INTEGRAL mission - Automation of Transponder Swaps Scheduling and Schedule Validation

Václav Pavlíček[1] and Stefano De Padova[2#]

[1]Imperial College London, United Kingdom
[2]The European Space Agency, Germany
[#]Advisor

ABSTRACT

Human-made satellites allow us to broaden the knowledge of the world by discovering new stars, and exoplanets, observing Earth, performing X-ray measurements and more. All satellites need to be controlled from Earth by the Flight Control Team (FCT) consisting of experts from a wide range of fields. The satellite INTEGRAL is controlled from the astronomy control room in the European Space Operations Centre (ESOC) together with XMM-Newton and Gaia. Automation allows a single person to control three spacecrafts simultaneously and helps to reduce operational costs. This paper describes the automation of transponder swaps scheduling and mission schedule validation using tools developed in the programming language Python. Both tools were carefully tested in the defined verification process and marked as ready to be used. The developed tools allow faster mission planning for the INTEGRAL satellite thus saving the human resources of the FCT.

Introduction

The INTEGRAL (INTErnational Gamma Ray Astrophysics Laboratory) satellite was successfully launched from Baikonur in Kazakhstan on 17 October 2002 at 04:41:00 (UTC) by the PROTON rocket [1]. The initial operational duration was set to 5 years with a possible extension period of 2 years. Due to the mission's success and scientific returns, the mission duration was extended, and the satellite is still operational in 2022. The long operational duration, and near-real-time design of the mission requiring 24/7 operation with significant manual work from the spacecraft controller, raise many challenges. To save costs on spacecraft operations, the flight control teams of XMM-Newton and INTEGRAL were merged in 2008 [2]. In 2018, the spacecraft controller in the Dedicated Control Room (DCR) for the astronomy mission became also responsible for controlling the Gaia satellite [3]. Both of these merges suggest a need to automate as many processes as possible to allow a single person to control all three spacecrafts. In the future, there is a plan to control EUCLID from the astronomy DCR as well leading to further automation. This project aims to automate processes for the INTEGRAL satellite. The first part involved the development of a transponder swap time insertion tool that would optimise transponder swap times for minimal frequency of swaps and impact on the scheduled commanding activities. The second part involved automating the timeline schedule validation for the INTEGRAL satellite.

Mission Operations and Planning

The INTEGRAL Ground Segment is split into operational and scientific parts. The INTEGRAL Mission Operations Centre (MOC) is located at ESOC in Darmstadt (Germany) and INTEGRAL Science Operations Centre (ISOC) is located at ESAC in Madrid (Spain) as well as INTEGRAL Science Data Centre (ISDC) at the University Geneva, Switzerland [4]. Figure 1 shows the diagram of the mission planning process that mainly consists of five steps. Firstly,

the Planning Skeleton File (PSF) containing times of orbital events such as eclipses and ground station visibility is generated by Flight Dynamics System (FDS) at the MOC. Then the ISOC inserts Pointing Requests (PREQ) i.e. the desired attitudes for targets/sources to be observed and instrument configuration parameters, based on the proposals of the scientific community, resulting in the Planned Observation Sequence (POS) file. The POS file is processed by Flight Dynamics System (FDS) with all necessary engineering and attitude control requests creating an Enhanced POS (EPOS) file [5]. The next step includes transponder swap scheduling followed by the execution of Mission Planning checks. Finally, the MOC Mission Planning System (MPS) translates the ISOC and FD requests into the Spacecraft Commands and command parameter values to perform the requested observations, creating the "Timeline". The commands from the Timeline are sent to the spacecraft via the Mission Control System (MCS).

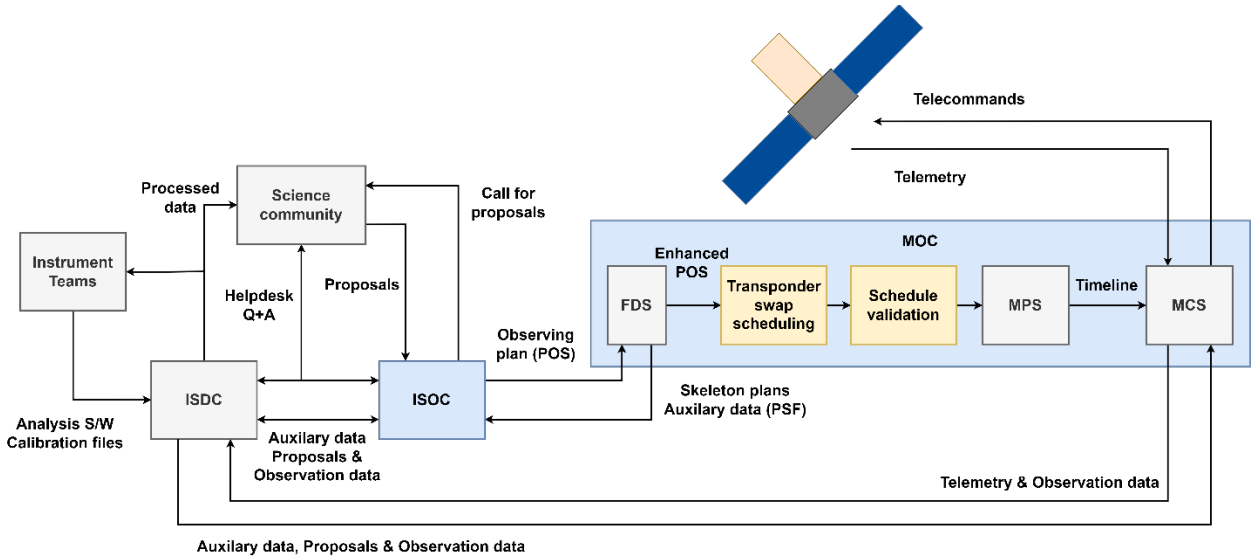

Figure 1. Mission planning for INTEGRAL

Transponder Swap Scheduling

As the spacecraft shown in figure 1a performs its observations, it is not always possible to communicate with the spacecraft using only a single antenna throughout the entire revolution due to the attitude of the spacecraft. Therefore, the spacecraft was mounted with two antennas as figure 1b shows. One antenna (LGA1) is located on the Sun side (0°) of the spacecraft with a 25° tilt from the Z-Y plane and the other antenna (LGA2) is located on the anti-Sun side (180°) of the spacecraft with a -20° tilt.

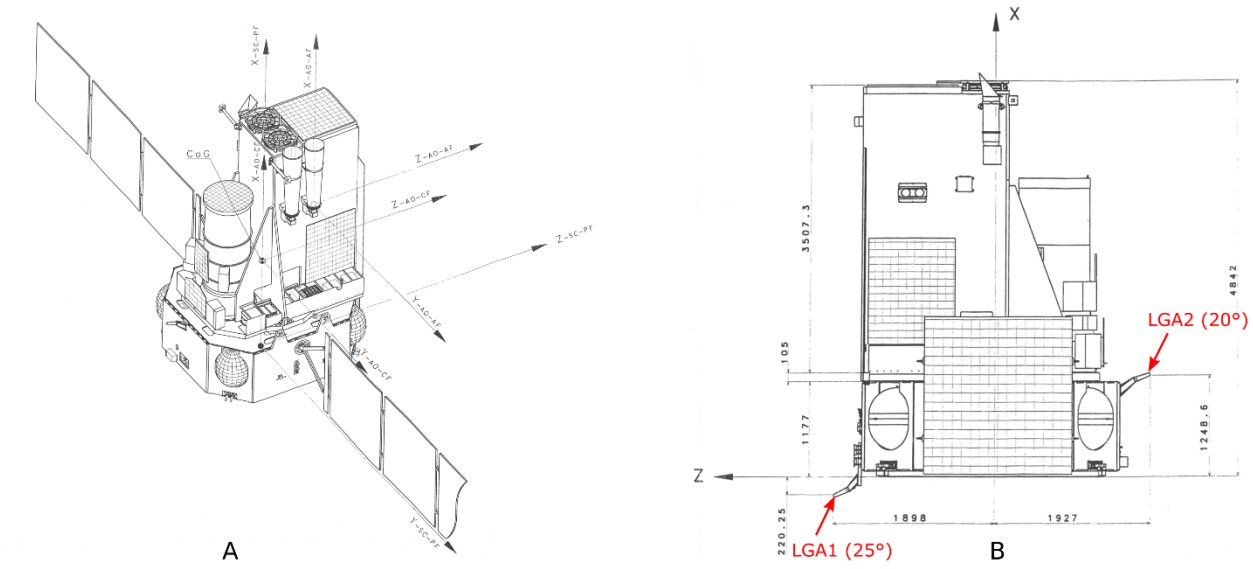

Figure 2. CAD drawings of the INTEGRAL satellite from the manual – overview in figure 2a and side view in figure 2b.

Figure 3a shows the details of the Radio Frequency Distribution Network (RFDN) subsystem that consists of two antennas, two electromechanical switches and two transponders. As figure 3b suggests, the electromechanical switches have two possible configurations - direct and cross. Before the RFDN failure in September 2021, these electromechanical switches were used to swap between the two antennas and allowed the same transponder to be used. This was done by sending a command, triggering the switch of antennas within a few seconds. After the RFDN failure and problems with the electromechanical switches, a different approach to swap between satellite antennas was required. Based on suggestions from industrial partners and experience with XMM RF failure [6], it was decided to swap between transponders with each antenna being hardwired to a different transponder. The swap is performed by turning off the source transponder and turning on the destination one when a change in geometrical visibility occurs. Before the RFDN anomaly, each of the transponders experienced around 1,000 ON/OFF cycles from the qualified 25000 power cycles. To maximise the lifetime of transponders, the requirement to minimize the number of swaps was set for the algorithm.

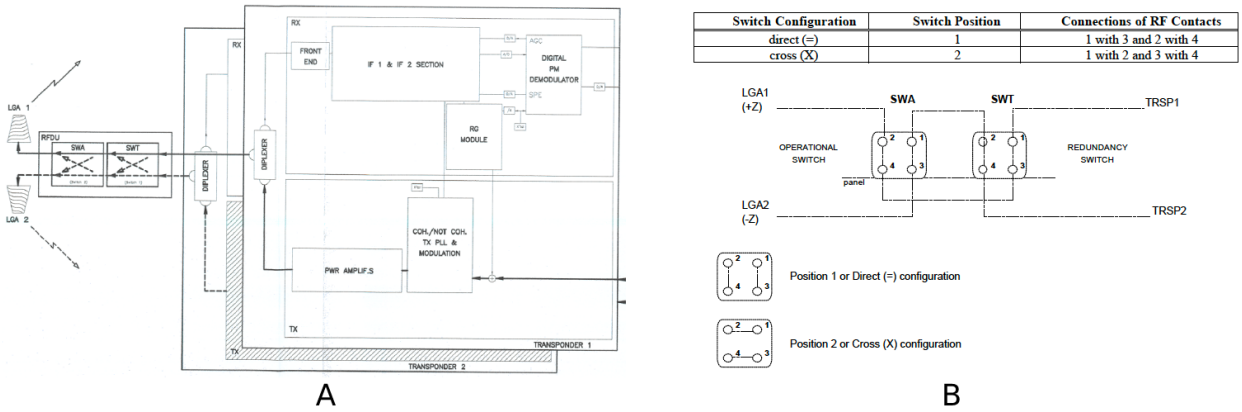

Figure 3. Diagram of the RFDN subsystem shown in figure 3a and the configuration of RFDN switches in figure 3b.

The antenna can be used if the angle between the antenna and the ground station is below the threshold of 100° for an altitude ≥ 60,000 km and 110° for an altitude < 60,000 km. The equation 1 shows how to calculate the angle for antenna 1 (LGA1), where Θ is pitch (rotation around y-axis), Φ is yaw (rotation around z-axis) and Ψ is roll (rotation around x-axis). The formula neglects the centre of rotation of the satellite and the angle is calculated between the ground station and Z-Y plane of the satellite. This simplification can be made because the altitude is many times greater than the distance between the antenna and the centre of mass. Similarly, the angle for antenna 2 (LGA2) can be calculated using equation 2. The swap between the transponders can be performed when both antenna angles are within the range from 80° to 100° for an altitude ≥ 60,000 km or if altitude < 60,000 km from 70° to 110° otherwise.

Equation 1: Calculation of the angle for antenna 1 (LGA1):

$$LGA1_{GS} = \theta + 25 * cos(\varphi) + \psi * sin(\theta)$$

Equation 2: Calculation of the angle for antenna 1 (LGA1):

$$LGA2_{GS} = 180 - \theta - 20 * cos(\varphi) - \psi * sin(\theta)$$

Figure 4 shows angles calculated for the fourth version of revolution no. 2541. At the beginning of the revolution, antenna 2 is above the specified threshold represented by the upper green dashed line, therefore only antenna 1 can be used. However, the angle of antenna 1 reaches the upper threshold around *2022-08-24T14:55:42*, resulting in a need to conduct transponder swap. The swap can be scheduled when both antenna angles are within the region shown by green dashed lines and into window with no commands for at least 10 minutes. For this particular case, the swap has to be scheduled between *2022-08-24T14:06:49* and *2022-08-24T14:55:42*. After that, the antenna 2 can be used for the rest of the revolution. The grey curve represents the altitude of the spacecraft and if it is below 60000 km, the swap range 70° to 110° can be used, otherwise, the 80° to 100° range is used.

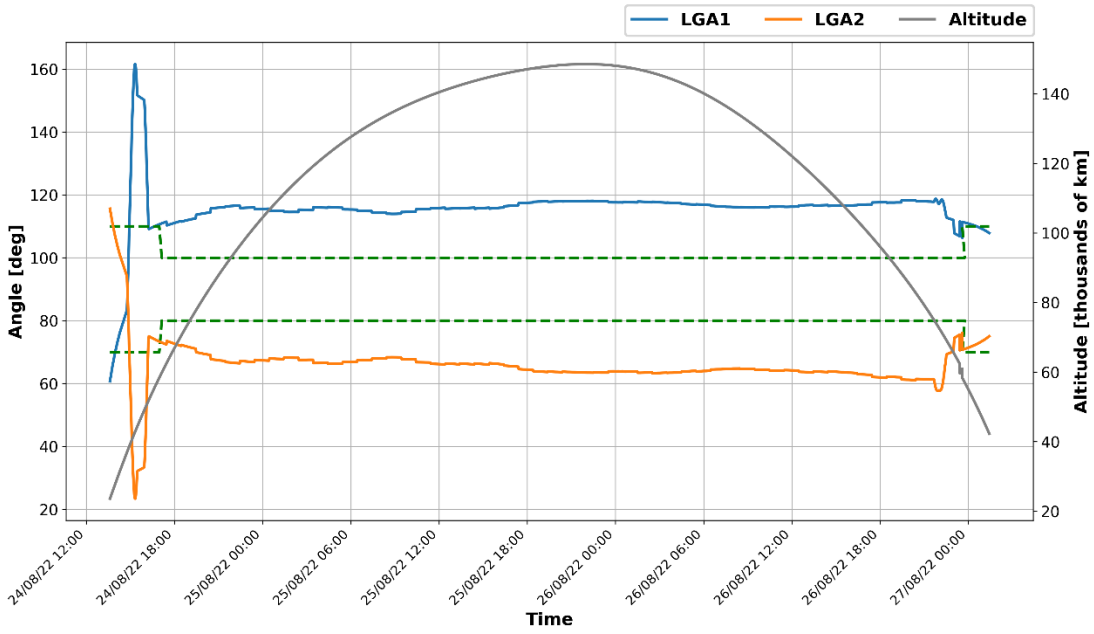

Figure 4. Antenna angles for version 4 of revolution 2541.

Implementation

The FD system suggests the start and end of possible intervals for transponder swaps, however, this system is too conservative and always uses lower limits (80° lower threshold and 100° upper threshold). This leads to extra swaps resulting in an unnecessary amount of transponder ON/OFF cycles. A new approach to calculate the smallest amount of swaps was developed and is described by the following algorithm:

load TC coverage intervals from EPO (timeline schedule)
filter attitude records from ESM (orbit parameters) based on TC intervals
calculate angles for both antennas
find time intervals when each antenna is above the threshold
find time intervals when antenna angles are within the overlapping region
if LGA1 and LGA2 can be used
    calculate swaps for each antenna if it was used at the beginning
    use antenna at the beginning with fewer swaps
    if antenna at the end of the previous revolution is not the same as the antenna with fewer swaps
        insert swap between revolutions
    end if
else if LGA1 can be used and LGA2 cannot be used
    calculate and use swaps for LGA1
    if antenna at the end of the previous revolution is LGA2
        insert swap between revolutions
    end if
else if LGA1 cannot be used and LGA2 can be used
    calculate and use swaps for LGA2
    if antenna at the end of the previous revolution is LGA1
            insert swap between revolutions
    end if
end if
generate plot and save the transponder used at the end of the revolution

Figure 5 shows the graphical output of the tool for revolution 2547, version 08. As the plot suggests, only antenna 1 can be used at the beginning of the revolution. This means that swap between revolutions 2546 and 2547 had to be scheduled because at the end of the previous revolution, antenna 2 was used. The chart also shows that three swaps were inserted in the revolution 2547. The first swap was scheduled at *2022-09-09T13:14:36Z* with approximately 90-minute window. The second swap at *2022-09-10T16:21:52Z* had gap only 5 minutes between commands, and therefore, the mission planner would have to reschedule some commands to increase the window duration. Lastly, the third swap was scheduled at *2022-09-11T16:06:56Z* with almost 60-minute gap.

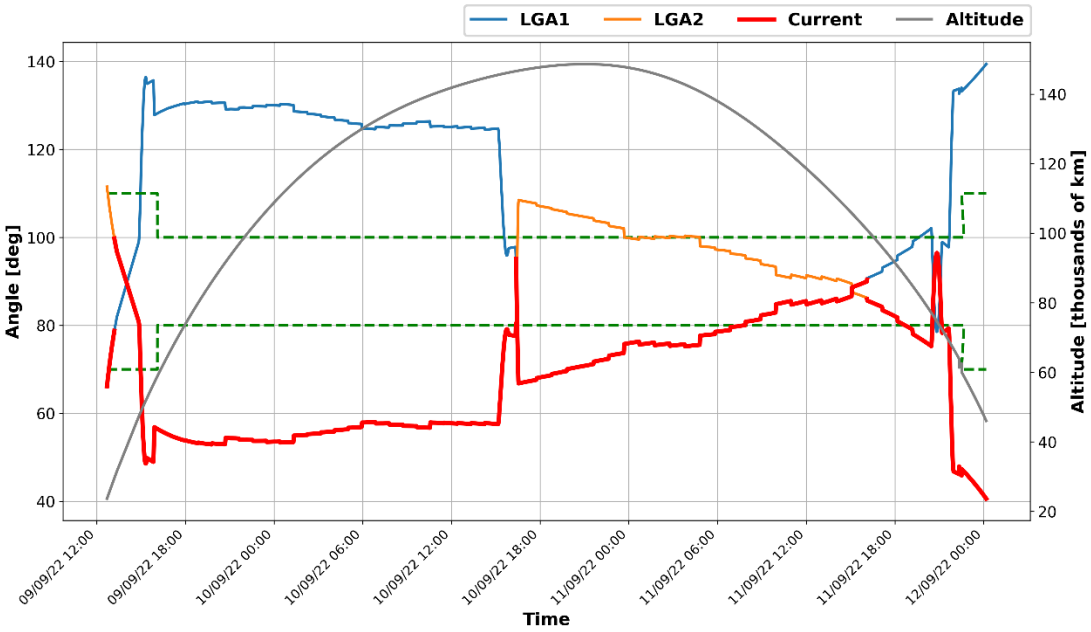

Figure 5. Transponder swap scheduled for revolution 2547, version 08.

Development process, verification and validation

Python was chosen to implement the tool due to its capability to run on multiple operating systems, library availability and development efficiency arising from its dynamic type system and its familiarity within the FCT. Firstly, the development process involved the high-level design of the algorithm by writing down its major steps. After the chosen approach was approved by the FCT, the implementation stage started. It involved heavy use of test-driven development (TDD). According to [7], the TDD is a development technique where test cases are written first, followed by the implementation code. By using the TDD approach, many test cases were created allowing instant feedback when a part of the code was updated. TDD allowed the setting of small development milestones and also an option to quickly verify the performance of all functions during the development. After the development and verification stages were finished, the validation process followed.

The validation process involved a comparison of the algorithm's output with manually scheduled swaps. Table 1. Transponder swap scheduling validation.shows a couple of lines of the validation table. The first criterion of the validation process involved checking that the number of swaps determined by the algorithm is less than or equal to the number of swaps suggested by the manual approach. During the second step, it was checked whether the largest suggested gap can be used. If that was the case, the revolution test run was considered a "Success". As the validation table shows, the algorithm suggests smaller or equal number of swaps for most of the cases. For revolution 2543_02, the algorithm uses smaller number of swaps, however, for revolution 2547_08 it suggests more swap. The close investigation showed that the manual approach ignored the fact that antenna 1 is above specified threshold at the beginning of the revolution. The designed algorithm is in this case more conservative and even for this case the requirement was not relaxed.

Table 1. Transponder swap scheduling validation.

Revolution	Number of swaps suggested manually	Number of swaps suggested by the algorithm	All largest gaps can be used	Status	Comment
2541_04	2	2	Yes	Success	One swap is between revolutions.
2542_05	0	0	Yes	Success	
2543_02	2	0	Yes	Success	Manual approach - swap between revolutions and then at the beginning.
2547_08	3	4	Yes	Success	Manual approach did not insert swap between the revolutions while the algorithm inserted it because the antenna was above the threshold, 3rd suggested swap did not have large enough ED and need to reschedule.
2548_01	2	2	Yes	Success	One swap is between revolutions and for the other there is a need to reschedule some commands.
2551_01	4	4	Yes	Success	One swap is between revolutions and for the other there is a need to reschedule some commands.

Timeline validation

The revolution timeline for the INTEGRAL satellite contains commands to be sent to the spacecraft to conduct instrument measurements, perform slews and put spacecraft instruments into the safe mode. Before this is done, the timeline needs to be validated. Previously, several validation steps were conducted, but these did not evolve significantly since the beginning of the mission. As the satellite operational time increased, problems such as thruster anomaly [5] occurred. Therefore, there was a need to implement a new set of checks reflecting the current state of the spacecraft as well as to automate checks previously performed manually.

Implementation

The revolution timeline is validated by the MPChecker tool developed in Python. shows the input/output and configuration diagram of the tool. The input files to the tool consist of five main types – EPO (timeline schedule), APF (command's parameters), ESM (orbit parameters), revno.txt (revolution start times) and eclip.txt (information about spacecraft eclipses). The revolution number, version and path to the log file folder are inputted via command line parameters allowing simple integration with the MCS. The tool can be configured with two configuration files – config.yml contains general options such as the location of revolution files and config-checks.yml allows to enable or disable certain checks. Even if the check is disabled, it is executed, but its output is not considered in the final result Success/Fail output. This logic was chosen to prevent information loss when certain checks need to be disabled. The tool produces a log file containing reports of all checks, and the final result that can have three possibilities: "Success", "Success with disabled checks" and "Fail". The log file is generated using Python's *logging* library.

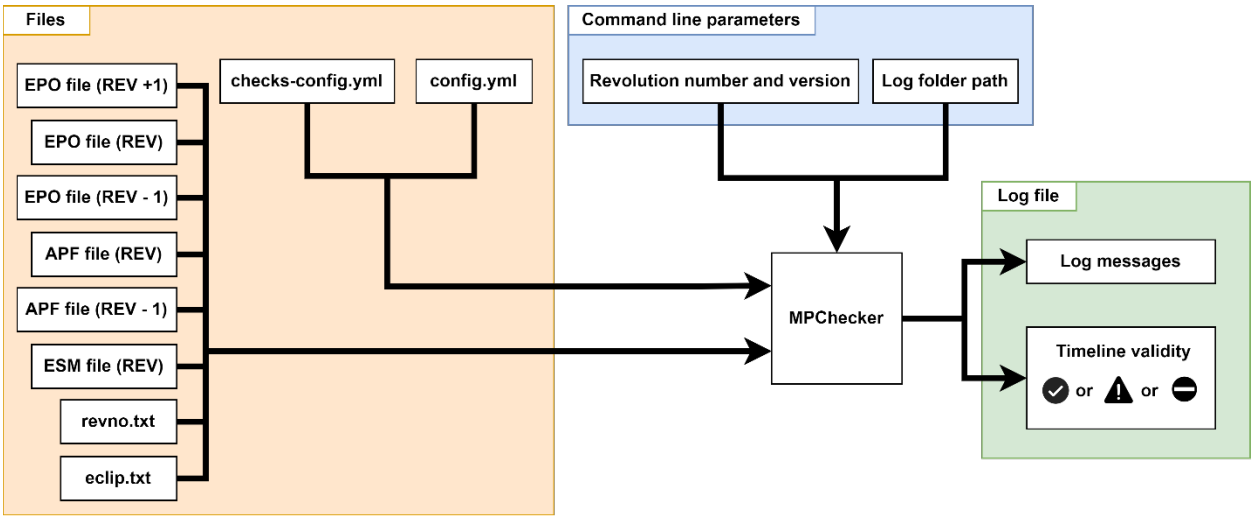

Figure 6. Functional diagram of the MPChecker.

The object-oriented programming concept was chosen to implement the tool allowing to split functionalities into different classes thus allowing for better code reusability and modularity. Figure 7 shows the simplified UML design of the class interconnection. The object of the main class *MPChecker* runs all of the checks where each check produces success or fail output. Data for checks are fetched by the object of *ParsersWrapper* which implements the facade design pattern [8]. *ParsersWrapper* provides a unified interface to the *MPChecker*. This approach splits the process of information extraction and checks execution.

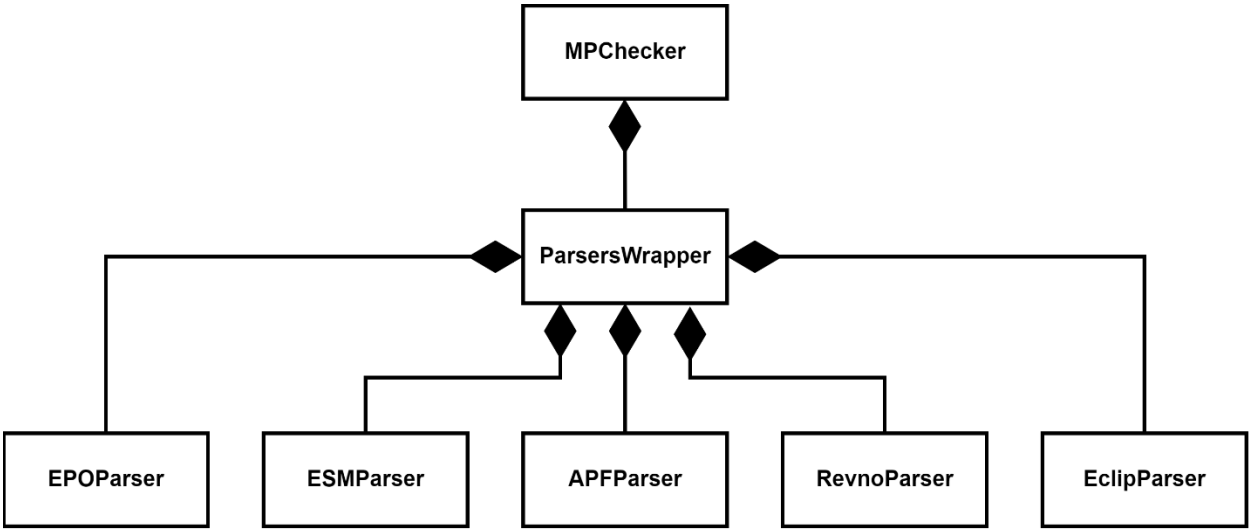

Figure 7. Object hierarchy of the MPChecker.

Before implementing parsers, it was considered to use those that can build an abstract syntax tree (AST) [9] such as Parsimonious. These tools require grammar to be defined, and based on this grammar, they construct an AST. The main disadvantage of this approach was the need to write code for searching in constructed AST during the data processing. In the case of Parsimonious, the suggested approach is to use *NodeVisitor* that visits nodes of the AST and returns the value based on the programmed condition.

Another approach investigated was similar to object deserialisation. As [10] suggests, during object deserialisation, data is first completely extracted from a stream and then converted into an object. An approach similar to deserialisation was chosen to fetch data for the *MPChecker* – each command or record of interest is turned into a predefined object. This approach was chosen mainly due to its simplicity and less amount of code needed to be written.

A total of 12 checks were implemented. These checks can be split into two types: eclipse and non-eclipse. The non-eclipse checks involve, for example, checking of all Event Designator (ED) commands are within the telecommand (TC) window, or that there are no commands controlling instruments after they have been put into safe mode. Checks of this type were implemented using searches through the list of commands and then comparing found results. Other checks require that the eclipse entry and exit times are calculated according to the given rules. An example check that was implemented:

DEBPG100 (ensure that the ED is for the revolution you are generating)

- Verify that the radiation belt entry/exit times are within 5 minutes of the *CRIT_INST_ALT_DESC* of this revolution and *CRIT_INST_ALT_ASC* of the next revolution.
- Check that the time for *CRIT_INST_ALT_DESC* is occurring before the time for *CRIT_INST_ALT_ASC*. If this is not the case, please inform Flight Dynamics. If FD are not reachable, then contact the spacecraft operations engineer (SOE) on call who will issue the relevant OI in order to change the BCPKT timer manually.

Non-eclipse season

- eclipse entry time (*ECL_ENTR*) = ED uplink time minus 2 days 1 hour
- eclipse exit time (*ECL_EXIT*) = ED uplink time minus 2 days

Eclipse season only

If there is an eclipse within the next 10 revolutions (see below), verify that the eclipse entry/exit times for the upcoming eclipse are consistent with the eclipse file for the next eclipse (within a 5-minute margin). To view the eclipse file, open a terminal session and type *eclip*.

- eclipse entry time (*ECL_ENTR*) = 3 minutes before penumbra start time
- eclipse exit time (*ECL_EXIT*) = penumbra end time

Radiation belt entry/exit, *CRIT_INST_ALT_DESC* and *CRIT_INST_ALT_ASC* times are loaded from current revolution and next revolution files using *ParsersWrapper*. If *CRIT_INST_ALT_DESC* is not before *CRIT_INST_ALT_ASC* (of the next revolution), then the check will fail. The next step involves checking if there is an upcoming eclipse season or not. This is done during the initialisation of the *MPChecker* object because more checks require the knowledge if there is an eclipse season or not. Possible cases for the revolution eclipse types are non-eclipse season, pre-perigee and post-perigee eclipse. The described check is specific because even in non-eclipse season, there might be a need to perform the eclipse part if there is an eclipse in the following 10 revolutions. This information is obtained from the *eclip.txt* file that contains information about when the eclipse umbra starts and ends.

The developed tool was integrated into the mission control system SCOS [11] to allow simple interaction with the tool. Figure 8 shows the screenshot of the SCOS when a generation of timelines from 2541 to 2547 is conducted. The mission planner only needs to press the "MPCHECKER" button that runs the tool. If the output is "Success" or "Success with some checks disabled", the timeline generation process can proceed. The option "Success with some checks disabled" was implemented to allow the mission planner to proceed with the timeline generation even in the case when some of the checks fail. If the provided revolution data produce a "Fail" output, the timeline generation cannot proceed. The mission planner has to review the log messages and perform either a replan or in an extraordinary event, disable certain checks. One of the reasons for this is that there are some rare timelines that need to be generated even with failing checks. Examples of these timelines are observations of Earth normally raising warnings and errors

[12]. Another reason is the if a bug in the check code was found, resulting in an errorneous fail, or a requirement change due an on-board anomaly/failure. In these scenarios, the *MPChecker* would prevent generation of the valid timeline.

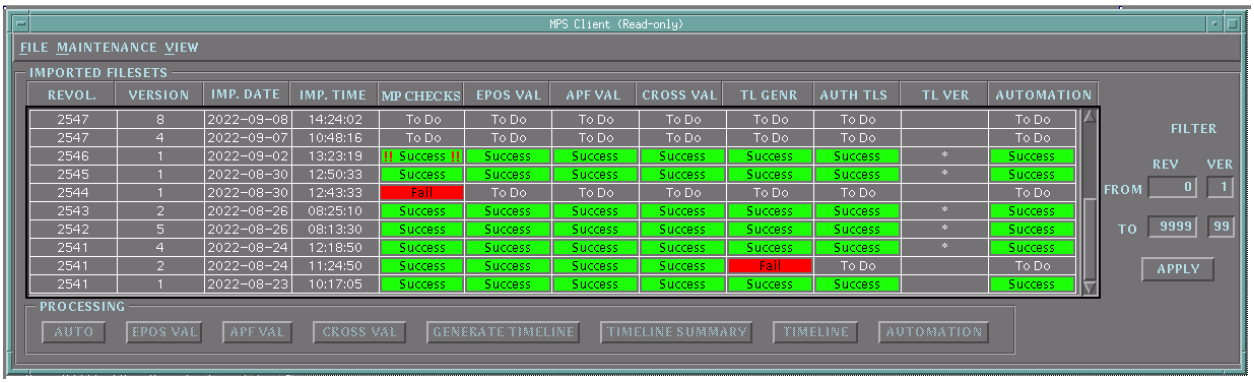

Figure 8. MPChecker integration into mission control system SCOS.

Development Process, Verification and Validation

Python was chosen to implement the tool due to its ability to run on multiple operating systems, library availability and development speed efficiency arising from its dynamic type system. Another reason was the fact that it was already preinstalled on the production machine with old operating system and there was no need to request another software to be installed. The development process also involved heavy use of TDD as for the transponder swap scheduling tool. When the development phase was finished, the verification phase followed. The Verification control document (VCD) was created according to ECSS-E-ST-10-02C and covered the verification process. This document also included a verification matrix similar to the one suggested by [13] which was created for each check. Fig \ref{fig:mpchecker-verification-matrix} shows the verification matrix for check 5.1. The matrix includes the check name, passing and failing conditions for all possible cases of input values, and the name of the test case method performing each verification step. Before the tool was inserted into the mission planning check timeline, there was a need to perform the validation. The FCT conducted the validation by manually inputting timelines from the archive to prevent any bugs. After the tool was fully tested on revolutions from the archive, it was handed over to the software support team who included the tool in the next build of the MCS.

Check 5.1: RMU Calibrations

There is at least 15 minutes gap between AEINT_00 and following PREQ. Each AEINT_00 must be in RMU window defined by RMU_START and RMU_STOP. Each RMU window must have exactly one AEINT_00 command. PREQ cannot be inside the RMU window.

Test ID	Requirement condition	Check output	Verifying test method
51001	Gap between AEINT_00 and following PREQ > 15 min	Pass	test_time_difference_between_AEINT_00_and_PREQ
51002	Gap between AEINT_00 and following PREQ = 15 min	Pass	test_time_difference_between_AEINT_00_and_PREQ_is_15_minutes
51003	Time of AEINT_00 = time of RMU_START	Pass	test_check_passes_when_AEINT_time_is_same_as_RMU_START
51004	Gap between AEINT_00 and following PREQ < 15 min	Fail	test_insuficiet_time_difference_between_AEINT_00_and_PREQ, test_AEINT_00_and_PREQ_with_same_times
51005	No PREQ after AEINT_00	Fail	test_check_fails_when_no_PREQ_is_present_for_one_AEINT
51006	No AEINT_00 inside RMU window	Fail	test_check_fails_when_RMU_window_has_no_AEINT_ED
51007	Neither AEINT_00 or PREQ present	Fail	test_check_fails_when_neither_PREQ_and_AEINT_are_present
51008	AEINT_00 outside RMU window	Fail	test_check_fails_when_AEINT_ED_is_outside_the_RMU_xwindow
51009	PREQ inside RMU window	Fail	test_check_fails_when_PREQ_is_inside_the_RMU_window
51010	Time of AEINT_00 = time of RMU_STOP	Fail	test_check_fails_when_AEINT_time_is_same_as_RMU_STOP
51011	Multiple AEINT_00 commands inside the RMU window.	Fail	test_check_with_multiple_AEINTs_inside_RMU_START

Figure 9. Verification matrix for check 5.1.

Conclusion

Two useful Python tools that automate the transponder swap scheduling and mission schedule validation, were developed for the INTEGRAL mission. The emphasis on the testing was taken into account from the early development phases by using the TDD approach. The final verification using defined verification processes followed by the validation was conducted and did not find any significant errors. These scripts are specific to the INTEGRAL mission, however, similar approaches and design decisions can be implemented for a different mission. The transponder swap script can be amended for a different mission by changing an input file, code used to calculate the antenna angle and the threshold constants. MPChecker can be amended for a different mission by replacing the input side consisting of parsers and implementing the specific check logic. The developed tools allow the INTEGRAL/XMM FCT to speed up the mission planning phase allowing to spend more human resources on the other parts of the mission. For similar projects related to spacecraft operations, it is suggested to follow the same development process that involves the heavy use of TDD and the modularity of the code.

Acknowledgements

I would like to thank the whole INTEGRAL/XMM FCT for being accepted to the team. Namely, I would like to thank Mr. Stefano De Padova for supervising the internship, Miss Greta de Marco for support on both projects, Dr. Timothy Finn for help with the transponder swap scheduling and lastly, Mr. Richard Southworth for providing the INTEGRAL CAD drawings and Christopher Saloman for developing the *EpoParser* that parses the timeline schedule from the EPO file into *Action* objects.

References

[1] P. L. Jensen *et al.*, 'The INTEGRAL spacecraft – in-orbit performance', *Astron. Astrophys.*, vol. 411, no. 1, pp. L7–L17, Nov. 2003, doi: 10.1051/0004-6361:20031173.

[2] M. Schmidt and D. Heger, 'Managing the Integration of Flight Control Teams Considering INTEGRAL and XMM-Newton Missions', in *SpaceOps 2008 Conference*, Heidelberg, Germany, May 2008. doi: 10.2514/6.2008-3474.

[3] N. Pfeil *et al.*, 'NEW OPERATIONAL CONCEPT FOR GAIA, INTEGRAL & XMM-NEWTON', presented at the International Astronautical Congress 2019, Washington D.C., Oct. 2019. Accessed: Oct. 12, 2022. [Online]. Available: http://iafastro.directory/iac/archive/browse/IAC-19/B6/3/54687/

[4] R. Much *et al.*, 'The *INTEGRAL* ground segment and its science operations centre', *Astron. Astrophys.*, vol. 411, no. 1, pp. L49–L52, Nov. 2003, doi: 10.1051/0004-6361:20031252.

[5] E. Kuulkers *et al.*, 'INTEGRAL reloaded: Spacecraft, instruments and ground system', *New Astron. Rev.*, vol. 93, p. 101629, Dec. 2021, doi: 10.1016/j.newar.2021.101629.

[6] M. Schmidt and M. Kirsch, 'XMM-Newton, ESAs X-ray observatory, the Loss of Contact Rescue and Mission Operations ready for the next decade', in *SpaceOps 2010 Conference*, Huntsville, Alabama, Apr. 2010. doi: 10.2514/6.2010-2123.

[7] J. Hunt, 'Introduction to Testing', in *Advanced Guide to Python 3 Programming*, J. Hunt, Ed. Cham: Springer International Publishing, 2019, pp. 165–174. doi: 10.1007/978-3-030-25943-3_14.

[8] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Reading, MA: Addison-Wesley, 2009.

[9] R. E. Noonan, 'An algorithm for generating abstract syntax trees', *Comput. Lang.*, vol. 10, no. 3–4, pp. 225–236, Jan. 1985, doi: 10.1016/0096-0551(85)90018-9.

[10] K. Eguro and R. Mueller, 'FPGA-Accelerated Deserialization of Object Structures', Sep. 2009, [Online]. Available: https://www.microsoft.com/en-us/research/publication/fpga-accelerated-deserialization-of-object-structures/

[11] R. V. Osorio, J. P. Lemos, T. W. Beech, G. G. Julian, and J.-P. Chaumon, 'SCOS-2000 Release 4.0 : Multi-mission/Multi-Domain Capabilities in ESA SCOS-2000 MCS Kernel', in *2006 IEEE Aerospace Conference*, Big Sky, MT, USA, 2006, pp. 1–17. doi: 10.1109/AERO.2006.1656141.

[12] C. Lozano and A. S. Murillo, 'INTEGRAL - RENAISSANCE OF OCCULTATION TECHNIQUES USING THE EARTH'. Accessed: Nov. 28, 2022. [Online]. Available: http://iafastro.directory/iac/archive/browse/IAC-11/B6/2/9774/

[13] Y. Liu, Q. Guo, W. Liu, R. Wang, and D. Zhao, 'Verification Matrix Applied During Verification Process of Airborne Software', in *Proceedings of the 5th China Aeronautical Science and Technology Conference*, vol. 821, Chinese Society of Aeronautics and Astronautics, Ed. Singapore: Springer Singapore, 2022, pp. 1022–1026. doi: 10.1007/978-981-16-7423-5_103.