absurdly slow”. Some of the workspace file sizes displayed below “reflect” though do not necessarily justify the observed memory consumption. Some experimental evidence suggests that a user could regulate memory consumption in Rodin Platform by regulating the sizes of the .bpr files, that is to say, by having as many contexts and machines as needed in order to ensure that no single context or machine has “too much proof information”. Unfortunately, that way does not work against cumulative memory consumption that is currently in use in certain functionalities in Rodin Platform.

The Event-B proof methodology in Rodin Platform questionably ignores the problem of inconsistent combinations of axioms and/or guards. In cases of inconsistency, even a very careful user may produce arbitrarily many proofs without noticing the inconsistency, as it is not necessarily likely for an inconsistent combination of axioms and/or guards to get used in a single proof. Rodin Platform should definitely have a proper interface for proving consistency among axioms and guards.

Scalability. The above-mentioned memory problems in the first pilot indicate a serious scalability problem in Rodin Platform because WP3 sees a need for much more detailed Event-B machines and for much larger proof obligation collections than those encountered in the first pilot.

5.3. *Model Checking*

Certain Event-B properties are currently almost beyond the scope of the Event-B proof methodology. One of such properties is deadlock-freedom. Deadlocks are reachable states where no event is enabled. An event is enabled when and only when all the guards and “guard-like actions” of the event are simultaneously satisfiable. From this definition it follows that deadlock-freedom is expressible as an invariant. However, mere expressing of such an invariant is somewhat uncomfortable, whereas not many people can be expected to successfully use Rodin Platform for producing all proofs needed for proving such an invariant as a whole for a given nontrivial Event-B machine.

Deadlock-freedom is a somewhat naïve property in the sense that deadlock-freedom can be artificially ensured by including a totally unconstrained event in the concerned Event-B machine. LTL can be used for expressing more “advanced” properties. Any invariant in any Event-B machine is easily expressible in LTL. So, deadlock-freedom is expressible in LTL, too, though LTL does not have any built-in way for compact expressing of deadlock-freedom.

For a given Event-B machine, the animator and model checker ProB (as a standalone tool or as a plugin of Rodin Platform) can be used for checking deadlock-freedom (without any formulation effort) or an invariant (without any additional formulation effort) or a property expressed in LTL. Whenever a check gets completed with a negative result, a counterexample in the form of a sequence of event occurrences is obtained and can be inspected. Positive results reported by such checks are almost inevitably incomplete because a typical check concerns only some part of the reachability graph of the Event-B machine and because even when ProB’s advanced heuristics such as the symmetry method are used, the reachability graph can be far too large to be completely checked using ProB “within reasonable time” and without exhausting the available memory. For convenience, ProB has a configurable timeout parameter such that when the limit expressed by the parameter is reached, ProB terminates the check with a timeout message.

The animation and model checking facilities in ProB are based on enumeration of reachable states and on simple evaluation of guards and actions. Therefore, for each constant or abstract set in the Event-B contexts directly or indirectly seen by the Event-B machine of interest, ProB replaces the constant or set with an “enumerative expression” in a way compatible with all axioms expressed in those contexts and with all “ProB configuration axioms”. Any successful instantiation of all needed constants and sets actually forms a proof of consistency among all the axioms. Respectively, any fully justified report about any strictly impossible instantiation forms a proof of inconsistency among the axioms. From the theory of computability and computational complexity we know that there is no polynomial-time algorithm for checking whether a collection of axioms is consistent. So, it is not a surprise that a run of ProB very typically ends with a timeout message without managing to instantiate the needed constants and sets.

In the first BepiColombo-specific pilot, many attempts to use ProB on the machines of the Event-B project have failed due to the impossibility to instantiate the needed axioms and sets “within reasonable time”.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

The effective scope of ProB could be extended by adding a symbolic model checking facility to ProB, e.g. by means of a translation to some reputed symbolic model checking tool. While waiting for such a facility, WP3 could try to fit its models in the effective scope of ProB by means of abstractions. Unfortunately, much abstraction work might then get done without any significant effect on the use of ProB.

Partially due to the problems considered in the above paragraphs, model checking and ProB have an intentionally low priority in WP3.

5.4. Requirements Modeling

DEPLOY has been beneficial for the BepiColombo SIXS/MIXS OBSW project only in the form of comments that have resulted from inspection of the OBSW requirements. None of the comments has been inspired by any modeling or proof activity. The situation would likely have been the same even if all requirements of the OBSW had been modeled.

So, the first pilot in WP3 has not formed any support to the use of the Event-B methodology in software development projects.

What comes to methodological observations in the first pilot, it suffices to say that the previous three sections are almost solely based on experience in the first pilot.

5.5. AOCS Modeling

The second pilot in WP3, despite of considering a fictitious system completely specified by the modeling team itself, has essentially the same motivational problem as the first pilot. Many design errors have been found in inspection of the Ada source code, whereas the modeling and proof activities have not resulted in any interesting observations about the modeled system.

The following methodological observations have been made as a result of the second pilot:

- It is often more difficult to find the “essential intended meaning” of a piece of source code than the “essential intended meaning” of a verbal software requirement. Therefore, the team for the second pilot very early made the decision to model the “literal meaning” of the source code. The price paid for such a decision is that the length of the Event-B description has basically nothing to do with the complexity of the modeled behavior. The second pilot thus supports the old conjecture that Event-B should in general not be used for modeling of source code.
- Event-B should not be used for detection of bugs that are in a routine manner detected by well-known software tools. For example, basically any compiler is able to detect uninitialized variables whereas e.g. valgrind is specialized to detection of inappropriate memory references.

Deploy deliverable D20 – Pilot Deployment in the Space Sector

- Event-B has not been designed with control flow modeling in mind. Even if statement numbering is “given” as in the second pilot, it is uncomfortable and error-prone to use such numbers in event definitions.
- It is by no means obvious how to prepare Event-B descriptions that would be as detailed and as readable as the source code of a “realistic system”. (Note that the source code in the second pilot is quite abstract when compared to actual space software source code.)
- A property of interest is often easy to express as an “invariant of a module”. This is yet another motivation for modularity in Event-B.
- The most important properties discounted for the second pilot are easily expressible as temporal properties for ProB but practically not expressible as invariants for the Event-B provers. This does not mean that ProB would be able to verify those properties. (Any execution cycle of the considered AOCS can make several nondeterministic choices. None of the model checking methods in ProB is likely powerful enough against such nondeterminism.)
- One of the few benefits in manual modeling of source code is that the modeler can recognize and utilize abstractions that are of long-term interest even if the model itself is of short-term interest only,
- The considered fictitious AOCS is analogous to various control systems, including systems that have nothing to do with any of the industry sectors represented in DEPLOY. Domain-independent ways to specify and model such systems should be found. This is a long-term goal for not only WP3 or DEPLOY.

5.6. Team Work

WP3 has not found proper documentation for a distributed construction of an Event-B project. Consequently, WP3 has so far only used the sequential approach where one person works for some time and then passes an exported Event-B project to a person who continues the construction.

Another team work problem in WP3 concerns the amount of people involved in Event-B work. Only three persons at SSF have done Event-B work for the two pilots, the contribution of one of those persons being limited to model reviewing.

6. Conclusions and Towards Enhanced Deployment

The pilot deployment phase has showed that fairly complex models can be built with a reasonable effort. Different modeling approaches have been tested for control systems of realistic complexity in the space domain.

Several technical challenges related to the platform have been identified that are problematic for the goal of taking Event-B into use at SSF. These will need to be

Deploy deliverable D20 – Pilot Deployment in the Space Sector

addressed during the enhanced deployment phase.

6.1. Towards Enhanced Deployment

Since the pilot deployment phase has identified many development needs for the Rodin Platform and there is a major extension to the platform planned during the project, the enhanced deployment phase should be done in steps so that improvements in the Rodin Platform as much as possible can be evaluated within the scope of the project.

The enhanced deployment will be based on series of case studies and the first goal is to continue work on the AOCS model. Unfortunately, the obtained experience suggests that the approach used in DepSatSpec015Model000 is “not economical enough” to deserve being used in any later Event-B project. Proving invariants without modularity in this setting is very time consuming. In addition, control flow also seems to require too much effort.

The AOCS model has proven to be a very interesting case study and work on it will therefore be continued during enhanced deployment. It is possible that it will even provide enough material for the remainder of the project. Since the goal is to connect the work on enhanced deployment to the tool development, an early priority will be to experiment with the modularity plug-in [RD6][RD7][RD8] using the AOCS model. We foresee that the AOCS model can be extended to accommodate experimentation on other relevant extensions of the Rodin Platform.