Automated Planning versus Manual Operations in the context of the Link Management System for EDRS – SpaceDataHighway

Sven Prüfer, Tobias Göttfert and **Maria T. Wörle**
Mission Planning System Engineers
German Space Operations Center (GSOC)
Deutsches Zentrum für Luft- und Raumfahrt e. V.
Münchener Straße 20, 82234 Weßling, Germany

Abstract

The Link Management System (LMS) is the automated planning system for the EDRS (European Data Relay System)–SpaceDataHighway payloads at the German Space Operations Center (GSOC) and has been in continuous use since 2016. Recently, it has been extended to include the EDRS-C mission, scheduled to launch in 2019, besides EDRS-A. In this paper we discuss lessons learned and take-aways for multi-mission development of planning software as well as operations of automated planning systems. The interplay between automation and manual intervention in the LMS case is explained in multiple examples giving rise to suggestions for future development of automated planning systems. Furthermore, the LMS is described in the context of CCSDS. This paper builds on and extends (Göttfert et al. 2018).

1 Introduction

1.1 The EDRS mission

The EDRS (*European Data Relay System*)–SpaceDataHighway is a joint ESA–Airbus mission consisting of two geostationary relay systems for transferring data to and from low-earth-orbit (LEO) satellites via laser-optical and Ka-band inter-satellite links (O-ISL and Ka-ISL, respectively). Since 2016 the EDRS-A hosted payload has been in operation on a Eutelsat platform and has supported more than 20 000 links since then. In contrast, EDRS-C, which supports O-ISLs only, is a dedicated satellite scheduled to be launched in summer 2019. The overall system is operated by Airbus.

The EDRS ground segment set up by Airbus and DLR GSOC comprises a Mission Operations Center (MOC), four ground stations and two control centers, see Fig. 1. The EDRS-A Devolved Payload Control Center (DPCC) and the EDRS-C Spacecraft Control Center (SCC) are located at GSOC whereas the MOC serving both missions is located at Airbus. EDRS customers have an interface to the MOC which forwards requests to DPCC or SCC to command the respective spacecraft (S/C) to support the link. For a general introduction to the EDRS–SpaceDataHighway mission, see (Hauschildt et al. 2014).

1.2 The Link Management System

The Link Management System (LMS) is the mission planning system (MPS) developed by GSOC as a part of DPCC and SCC for the EDRS mission, see (Göttfert et al. 2016) and (Göttfert et al. 2018). It automatically processes requests by MOC for Ka-ISL and O-ISL links and send corresponding flight operations procedure requests (FOP requests) to the command tools within the control center, such as the Generic Procedure Generator (GPG) or the Automation Engine (ATM), see (Scharringhausen and Beck 2017). In addition, the LMS

- schedules various laser-communication terminal (LCT) related tasks,
- checks for multiple consistency requirements between the various activities,
- reports the status of planned events,
- tries to maximize the fill-level of the Time-Tagged Telecommand (TTC) onboard buffer, i. e. it tries to schedule as many links as possible and
- schedules the uplink of forward data (FWD, see Section 2.2) to an on-board buffer for later transmission to a customer satellite.

For EDRS-A, in addition, the LMS initiates pointing calculations for the Ka antenna and keeps track of the onboard pointing record table. The LMS therefore supports a lot of interfaces to the MOC and several services within DPCC and SCC, respectively, which is illustrated in Fig. 2. Notice that although the LMS is not responsible for the scheduling of the links themselves, it has several scheduling duties regarding e. g. LCT-related tasks and FWD.

From a software point of view the LMS runs continuously as a Windows service and uses a *connected transaction loop mechanism* which cycles through various phases, in particular file ingestion, scheduling and task export, see Fig. 3. The LMS GUI is separate from the service and connects via a Windows Communication Foundation (WCF) interface. This makes the automatic system run independently of the GUI. There are multiple instances of the LMS installed for EDRS-A and EDRS-C as well as for prime, backup and simulation purposes.

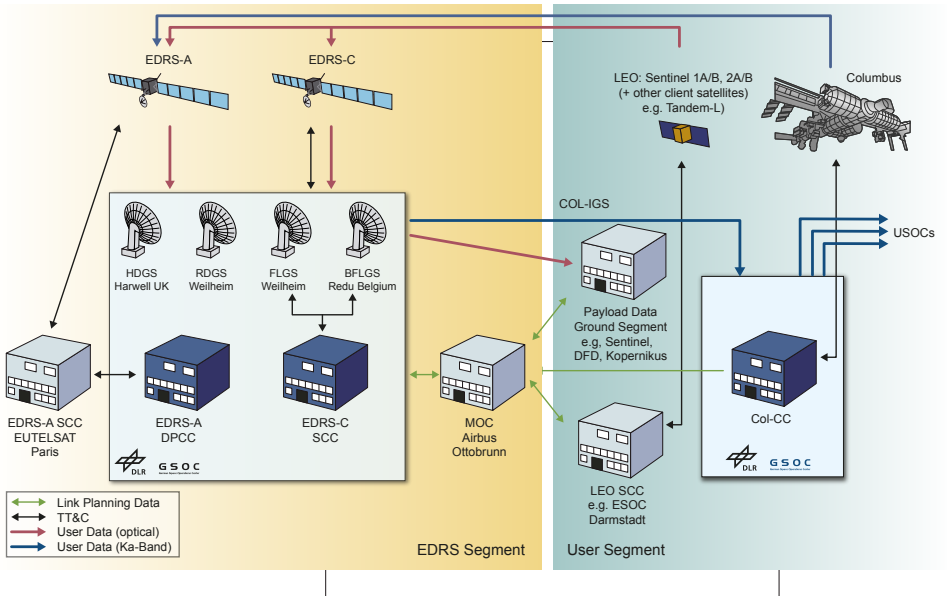


Figure 1: Overview over the EDRS–SpaceDataHighway ground system

2 Combining two Missions in one Planning System

As EDRS-A and EDRS-C are very similar missions regarding the link workflow and interfaces, the EDRS-A LMS code base was reused for EDRS-C instead of completely rewriting a new software component. In particular, both missions serve the same link request interface with the MOC and use a similar workflow for FOP requests with the GPG and the ATM, respectively, at least from the LMS point of view. Since the overall link scheduling is done by MOC, there is no need for synchronization between the two LMS instances for the two missions, as e. g. customer link requests are distributed among the two satellites by MOC already. The following aspects are treated differently by EDRS-A and EDRS-C:

- There is no Ka-ISL antenna in EDRS-C and thus the corresponding Link Management System functionality is not used.
- LCT maintenance for EDRS-A includes updates for the on-orbit propagation products, LCT time and LCT alignment, whereas for EDRS-C both on-orbit propagation and time updates are handled by the LMS as parts of the satellite bus maintenance.
- Station-Keeping Maneuvers are known to the EDRS-C Link Management System in advance and thus additional consistency checks are possible.
- A bigger on-board time-tag schedule buffer which is used solely by the Link Management System allows for easier usage of TTC upload capacity.

See (Göttfert et al. 2018) for a more detailed explanation of the similarities and differences. In the next sections we will discuss various operational requirements and experiences that led to the design decisions for the LMS. In particular we will describe the new *remote monitoring* feature of the LMS.

2.1 Operational Requirements

Even though GSOC's responsibilities for EDRS-A and EDRS-C differ as they include the platform operations for EDRS-C, the payload operations are still so similar that both missions will be operated from the same control room. There is a large overlap in the team members for both missions. The experience gained from EDRS-A payload operations influenced multiple aspects of payload operations for EDRS-C such as the redundancy concept or the automated flight procedure instantiation. Reusing the LMS for EDRS-C means that operators are already used to the GUI and its operations.

Having the same people operating two different missions using the same software in the same control room also poses a risk as operators might easily confuse which mission they are dealing with in a particular situation. This risk was noticed very early in the design phase and various countermeasures have been implemented, some of which we will describe in the next sections.

LMS GUI The LMS GUI needs to show clearly which mission is being served by the respective instance. This is done by a unified color scheme such that both missions are easily distinguishable for all software components at a first glance. Of course, distinguishing a GUI via colors has its drawbacks as this might be difficult for color-blind people or more than two satellites. Due to the existence of multiple processing chains per mission for redundancy as well as testing purposes, the LMS GUI needs to show clearly which of these chains it is connected to as well. In case of EDRS-C

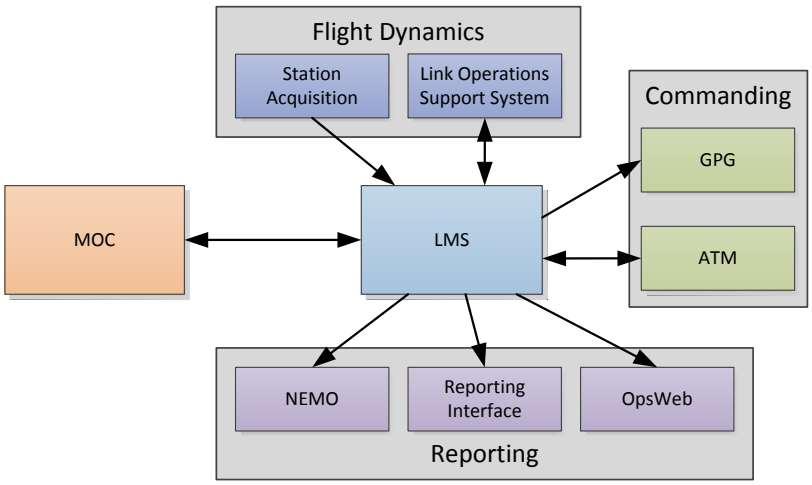


Figure 2: A schematic overview of the top-level LMS interfaces. The arrows indicate the direction of the flow of information.

there must also be a notification which database the LMS is connected to, as a redundancy switch to a different processing chain requires a database switch on the corresponding process server. This switch can be set in the LMS configuration and requires a restart, however, afterwards the operator needs to know that this switch was indeed executed. For EDRS-A this is not necessary as the LMS always connects to a local database. Regarding the different functionality the LMS GUI uses various mission-specific views and controls for the respective missions. A comparison between the two new GUI versions' layouts can be seen in Fig. 4.

Configuration Management Besides the GUI and its usage, also the question of configuration management had to be tackled. For the LMS, the default configurations are deployed together with the installer, i. e. any operational change during the mission needs to be tracked by the developers and included in the next release. For EDRS-A this has been used quite successfully, for EDRS-C however this needs some change to the installation procedure as an installer needs to deploy slightly different configurations depending on the usage of that particular LMS instance. Consider e. g. two LMS's, one of which shall be operational and thus connects to a database with the operational data and another one which connects to a simulation database. As this configuration is deployed with the installer one needs a safe default in order to avoid any unintententional setup.

In general, there are a lot of different ways to deal with this issue depending on the deployment system: If the software is installed as an application on a server one might e. g. deploy the configuration together with the installer or have a local configuration on the server which does not get modified when updating the application. The latter approach for local files has the advantage that one can track changes in the configuration very easily via some version control system. A configuration roll-out by the installer makes adding new configuration settings safer but necessitates additional manual steps during installation in order to keep machine-specific settings. As the LMS configuration still changes quite often due to newly added features, it was decided to continue rolling out the configuration together with the installation.

2.2 Development and Deployment Strategy

As outlined in (Göttfert et al. 2016) the code base for the Link Management System is the same for both EDRS-A and EDRS-C in order to reduce work overhead from e. g. common bug-fixes and interface changes. A compile-time flag switches between EDRS-A and EDRS-C. There are extensive unit tests covering both common aspects and mission-specific ones. Any change to the code base is automatically tested for both missions by a continuous integration server whenever it is pushed to a central repository. This allows for a short feedback loop and fast discovery of regressions. An even more extensive continuous integration pipeline including black-box integration tests of the whole running system would allow for catching more complicated problems earlier in the development process, however, this has not been setup yet for the LMS. In particular, time-sensitive issues, such as timing problems in asynchronous operations, often elude unit tests and are generally more difficult to test. A sound integration-test framework can help resolve these problems from the beginning. Let us illustrate this in an example:

One feature of the EDRS–SpaceDataHighway mission is the provision of so-called forward data (FWD) which refers to the uplink of data *to* the customer S/C. In case of O-ISLs this data is uplinked asynchronously to the EDRS satellite in advance to the connection. In order to avoid sending wrong or incomplete data to the S/C, there is a safety Delete-FWD-Buffer TTC scheduled shortly before the beginning of the link if the FWD upload has not finished at the time of the O-ISL uplink. This TTC gets deleted once the FWD upload is finished. However, since the FWD upload can be deleted and resubmitted by the MOC (e. g. if the customer notices a late error in their originally submitted FWD) the safety Delete-FWD-Buffer TTC might need to be recommanded when a new FWD uplink for the same O-ISL begins. This opens a vast amount of possible scenarios depending on

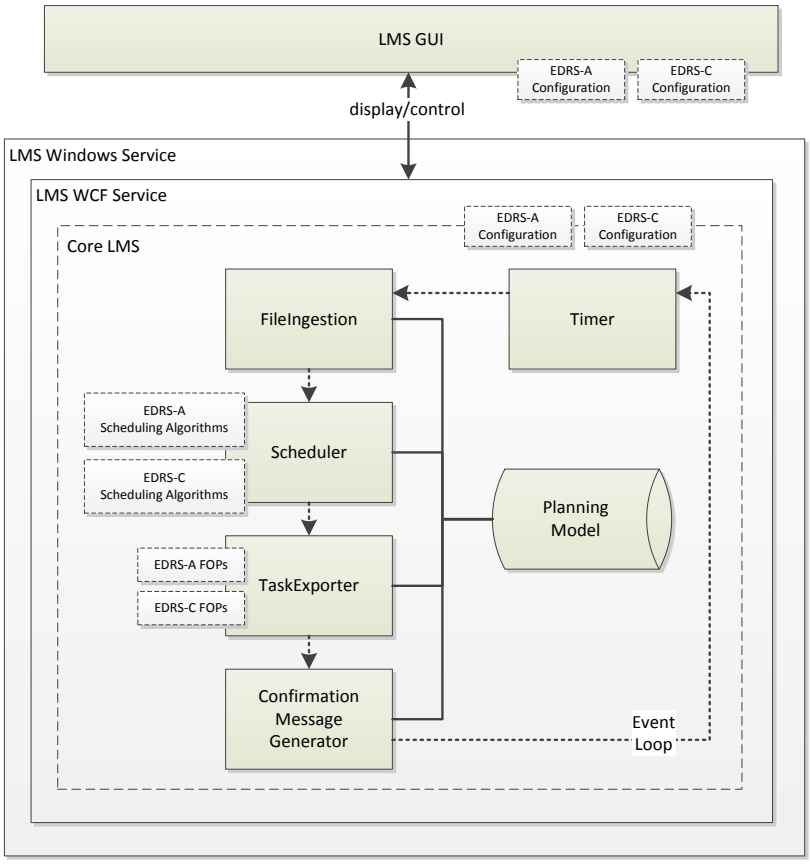


Figure 3: The architecture of the LMS. The main part is the connected transaction loop, which executes perpetually the following tasks: importing new files, scheduling of requests contained in these files, exporting FOP requests and sending feedback to MOC.

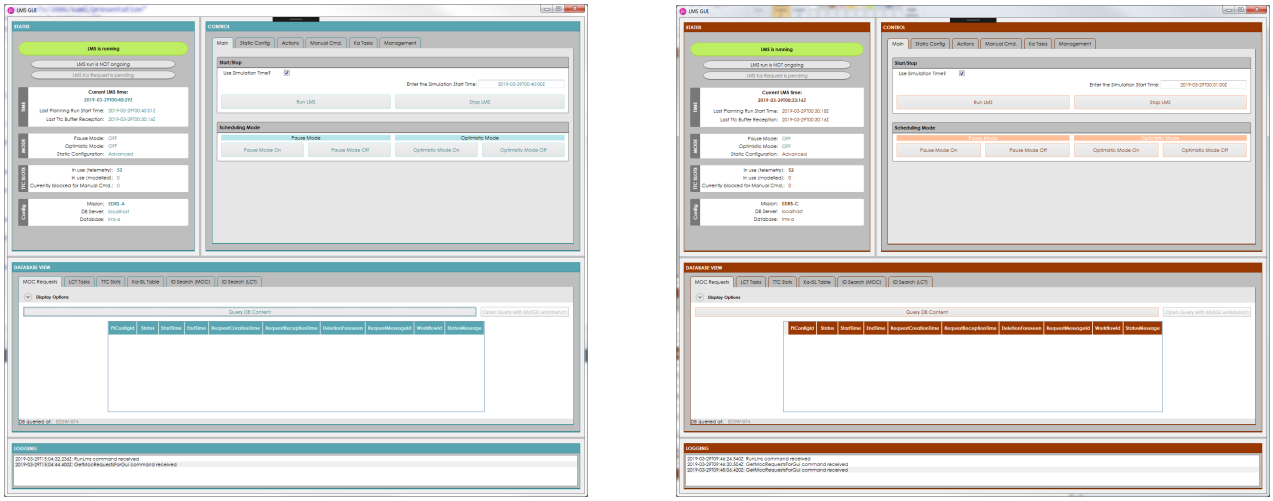
the timing of the various requests and confirmations, all of which need to be tested. Since it is easy to accidentally break this functionality during later development, one can benefit tremendously by automating suitable scenario or black-box tests.

To make the LMS even more resilient against an accidental mixup of the two missions, it was decided to include mission-specific files in the respective mission's installer only. This includes configuration files of the Plato scheduling algorithms and planning constraint generation, see (Lenzen et al. 2012),e. g. for Ka-ISL handling which are obviously not needed for EDRS-C. Together with the mission-specific configurations, this approach asks for an automation of the release pipeline in order to reduce the manual overhead as well as error-proneness for new releases. The automation of the LMS release process is not completed yet.

As the LMS is a standalone application and not used as a library, it is not necessary to use semantic versioning or similar techniques. Instead, versions of the LMS correspond to major features or phases in the EDRS timeline when an upgrade of the running system is feasible. However, due to the mission compile-time flag, there are effectively twice as many versions, making the mission configuration management more difficult. One may avoid this problem only, if the individual missions were similar enough to allow for common configurations or by unifying the individual MPS's which couples both relay satellite operations needlessly. The LMS installer uses a different application name for both missions in order to facilitate the distinction.

2.3 Remote Monitoring

To support test runs of the EDRS software including the LMS alongside the operational system, several independent and parallel processing chains were established. Thus, we run for each EDRS mission multiple LMS instances at the same time. Furthermore, the control room terminal environment is completely virtualized such that an operator can access any machine from any console. Although the current spacecraft controller shall be the only person in control of the LMS in nominal situations, there might be the need to access information from the LMS for e. g. debugging purposes during normal operations by other operators, too. In that case one needs access to the LMS via a GUI in a pure *monitoring* mode which is connected remotely to the operational system. The original design of the LMS allowed for an easy implementation of this requirement imposed later on during the operational mission phase: In fact, the WCF ser-



(a) GUI for EDRS-A

(b) GUI for EDRS-C

Figure 4: The new LMS GUIs with different colors to facilitate distinction in a shared control room environment.

vice running the LMS can be configured to allow for connections via a network interface. This way, a second GUI can connect to the system to monitor the state of the LMS. Special care had to be taken to restrict the commanding capability to at most one such remote GUI connection at a time in order to have a clear command structure. It should be emphasized that this is not the same as authentication of users which can also restrict command capabilities to a small set of users, but even e. g. among flight directors there may be only one in command at any time. The LMS GUI itself then also needs to show whether it is in a commanding state or only monitoring besides providing a way to request or relinquish command. This setup is illustrated in Fig. 5.

Currently, for the remote monitoring functionality of the LMS a working prototype exists, however, the details described above which are necessary for safe operations still have to be implemented and tested.

3 Automation and Manual Operations in the context of EDRS

In this section we want to discuss some experiences regarding the interplay of automated planning systems and manual operations in the context of the Link Management System. Even though the LMS is well tested and is capable to deal with most nominal and off-nominal situations, there is still a need for some manual interaction from time to time. Such situations might occur due to some error in the LMS or some outside anomaly. In any case, operators need to deal with this situation, reliably and quickly.

3.1 Coexistence of Automatic and Manual Operations

The main requirement for an automatic planning system is of course that it is continuously running and reacts correctly to the various kinds of input it receives. In this section, we want to collect issues related to operations one should keep in mind when designing such a system.

First, if there is a problem with the planning system itself or some invalid situation is detected, it shall alert an operator via a well-designed interface. In case of the LMS this is NEMO, see (Hauke et al. 2012), which has been in use for other missions at GSOC as well and operators are thus already used to working with it.

Furthermore, in case of a contingency it needs to be possible to stop the automatic operations in order not to interfere with any manual efforts. Note that *stopping* might have to mean very different things depending on the type of mission and contingency. In case of the LMS, for example, it still accepts link requests even in "paused" mode. However, it does no longer schedule activities or export FOP requests and does therefore not interfere with manual commanding. This has the advantage that all reporting functionality to the MOC is still working and the system stays responsive.

Finally, even a perfect automatic planning system will have to deal with external problems during real operations, and operator interactions eventually have to take place for repairing these. The LMS therefore needs to be able to work alongside limited manual operations. For example, suppose a command execution fails after the LMS has exported a FOP request, so the LMS gets notified and sends corresponding status messages. If operators are able to recover from this error, e. g. by repeating the command execution manually, they need a way to let the LMS know what happened. This might be because updated status messages have to be sent via the nominal workflow (so that the MOC knows that the command was finally successfully executed), or because the execution of this command needs to be known to the LMS in order to continue with nominal operations (e. g. in case of Ka-ISL FOP requests the LMS keeps track of the state of the Ka-antenna which therefore must be modelled correctly). The automated system thus needs a predefined and well-tested way of overwriting its in-

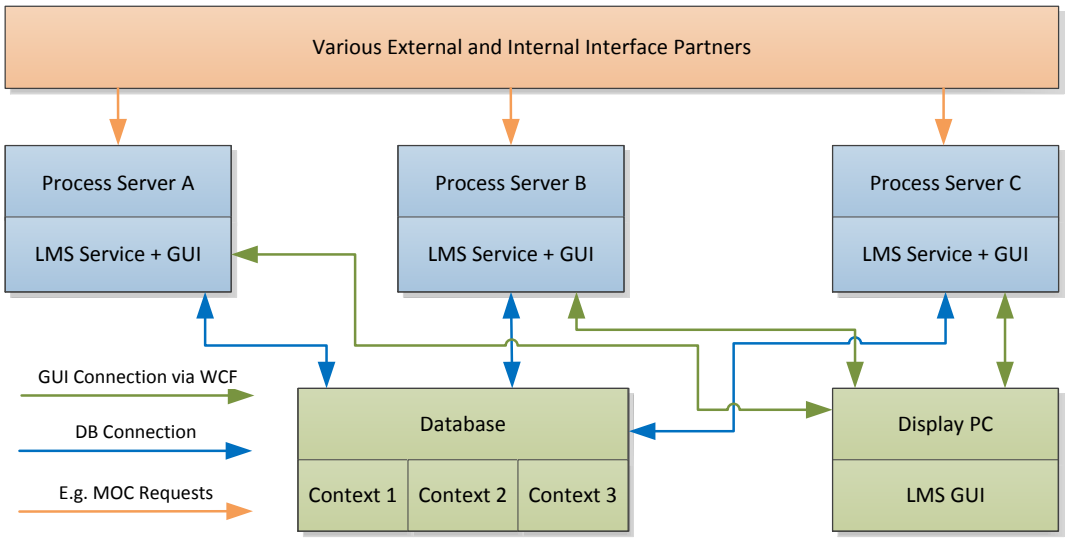


Figure 5: For EDRS-C, the LMS will get a separate standalone GUI allowing operators to access the LMS state remotely. This figure shows the various LMS instances together with their database and GUI connections. Here, *Database* stands for a high-availability cluster containing several databases for operational as well as simulation purposes, referred to as context. Note that this figure concentrates on the LMS and does not show multiple further SCC features such as redundancy for the TM/TC chains and the database or the other software components running on the Process Servers.

ternal state in case there are unforeseen external changes as well as a possibility to allow for manually triggering certain processes such as e. g. a repeated export of a status report. For the LMS these interfaces are added in an ad hoc way whenever the need arises. Some LMS tasks, such as sending a revised status report, are easy to execute manually due to file-based transfers.

3.2 Operational Experience

In this section we will add further operational experiences to those described in (Göttfert et al. 2018) already.

Of course, having an automated planning system reduces the manual work operators need to perform but this comes with several new difficulties that need to be addressed. The LMS has been running for months without any interruption and minimal manual operations needed to the extent that the LMS GUI is usually turned off and only consulted when specifically needed. About once per week on average the LMS has been paused for a few minutes, usually due to some outside interruption in the processing chain, in which case the automated LMS operations would have been distracting. Note that the LMS kept running during these pauses, see Section 3.1.

One difficulty is the reduced experience of the operators with the LMS as they barely ever need to interact with it. This forces developers to take care about different user interface aspects, such as the way LMS problems are presented to the operators. Messages should be as concise as possible and should avoid any ambiguous expressions. For example, from time to time the LMS is not capable of deleting an onboard TTC immediately because it is still waiting for some data necessary for deletion. In this case the LMS buffers the deletion and retries later on. Since this deletion is critical for operations the operator was made aware of the buffering via NEMO and told to check at some later time if the deletion was successful. Operators misinterpreted this message to say that there was a serious problem and became alarmed, requiring explanation of this issue by the developers. This notification had to be adapted to be more informative and an operations procedure had to be deliverd for the operator to know when and how to check that the situation was back to completely nominal.

The LMS pause mode turned out to be surprisingly useful for reducing difficulties that the operators had with the LMS, as it allows to decouple the automatic MPS from manual operations e. g. during contingencies. As this mode keeps the LMS running, operators are still able to examine the planning state when needed and the interface to MOC is still served normally. Since pausing the LMS is very easy, operators have adopted the strategy to turn on the pause mode whenever nominal operations are interrupted, thereby reducing the need to deal with unwanted interference from the automated MPS which in some situations might require more detailed knowledge about the LMS.

After three years of operations, the LMS receives very few bug reports but of those received, a large part is due to misleading reports or feedback of the LMS. Another potential source of confusion in reporting messages is a different usage of the same terms. One needs to keep in mind that an operator sees the whole processing chain and not just the LMS causing e. g. words like *ingestion* or *export* to have different meanings. It is thus heavily recommended to design reporting frameworks and their content in a concise way to minimize any possible operator confusion later on. In addition, a regular training refreshment from time to time with the possibility to highlight seldomly used aspects might also

be helpful, not only for the operators but also for the developers to stay in touch with the users and their needs.

When designing an automated MPS, it is very difficult to foresee any possible way external problems might interfere with automatic planning. Therefore, at some point there is a need to synchronize the state of the MPS in an irregular manner with some external source. A regular state synchronization is e. g. an operator commanding a TTC directly in which case the LMS receives this information via the standard ATM–LMS interface, by the respective uplink and execution confirmations and the TTC buffer content notification. However, in case e. g. the onboard Ka-ISL pointing table is deleted, the LMS needs to know this in order to assume an empty table and reexport the correct Ka-ISL pointing records. In nominal automated operations this type of interface is not needed and was therefore not implemented initially. Instead, this was added later on in an ad hoc way which caused a lot of additional work. Of course, it would have been better to put this generic state-synchronizing workflow into the design from the beginning.

3.3 Evaluation of CCSDS for the LMS

The CCSDS Report Concerning Mission Planning and Scheduling (CCSDS 2018) defines four principal mission planning services that are expected to be implemented: planning request service (PRS), plan distribution and retrieval service (PLS), planning process management service (PMS) and plan execution management service (PES). Furthermore, the information model of CCSDS defines planning requests (PRQ) and plans (PLN) as the main exchange objects between planning systems which may contain and/or reference further planning data objects such as events and activities. Below this communication level there exists the Message Abstraction Layer (MAL) specified by CCSDS, which allows a finite set of interaction patterns between mission operation services, see (CCSDS 2018) and (CCSDS 2010). For more ideas how to implement non-planning related CCSDS features in a mission, see e. g. (Coelho, Koudelka, and Merri 2016) and (Evans 2016).

Considering the tasks of the LMS from Section 1.2, the LMS is a mission planning system which incorporates PRS, PLS and PMS. This is because it executes the planning itself, delivers (partial) information about the plan, self-triggered as well as upon request, and is able to control parts of the planning via the GUI by e. g. scheduling some manual tasks or stopping the system. However, the LMS GUI might also be considered a planning user for observing and managing the planning process because it is separated from the actual LMS service, see remote monitoring in Section 2.3. The PES, in contrast, is not part of the LMS functionality, but the related command system components. Furthermore, the overall link planning workflow for EDRS constitutes a devolved planning system formed by two separate entities. Since the actual inter-satellite link scheduling is done by MOC and its result schedule (split into single or multiple requests) is taken as input for the LMS we have in fact two planning systems, each implementing aspects of these three CCSDS services in the EDRS–SpaceDataHighway mission. The interface between MOC and LMS is completely asynchronous, for some requests there is a single response while for others there are mutiple responses to be sent to MOC. The MOC requests links to be scheduled by the LMS and expects reports on them which is the basic premise of a PRQ. Although the LMS checks for consistency and may reject links if there were any problems found, it is not allowed to change the link parameters. An outline of the interfaces using the CCSDS nomenclature is illustrated in Fig. 6.

Improvements of the overall project set-up using the services proposed by CCSDS might be described as follows:

The scheduling of links may still be done at MOC including all feasibility checks. A common planning language among MOC and LMS might be used or, at least, both need to be compatible such as to allow for easy translations between them. The MOC may then send a plan containing these links and all corresponding parameters including resources and constraints to the LMS via the INVOKE pattern from the MAL layer. The asynchronous response from the LMS might include feedback about whether there was a problem updating the local link schedule maintained by it. Having received and accepted an updated link schedule, the LMS may then export corresponding FOP requests (including deletions of TTCs in case a link was removed from the timeline) and plan other activities taking into account the MOC-provided link schedule. The reporting part of the MOC–LMS interface might then be described as a PLS service by the LMS where the MOC can request PLN objects containing planning data via a REQUEST or INVOKE, i. e. a single, either synchronous or asynchronous, response, or even a PUBSUB pattern constantly awaiting updates about the current plan content.

One advantage of this would be an improved separation of concerns, thereby reflecting better the actual relationship of the two planning services and avoiding additional communications necessary to synchronize the state between the two planning systems. Furthermore, the CCSDS approach would simplify the interface in general and thus reduce the necessary amount of testing as interfaces could be reused. If both services were capable of making use of CCSDS-specified export and import of plans, both directions of passing PLN objects, i. e. planning requests from MOC to LMS and reports from LMS to MOC, might be unified in a single interface.

4 Conclusion

For the development of future automatic MPS we would like to summarize the following lessons learnt.

Although a completely autonomous system is very appealing, operational experience shows that there will be (perhaps externally caused) problems that need to be resolved manually. To deal with this problem one should include a safe way to modify the planning state of the automatic MPS into the design from the start. Adding such a functionality later increases the risk of introducing bugs and might cause additional work.

Secondly, one should not ignore the operational side of an automatic planning system as it needs to be monitored and sometimes manually intervened in a high-reliability environment. This includes taking care about distinguishabil-

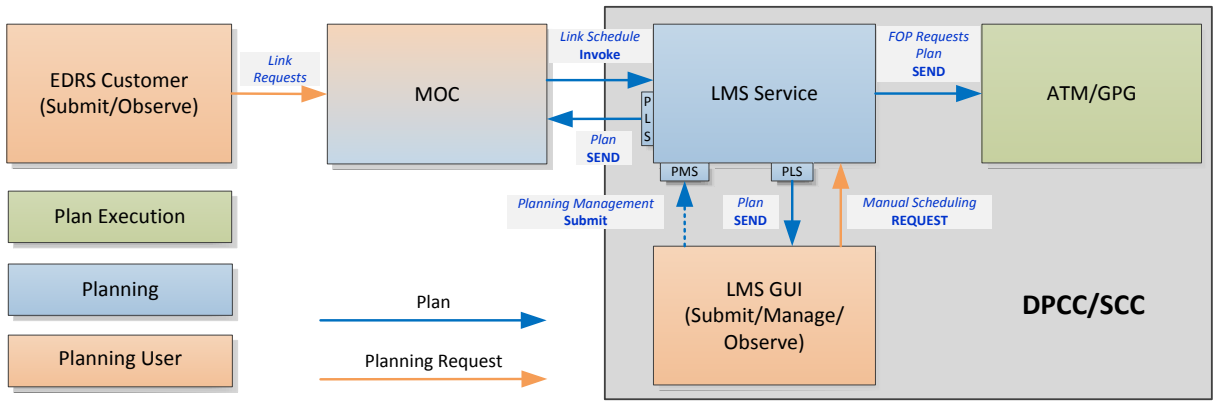


Figure 6: The LMS including its GUI and its most important interface partners using alternative CCSDS nomenclature for the interfaces.

ity of similar-but-different systems, formulation of operator warnings and provision of "human readable" insight into the current planning state.

Automating the build, test and deployment aspects of development as much as possible, is very beneficial, too. In particular when developing the same tool for multiple missions there is a significant overhead associated with repeated work and avoidance of errors when including adaptations, e. g. needed during the mission life-time or required as flavors only needed for one mission, so it should be tried to minimize this.

Finally, when developing an interface between multiple planning services, the overall system design should evaluate setting it up in a CCSDS conform way. In particular, if both systems use compatible planning languages, also including planning items, resources and constraints into the interface should be considered and is recommendable. Even if not, the interface shall be implemented in a way that there exists a well-defined separation of concerns and any unnecessary duplication is avoided.

References

[CCSDS 2010] CCSDS. 2010. Mission operations services concept. Report Concerning Space Data System Standards (Green Book) 520.0-G-3, The Consultive Committee for Space Data Systems.

[CCSDS 2018] CCSDS. 2018. Mission planning and scheduling. Report Concerning Space Data System Standards (Green Book) 529.0-G-1, The Consultive Committee for Space Data Systems.

[Coelho, Koudelka, and Merri 2016] Coelho, C.; Koudelka, O.; and Merri, M. 2016. Nanosat mo framework: Achieving on-board software portability. In *SpaceOps 2016 Conference*, SpaceOps Conferences. American Institute of Aeronautics and Astronautics.

[Evans 2016] Evans, D. J. 2016. Ops-sat: Operational concept for esa's first mission dedicated to operational technology. In *SpaceOps 2016 Conference*, SpaceOps Conferences. American Institute of Aeronautics and Astronautics.

[Göttfert et al. 2016] Göttfert, T.; Grishechkin, B.; Wörle, M. T.; and Lenzen, C. 2016. The link management system for the european data relay satellite program. In *SpaceOps 2016 Conference*, SpaceOps Conferences. American Institute of Aeronautics and Astronautics.

[Göttfert et al. 2018] Göttfert, T.; Wörle, M. T.; Lenzen, C.; and Prüfer, S. 2018. Operating and evolving the edrs payload and link management system. In *2018 SpaceOps Conference*, SpaceOps Conferences. American Institute of Aeronautics and Astronautics.

[Hauke et al. 2012] Hauke, A.; Ohmüller, T.; Häring, U.; and Center, G. S. O. 2012. An innovative monitoring- and control-system at gsoc and weilheim ground station. In *Ground System Architectures Workshop, GSAW*.

[Hauschildt et al. 2014] Hauschildt, H.; Garat, F.; Greus, H.; Kably, K.; Lejault, J.-P.; Moeller, H.; Murrell, A.; Perdigues, J.; Witting, M.; Theelen, B.; et al. 2014. European data relay system–one year to go! In *Proceedings of the International Conference on Space Optical Systems and Applications (ICSOS)*.

[Lenzen et al. 2012] Lenzen, C.; Wörle, M. T.; Mrowka, F.; Spörl, A.; and Klaehn, R. 2012. The algorithm assembly set of plato. In *SpaceOps 2012 Conference*, SpaceOps Conferences. American Institute of Aeronautics and Astronautics.

[Scharringhausen and Beck 2017] Scharringhausen, J.-C., and Beck, T. 2017. Automated procedure based operations for the european data relay system. In *68th International Astronautical Congress (IAC)*.