# The Incremental Planning System – GSOC's Next Generation Mission Planning Framework

Maria Theresia Wörle[*], Christoph Lenzen[*], Tobias Göttfert[*], Andreas Spörl[*], Boris Grishechkin[*], Falk Mrowka[†] and Martin Wickler[‡]

*Deutsches Zentrum für Luft- und Raumfahrt e.V., German Aerospace Center, Münchener Straße 20, 82234 Weßling, Germany*

**The paper at hand presents the new generic framework for automated planning and scheduling in future mission planning systems developed at GSOC (German Space Operations Center). It evolved from the experiences made in past and current projects and the evaluation of internal and external requirements for upcoming projects. In customary systems such as the one used within GSOC's TerraSAR-X/TanDEM-X mission, succeeding planning runs to combine all collected input to a consistent, conflict-free command timeline take place at fix, dedicated points in time, e.g. twice a day. In contrast and as a main difference, with the new system each new input is processed immediately and so a consistent up-to-date timeline is maintained at all times. We show that this approach provides a set of important advantages and new possibilities for spacecraft commanding and user satisfaction. For example, uplink schedules can be flexibly modified due to short-term notifications, or up-to-date, extensive information about the planning state is always available, which means that conflicts can be seen before finally submitting a new request and, if applicable, can be resolved by selecting a suggested solution scenario. The presented system constitutes a generic tool suite which is scalable in performance critical areas, which is configurable to various mission scenarios and which defines a dedicated set of interfaces, specifying the functionality that remains to be implemented by each individual project. The declared goal is that all upcoming GSOC missions will benefit from using the Incremental Planning framework in terms of cost reduction, implementation duration and system robustness.**

## I. Introduction

IN order to serve the TerraSAR-X and TanDEM-X missions, the Mission Planning team at GSOC has set up a fully automated mission planning system, which is in operation since launch of the TerraSAR-X satellite in 2007 resp., with adaptations, the start of the TanDEM-X mission in 2010. Based upon the experiences gathered since, including further desires of the end users and the flight operations team which emerged during operations, we have recently developed our next generation framework for planning and scheduling, called "Incremental Planning System", which shall be presented in this paper.

### A. Disadvantages of Customary Mission Planning Systems

Although e.g. the complex, fully automated TerraSAR-X/TanDEM-X Mission Planning System[1,2,3,4,5] is running smoothly and further projects can be served well using GSOC's existing planning tools, we identified several restrictions we intend to overcome, in addition to the fundamental goal of software engineering to identify generic functionalities and to factor them out from project-specific implementations in order to increase stability and reduce costs of future mission planning systems.

In detail, here are some of the disadvantages found in customary mission planning approaches, in which scheduling is based upon a time schedule:

[*]Mission Planning System Engineer, Mission Operations Department, German Space Operations Center
[†]Head of Mission Planning Team, Mission Operations Department, German Space Operations Center
[‡]Deputy Head of Mission Operations Department, Mission Operations Department, German Space Operations Center

- The plan currently visible for the user is based on outdated information, i.e. the one available at the generation time of the latest timeline.
- The consequences of incoming information (new planning requests, updated constraining conditions such as ground station availabilities, etc.) will be not visible before the upcoming planning run has finished. Unforeseen and undesired side-effects thus cannot be avoided, e.g. the deallocation of other requests.
- The user has to wait for the response from the planning system until he knows whether a request can be scheduled or not before he can consider the ordering of alternative requests. Also the allocation of additional resources such as additional downlink opportunities that would resolve conflicts and avoid the rejection of requests which compete for one or more of these restricting resources then is likely to be too late.
- The last opportunity for submitting new input to be considered (i.e. the order deadline for planning requests or the reception of updates for constraining conditions), has to be significantly before the command uplink time, since all information which accumulated since the last planning run has to be considered in this planning run.
- The system detects changed planning information first during the planning run. In particular, it is unable to prepone the upcoming planning run on short notice, since it is suspended until then. Thus, all circumstances that affect the planning rhythm have to be kept stable at least until the time of the upcoming planning run. For instance, the system cannot automatically cope with a disturbance of the uplink schedule: Consider the situation to have a planning run at 6 am for an uplink at 8 am and with the knowledge that the next but one uplink will be at 7 pm. Then the system sleeps until about 5 pm to trigger the next planning run. In case at 10 am a ground station for uplink at 1 pm would announce availability, the system would not be able to start the planning run at 11 am autonomously. Thus possible support for unforeseen emergency situations with the help of spacecraft data acquired in the afternoon cannot be performed.

Further improvements of the proposed mission planning framework are related to the commanding, where real-term reactions to uplink-failures in combination with in-pass command execution or telemetry feedback inclusion are required. For this purpose the Incremental Planning framework also provides the basis for a sophisticated solution which is described under the topic “Robust Commanding” (see [6] and further explanations in chapter II.A.7).

## B. The Idea of Incremental Planning

In order to address the above mentioned restrictions and improvement possibilities and meet foreseeable requirements of future missions, the decision was made to implement a new generic framework for upcoming mission planning systems, the “Incremental Planning System”.

### *1. The Goals of the Development of the new Framework*

The first key requirement is to facilitate transparency and responsiveness of the planning process. It shall be possible to provide the customer (or the authorized entity) with best possible control over the timeline, which includes immediate response about the current planning state as well as previews for intended requests and even interaction during the conflict resolution process.

Closely related is the requirement for a late order deadline: modifications of the timeline shall be accepted as late as possible.

Another key requirement is the support of ground station schedule adaptations on short notice, including the adaptation of the uplink schedule.

The last system specific requirement is robustness with respect to uplink-failures in combination with in-pass command execution or telemetry feedback inclusion etc.

Furthermore the basic requirement for any good generic software framework shall apply: The new mission planning tool suite shall restrict itself to the generic workflows, defining distinct interfaces which describe in detail what the project-specific code has to implement. These interfaces shall be defined in a way such that as much functionality as possible remains within the Incremental Planning System, without restricting the project’s capabilities. Thereby, the Incremental Planning System shall reduce costs and increase the reusability and reliability of our future planning systems. For this purpose it is set up as a generic framework of multiple components, which allows tailoring the planning system to the specific requirements of the project. Furthermore there exist distinct plug-in interfaces for additional functions, which on the one hand allow invoking code covering unpredictable project-specific requirements and on the other hand exactly define which algorithms the project has to implement for making use of the generic functionalities. This way reuse of the Incremental Planning System is promoted and a standardized way of setting up new mission planning systems is provided. These can then serve fully as well as

semi-automated approaches, as well as different kinds of missions, from LEO to GEO spacecraft, from one single satellite to multiple satellites' constellations.

*2. Incremental Planning vs. From-Scratch Planning*

In favor of transparency and responsiveness, the scheduling engine is no longer started just in time before a new timeline is needed. Also the new plan is no longer created "from-scratch" every time, i.e. almost without consideration of the previous planning run's result for the upcoming planning horizon. Instead the Incremental Planning tool suite is running constantly, incorporating all input immediately after reception into its persisting timeline, the so-called "master timeline". For illustration, see Figure 1.

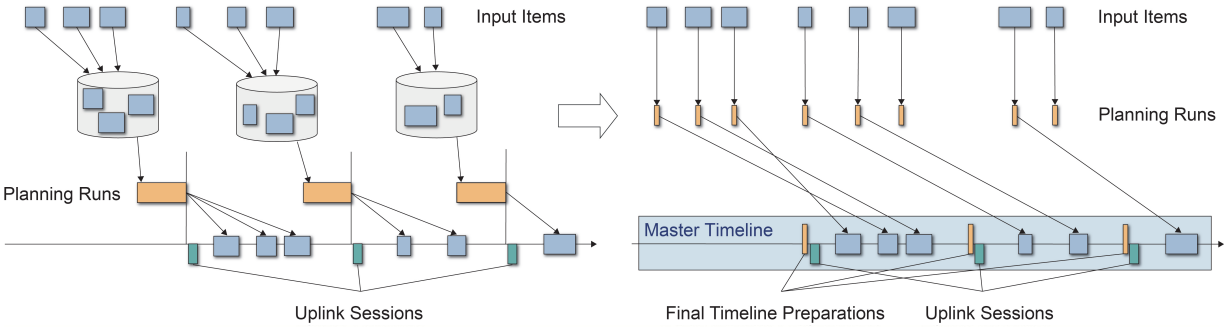


**Figure 1. From-scratch vs. Incremental Planning approach.**

Having a look at other publications, it could be seen that the term "incremental planning" is already used and well-known for the description of planning approaches, but mostly with a different meaning. For instance, compare the definitions and descriptions in [7,8,9,10].

Whereas in other environments "incremental planning" refers to either a break-down of planning problems into smaller entities with a later composition of sub-plans, or a stepwise refinement of plans by accepting approximations first and incorporating more and more precise information later on or doing repair steps, our idea is to have one correct, consistent plan at every point in time. Of course this plan can change over time until it is finally released for execution in a certain time frame, but not necessarily because there had been simplifications assumed in the first calculations, but simply because new, additional input has been received. The only exception from this is the latest orbit information and re-calculated events resulting from it, such as the exact ground station visibilities, of course. Here indeed a refinement takes place. However in general, each new *increment* of the plan is a fully consistent, not approximated solution of the overall planning problem, of which snapshots of the next upcoming timeline parts can be used for export without further scheduling adaptations. Instead of time-consuming planning runs, only a short final timeline preparation in terms of generating spacecraft commands is to be performed then.

A similar meaning of the term "incremental planning" however has been found in [11,12], referring to a continuous adaptation of an existing, consistent plan. In the onboard environment presented therein, as new inputs that influence the plan by triggering a re-scheduling, updated telemetry values are primarily mentioned. The idea of continuous planning interleaved with execution and failure repair has furthermore been detected in [13].

## II. Overview of the Incremental Planning System

An overview of the components of the Incremental Planning System and how they are embedded in a missions' ground segment with their external interface partners can be found in Figure 2. The components marked in green are the core generic tools of the Incremental Planning framework, the ones in blue are optional and depending on the external interface partners and project needs, but have been newly or re-designed with the new approach of mission planning, too. The grey boxes with rounded edges show functions or collections of algorithms whose detailed functionalities are more or less developed and/or composed project-specifically. Nevertheless their role is very important for the whole planning, scheduling and commanding duty of the system on the one hand and very clearly defined on the other hand in order to enable reuse and a fast re-configuration possibility for new projects or changed requirements in one ongoing project.

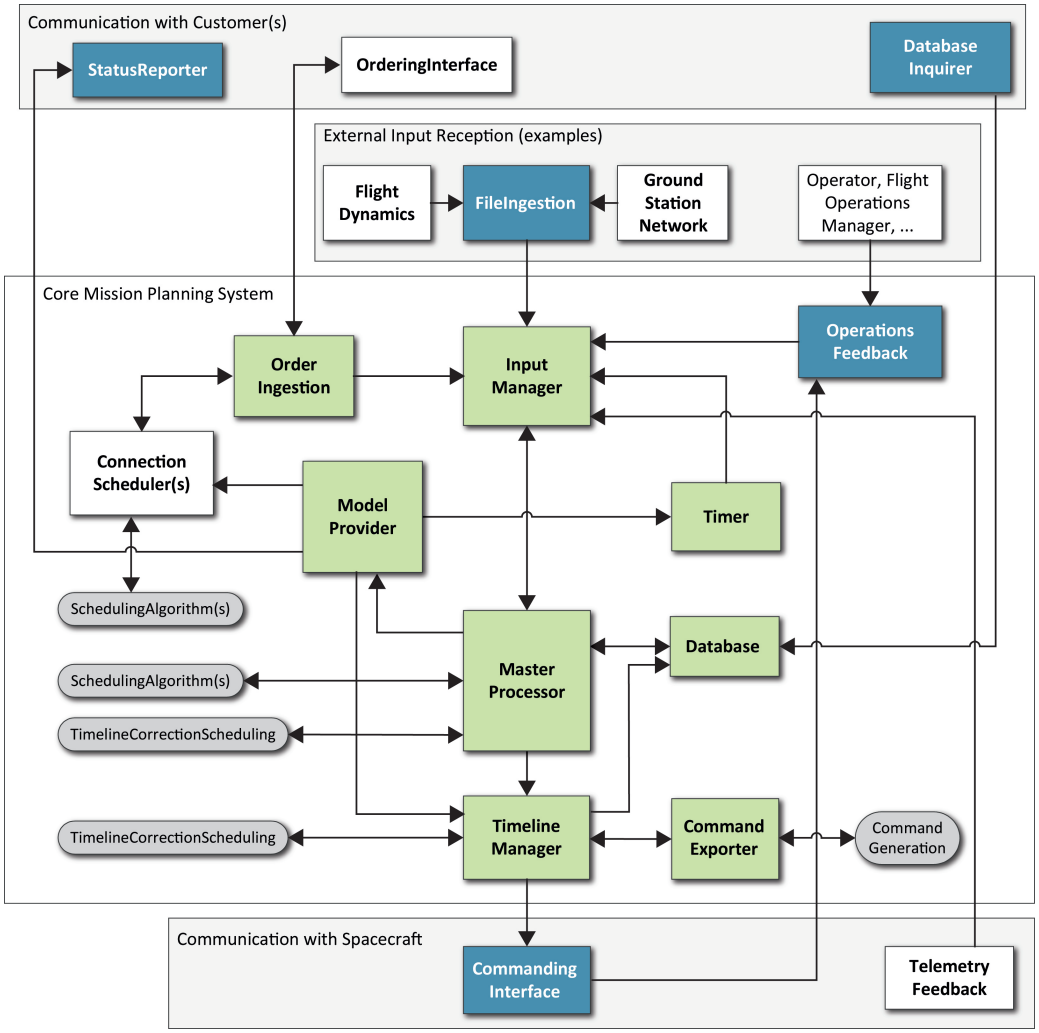


**Figure 2. Overview of the Incremental Planning System components and their interfaces.**

In the following the generic components are presented, with a brief description of their tasks and interaction.

## A. The Core Components of the Incremental Planning System

*1. MasterProcessor*

The MasterProcessor is the central component of the planning system. It gets all input items and processes them one after the other by applying dedicated functions according to the type and content of the input, especially the project-specific scheduling and rescheduling algorithms (see "TimelineCorrectionScheduling" in chapter II.C.2). Instead of creating a completely new plan at several points in time after having received a bunch of inputs, each of the items is separately ingested. In order not to block the execution of important, urgent input, it is foreseen that such new input can lead to aborting the current processing, i.e. aborting of the calculation of the timeline modification for the previous input.

*2. Database*

The Database is always kept in sync with the current state of the model and the master timeline by the MasterProcessor in order to be prepared for a breakdown and a re-start of the system. Apart from the TimelineManager, the MasterProcessor is the only component allowed to write to the database.

*3. InputManager*

The InputManager receives all input items and forwards them to the MasterProcessor. In order to avoid that lower-priority but calculation-time-extensive input blocks input that might have been sent to the planning system

slightly afterwards but is more important, the InputManager sorts the input items according to their priority before forwarding them one after the other, such that the MasterProcessor always processes only one input at a time.

Although most of the time all incoming input is expected to be forwarded to the core processing without any latency, the InputManager is required for peak situations in which a lot of input items reach the system almost in parallel.

Therefore the InputManager is also able to take into account the temporal urgency of an input. For example, the request for a timeline export should have high priority, but nevertheless high-priority order requests might be allowed to be processed first, so that they can still be considered for the upcoming uplink. However when uplink time comes closer, the export will become even more critical in order not to be at risk at all.

If necessary, the InputManager can interrupt the current calculations of the MasterProcessor in favor of processing a new input of higher priority and/or urgency without any delay. Therefore it has to monitor whether the MasterProcessor is currently busy, and, if yes, which item is currently processed by it, too. Then of course, it additionally has to take care that the items, whose processing was interrupted, have to be re-sent after having them queued again.

The prioritization of the different input items and, if applicable, the decision whether to interrupt the MasterProcessor, are done via configurable rules that can be more or less complex and might also rely on the content of the respective input item. For instance only the different input types and sub-types could be differentiated according to simple agreements because their processing takes more or less time, or also the inner priority of planning requests among one another or their final execution time in comparison to the current time might have to be considered in systems with a high workload or very tight time dependencies resp. required response times.

*4. ModelProvider*

As stated above, there is no central database with competing common read and write access where all input information, processing results, timelineentries etc. are collected, read, and modified by the various components. Instead its content and therefore the current project (current model plus current timeline version) is almost exclusively available for the MasterProcessor.

For this, a message-based approach has been chosen: All components which require access to the current state of the model, including the timeline, maintain a private copy of the current project. Whenever the MasterProcessor modifies it (e.g. a new order has been scheduled), the modification is reported to the ModelProvider via so-called changeSets (binary modification messages), which sends the update to all registered components.

As indicated above, each component that needs the current model must register itself on startup within the ModelProvider, which initially supplies the component with the complete model. Afterwards, the ModelProvider only forwards incoming modification messages from the MasterProcessor to each of the registered instances. By applying these changeSets one after the other to the saved latest version of the project, the components keep their copy at any one time synchronized with each of the modifications performed by the MasterProcessor.

This ingestion of the changeSets of course has to be done by the ModelProvider itself, too, in order to be prepared for the next request for the full serialized project of one of the other tools. So in the end, the ModelProvider is the only component which gets the serialized planning model and modification messages directly from the MasterProcessor and takes care of the consistent and fast distribution at each point in time.

*5. OrderIngestion*

This component serves the ingestion of input from customers and other users, “the outside world” of the Mission Planning System. The OrderIngestion in general receives planning, cancellation and update requests from the customers, which can be end-users themselves or intermediate coordinator users. It also checks them for consistency, and generates responses for the customer. Depending on the project’s or specific user’s needs and/or rights this response can vary from only a reception notification to a simple answer about the consistency to the more or less complex result of a test-planning feasibility check performed by a ConnectionScheduler (see II.B.1).

In the most sophisticated version, the OrderIngestion sends a request to the ConnectionScheduler. In case the new order would require other, already scheduled orders to be moved or removed from the timeline, the ConnectionScheduler will answer this request with one or multiple solutions that are forwarded to the user by the OrderIngestion. In case the user selects one of these solutions, it is included in the request and forwarded to the InputManager. For more information see chapter II.B.1.

The OrderIngestion component or its functionalities can either be standing alone, being invoked by or being directly embedded in an OrderingInterface, depending on whether such a communication and/or visualization interface exists for the mission, and then, furthermore depending on whether this is an external tool or web-service or GSOC’s own so-called Swath Preview and Ordering Tool (SPOT).

*6. Timer*

The Timer takes care that certain activities are performed at the correct point in time. For example, it can signal that the time has come for exporting commands before an upcoming uplink session, or that some time before the earlier order deadline has been reached when some final preparations of the master timeline might have to take place. Furthermore, on request from the InputManager, the Timer has to provide a configurable list of upcoming events like the ones just mentioned in order to enable the InputManager to sort incoming input according to timing specifications and dependencies.

For these functionalities the Timer has to have access to the current state of the project including the master timeline with the timelineentries of the respective events, which is kept up-to-date via the ModelProvider. Whereas the Timer in a customary fully autonomous mission planning system such as the TerraSAR-X/TanDEM-X system 'wakes up' the other components at dedicated fix events, here it dynamically adapts its list of events and the times to trigger other components dynamically along with the modifications that are made on the master timeline by the MasterProcessor.

In general, the Timer is the only component of the whole tool suite that has to/is allowed to have access to the system time. Additionally it supports a simulation-time mode for test purposes.

*7. TimelineManager and CommandExporter(s)*

Whenever the appropriate points in time for preparation of the next uplink session are reached the TimelimeManager is triggered by the MasterProcessor with a notification about the model and timeline version to export. Via the ModelProvider, it is or has already been provided with the desired state of the model and timeline. Furthermore it has access to the Database to read the latest commanded timeline, i.e. the one that has been successfully commanded to the spacecraft. From this information and with the help of a so-called TimelineCorrectionScheduling algorithm (see II.C.2) it prepares the so-called target timelines for the different possible uplink scenarios of the upcoming session.

If the satellite supports commanding in a transactional manner and the proper repair algorithms are implemented, the different target timelines may prepare for all possible uplink failures in the best possible way. For each target timeline, the TimelineManager calls the CommandExporter with the target timeline and its expected commanded timeline. The CommandExporter derives the commands required to transform the onboard timeline from the expected commanded timeline into the target timeline. The generation of the according command sequences is again to be handled project-specifically, thus a plug-in mechanism assures that the overall functionality as well as the project-specific requirements are met.

When the command sequences have been generated for all target timelines, they are stored in the Database, if configured, and are forwarded to the CommandingInterface, together with the information about in which case to apply which command sequence.

For a detailed description of the functionality and complex possibilities with which the TimelineManager might serve a project, see the separate paper on "Robust Commanding"[6], the novel commanding concept which also evolved from the Incremental Planning design discussions.

**B. Further Components that can/should belong to an Incremental Planning System**

The following components are optional enhancements for the Incremental Planning framework but are not necessary for the core functionality. However they make use of the opportunities provided by this scheduling approach in order to either provide the users or the spacecraft commanding with special services that are based on it.

Especially the first two components play significant roles for achieving resp. utilization of the full range of advantages the functionality of Incremental Planning as described provides.

*1. ConnectionScheduler(s)*

Before or instead of forwarding a new planning or update request to the InputManager, if specified, the OrderIngestion can initiate a test-ordering to check the availability for or consequences of scheduling this request and provide the user with special interaction possibilities.

In general, the specification whether the ConnectionScheduler shall be invoked for a request and – if yes – how it shall process it and what information the response shall contain, can be pre-configured or can be provided by the content of each input item, so that it has to be evaluated by the OrderIngestion and/or ConnectionScheduler. For instance, it can vary how the ConnectionScheduler shall be invoked or how the results of the ConnectionScheduler run shall be handled/forwarded/filtered by the OrderIngestion. Differences can be made either on the type and content of the request, or for/from which user (group) it comes, who can have different interaction and information access rights.

The result given by the ConnectionScheduler can vary from telling

- whether the new request can be scheduled under the current circumstances mapped in the current project without removing already planned competing requests, or
- the reason why the new request is currently assumed to be not schedulable, e.g. the resource whose maximum value is reached, in case its priority is too small to oust other requests, up to
- an enumeration of so-called "deconflicting solutions", i.e. scenarios which would enable scheduling the new request without any conflict, at the expense of other requests.

The last possibility means that a list of alternative deconflicting solutions is determined by the ConnectionScheduler based on the current project state. Each solution contains the information where the new timeline entries would be placed and what other timeline entries need to be removed or where they have to be moved so that the new request could be scheduled without conflict. In case of resource conflicts, multiple solutions may be calculated. The ConnectionScheduler will return a reasonable set of solutions to the OrderIngestion, which in turn may allow the user to select one of these solutions. According to a user's wishes on the one hand and access rights on the other hand, the rules on which competing requests may be dismissed, the algorithm complexity for searching such solutions, and the number of solutions to be displayed, have to be configured. For simple models, configuring a generic algorithm may be sufficient (see [15]), otherwise the implementation of a specialized, project-specific algorithm might be necessary, see also chapter II.C.

When the user has selected one of the suggested deconflicting solutions, it is added to the planning request and will be considered by the MasterProcessor when finally scheduling this input item later on. Note that then the complex scheduling task has been performed by the ConnectionScheduler and the MasterProcessor will be presented with a proper solution to ingest, so its only task is to follow the instructions of the solution and to re-propagate the resulting resource modifications. Besides, in order to enhance the responsiveness and shorten the response time, there can be more than one instance of the ConnectionScheduler component/service kept available and running in parallel and independently of one another. Due to the design of the ModelProvider, the number of ConnectionSchedulers can be increased by launching further instances within the running system. So the decision whether a further copy is needed can be made flexibly and on relatively short notice in a mission, e.g. for general speed-up reasons or to provide special services for a special user group via another configuration.

Furthermore, the ConnectionScheduler(s) cannot only be used to supply the best possible user interaction. It can also be the core concept of scalability for the whole Incremental Planning System, because, even if no customer interaction is required and implemented, multiple ConnectionSchedulers can be in charge to find a proper solution for each planning request via pre-configured scheduling rules and algorithms before forwarding it to the InputManager. It is even possible to launch or shutdown ConnectionSchedulers within the running system, allowing to adapt the planning system to an increased workload and to replace the servers hosting the ConnectionSchedulers.

Of course this concept suffers from the drawback that contemporaneous requests interact on a model which does not reflect the modifications the concurring requests might imply. However, no better possibility exists for such a case than optimistic concurrency. Also the users selecting a deconflicting solution always have to be aware that the basis for the given information and calculations and so their interaction decision might have already become outdated by contemporaneous ingestion of other input items into the master timeline. Exceptions could be made for the case of high-priority orders, e.g. to support emergencies. For these, the ConnectionScheduler can be bypassed, i.e. the MasterProcessor will be blocked until processing of the high-priority order has completed.

*2. StatusReporter*

The StatusReporter has access to the current project state via the ModelProvider. According to project-specific configuration, it filters the binary modification messages of the ModelProvider and sends relevant information to dedicated interface partners via xml files. For instance, it can notify the customers about every change of the planning state of their requests or can send information to the ground stations or processing facilities about when which data will be dumped.

Here again, a parallelization of more than one instance of the StatusReporter, each with another configuration filtering for different modifications of the timeline or for modifications on different objects, would help reducing the system response time and would be easily practicable if becoming necessary sometime.

*3. FileIngestion*

This component allows the ingestion of further input from "the outside world", i.e. project segments beyond the core Mission Planning System. While the OrderIngestion serves the reception of input items from payload data customers, the FileIngestion is used for the reception of ground segment internal input such as notifications about ground station availabilities, orbit information, maintenance and maneuver announcements, etc. The FileIngestion is needed to perform pre-configured consistency checks and the transformation of these input items into a common format, so that the forwarded content is included in a file readable by the InputManager (and MasterProcessor later on).

*4. CommandingInterface*

The CommandingInterface is provided to the commanding system in order to enable sending the sequence of adequate, applicable command sequences in real-time according to the effective uplink scenario during the ground station contact(s), see also explanations in [6].

*5. OperationsFeedback*

The OperationsFeedback enables manual or autonomous ingestion of information about the past or future of the system, e.g. the success reports of commanding sessions or the notification about planned maintenance phases.

*6. DatabaseInquirer*

Depending on mission requirements, the DatabaseInquirer can provide the same information as the StatusReporter, but not autonomously as a reaction to changes in the current project, but as a response to requests from interface partners. Therefore it might also keep track of the current planning state and model content (informed via the ModelProvider). However, there can be further use cases such as the generation of reports and statistics which require not only the insight into the current project but into previous versions of the model and master timeline, too, so this component also needs read access to the Database for more complex queries.

### C. Interfaces to Project-Specific Code

The Incremental Planning System constitutes a framework of tools that shall factor out the commonalities of mission planning systems. As already mentioned above, it therefore does not include explicit algorithms wherever these are dependent on the particular project and its current requirements. Instead it defines dedicated interfaces where the project-specific code shall be invoked. These interfaces specify which functionalities are expected to be implemented and therefore may be seen as a roadmap for each project when using the Incremental Planning System and shall allow full flexibility with respect to their implementation.

*1. CommandGeneration Algorithm(s)*

Given the currently commanded timeline and the desired timeline prepared on ground, the project-specific code invoked by the CommandExporter must generate a sequence of commands which converts the tele-commands on board of the to-be-commanded unit(s) into the desired state of tele-commands.

*2. Scheduling and TimelineCorrectionScheduling Algorithms*

The collection of scheduling rules and algorithms as well as the correspondent rescheduling algorithms that are applied by the different components are also tasks for the project-specific implementation.

- *SchedulingAlgorithm(s) of the ConnectionScheduler*
  This algorithm is given a new planning request and, depending on the project's requirements, it returns the information whether the new request can be scheduled, why it cannot be scheduled, what decisions would have to be made to schedule the request, or what deconflicting solutions can be presented to the user, see B.1.
- *SchedulingAlgorithm(s) of the MasterProcessor*
  For each possible input item this algorithm has to adapt the project accordingly. Such input may be allowed to be rejected if unfeasible, i.e. not being schedulable without causing resource conflicts, e.g. planning requests with or without pre-selected deconflicting solution, or it may not be rejected at all, e.g. new orbit information. In this case, the algorithm must adapt the timeline to the reality by removing all conflicts. The project should be set up in a way that this is always possible.
  Of course, the more complex the scheduling rules, with a priority-based approach and a lot of constraining resources to be considered, the more extensive can be the re-scheduling necessities when trying to insert the next input. According to the respective project needs and/or timing demands, for instance optimization functions might have to be introduced, or algorithms might have to be found that avoid a cascading or alternating scheduling and re-scheduling of requests on the timeline.
- *TimelineCorrectionScheduling of the MasterProcessor*
  This algorithm has to adapt the timeline according to the feedback about the previous uplink session. In the nominal case (all uplinks succeeded), this algorithm does not modify the timeline. Otherwise it has to repair it and remove conflicts according to what has been commanded or failed. For multi-satellite projects a complete deconflicting might not always be possible, however it has to assure the safety of the mission.
- *TimelineCorrectionScheduling of the TimelineManager*
  This algorithm is given the source timeline, i.e. the one assumed to be already commanded, the master/target timeline and a time interval. It then has to calculate a modified version of the source timeline, which equals the target timeline during the time interval and possibly takes over further parts of the target timeline or other activities outside this time interval in order to keep the timeline conflict-free. The TimelineManager uses this interface in order to prepare the different uplink scenarios[6], see also A.7.

As one can see, these interfaces impose remaining challenges when setting up a new project. However it can also be chosen not to implement all of these interfaces completely, but nevertheless these interfaces show what has to be considered, no matter whether a fully automated resolution shall be implemented or not. For example it might be preferred to use back-up uplink passages and consider the case of a failed uplink pass as a contingency. Then the TimelineCorrectionScheduling of the TimelineManager may be omitted, since by configuration it will not call this interface.

In order to cope with the complex requirements, all the algorithms, command generation as well as scheduling and TimelineCorrectionScheduling algorithms resp. algorithm collections, will be designed and configured specifically for each project to be plugged in into the according MasterProcessor, ConnectionScheduler and TimelineManager component, where applicable and foreseen.

However, even these algorithms will still be not implemented completely from scratch but by using the wide range of capabilities provided by GSOC's generic Plato library for the problem description in its certain modelling language [14], as well as for composing and configuring the complex algorithms from an extensive, robustness-proven collection of variable, configurable basic algorithms and filters. The beginnings of the latter approach have been first described in [15]. In the meantime this collection has been continuously extended, enhanced and made more flexible in order to handle a wide range of scheduling problems.

Since customer wishes do not only differ from mission to mission, but very probably change during the life time of a mission, too, especially here the possibility to reconfigure and flexibly include another scheduling algorithm via a plug-and-play interface is very valuable.

### D. Interfaces to further Project-Specific Tools

In addition to applying project-specific algorithms via the interfaces described in II.C, further project-specific tools can also be easily integrated to support special mission requirements. On the one hand, this can be done by using the functionalities of the ModelProvider whenever information about the current model and planning state shall be retrieved and included in further functionalities. On the other hand, in case additional input has to be considered for the timeline generation, either an additional direct interface to the InputManager can be set up, or the input items are sent to the system via the FileIngestion if consistency checks and/or a translation to the common input format have to be performed first.

For instance, it is thus possible to include additional possibilities for user interaction into the Incremental Planning System by attaching a GUI which allows modifying the timeline manually. This will be implemented e.g. to support mission preparation phases, LEOPs or other special planning and scheduling problems that won't be handled by fully-automated systems by inclusion of PINTA, GSOC's Program for Interactive Timeline Analysis, as a user frontend.

Another example is the ingestion of telemetry feedback from the spacecraft in a project with rather unpredictable onboard behavior. The consideration of information about what exactly happened onboard in the planning system might especially be desired in case of spacecraft mission planning and scheduling autonomy. With a concept such as described in [16,17,18], in which the decisions which actions to perform by one or more of the instruments of the satellite is partly made on-ground and partly made onboard embedded in the payload software according to telemetry measurements or triggered by dedicated detected events, a synchronization of the two information and decision branches from time to time can be beneficial and required.

## III. Summary of Design Ideas of the new Framework

Besides the advantages for each mission that result from addressing and overcoming the whole set of drawbacks of customary mission planning systems as enumerated in chapter I.A, the described design provides the following positive aspects:

### A. Component Interaction and Interfaces

In order to make the system scalable, responsive and thread-safe, it was decided to set up the Incremental Planning System as a distributed set of different components with a message-based communication in between. The interaction and distribution of information via messages has already been described and reasoned above.

Whereas the internal interfaces inside the Incremental Planning System have been strongly typed, the external interfaces have to be kept sufficiently flexible in order to allow an application to plug in into the Incremental Planning System without recompiling the generic tools on every change of an external interface.

Nevertheless, there can be restricting project-specific interface needs (e.g. parameters of an acquisition request, notified events from flight dynamics, as well as to be reported information via status reports etc.). The consistency of the content of received/to be sent information then has to be ensured by project-specific configurations for the components that serve these external interfaces (e.g. Order- and FileIngestion, StatusReporter), e.g. by specifying an XML schema the input has to obey.

### B. Configurability and Reusability

All of the different components of the Incremental Planning System provide a huge amount of configurability according to various project requirements. Based upon the knowledge of more than 20 years of mission planning at GSOC, the Incremental Planning tool suite aims to factor out the common functionalities of different mission planning scenarios. Although future missions may show that minor adaptations may still be required, we expect the Incremental Planning framework to evolve to a stable version very soon. Explicitly no project-specific versions of the tool suite shall be created then, but the goal is to perform such improvements on the generic components and enable more possibilities for configuration with every new, not yet considered/imagined challenge so that the whole product is stepwise enhanced and becomes applicable and sufficient for an increasing variety of projects and applications. This way not only implementation effort may be saved but also a roadmap for implementing new mission planning systems is defined, which will simplify system engineering, too.

For the remaining, project-specific code, GSOC supplies a generic tool suite which integrates well into this framework and which should be sufficient for the requirements of a standard mission.

One example for this are algorithms developed from the generic Plato library[15] (see detailed explanation at the end of chapter II.C) which will be included for the core functionality of the according planning system. The set of algorithms, filters etc. there has also increased and been further developed and continuously extended during the implementation and support of various mission planning projects. As another example, the planning model, reflecting the tasks and resources and constraints to be scheduled, maintained and considered in the according project will be in GSOC's generic planning language[14] and will be mirrored in the Plato database on which the MasterProcessor will be operating.

With customer wishes varying not only from mission to mission, but very probably also changing during the lifetime of one mission, especially the flexibility given by the plug-in interfaces for project-specific code will prove to be very valuable.

### C. Scalability

Except for the core components, which should be part of any Incremental Planning application, the framework is designed such that Mission Planning Systems of various complexities can be served and built.

Several of the described features that the Incremental Planning System can provide to a project can be omitted or switched off by configuration in order to simplify the workflows for a project or application with relatively simple, low-demands requirements. On the other hand, each of the more sophisticated features can be switched on during the operation time of a mission without complex recompiling and releasing new/changed software but only via changing the components' configuration files or even by just adding the respective generic application to the project's tool suite, for instance the StatusReporter or the ConnectionScheduler.

Additionally, when a speed-up in one of the components or concurrent accesses of various users are required, copies of some of the interface-serving components can be started and used in parallel. With the features of the ModelProvider the synchronization is given then. For example, using a range of ConnectionSchedulers to do the time extensive calculations for preparing the scheduling solution for every incoming planning request prior to sending it to the core planning system minimizes the workload on the MasterProcessor, which is the time-critical bottleneck component, since it is not parallelizable itself. This scalability in terms of multiple copies of one of the components might be very important also during a mission's lifetime.

As another example, more than one StatusReporter might decrease the response time to inform about new states. With them being differently configured, they can send information to the different user groups in parallel and so very quickly instead of one after the other. The overall duration difference might be especially significant and useful when e.g. an orbit update has just been processed and leads to slight time shifts of almost all timelineentries for events about which the data recipients and/or ground station operators have to be notified.

### D. Maintainability and Robustness

The distribution of the Incremental Planning framework to several components that have been thoroughly designed leads to a high degree of maintainability and robustness.

Instances of the various tools can be flexibly integrated and removed. Generic issues like the persistency of all collected data are secured by the Incremental Planning System's setup. For most of the components this is provided via the functionalities of the ModelProvider, that enables a re-synchronization with a consistent project state at all times. Furthermore those interfaces are handled carefully by which non-recurring, not-yet-saved-in-the-model/ timeline information is transferred via the content of specific messages, e.g. a planning or cancellation request from the customer via the OrderIngestion to the InputManager and then to the MasterProcessor. Therefore a sophisticated system of data storage has been found and control mechanisms for the communication between the components have been included in their design, so that the whole system should even be able to cope with power breakdowns or hardware replacements and following restarts without any negative side-effects.

## IV. Conclusion and Outlook

As pointed out in the last chapter, the presented "Incremental Planning System" provides a robust, scalable, generic framework with a lot of advantages for the mission planning support of future missions operated at GSOC.

However, it will be interesting to see, which additional demands will be risen as soon as the users have become used to the new tool suite, how the project-specific configurations will have to be made in order to satisfy the respective customer's requirements best, and whether the foreseen plug-in possibilities have been chosen well enough to integrate these specific demands into the system. Furthermore, especially the set-up of the scheduling and re-scheduling algorithm(s) will still impose some future challenges. For example, the necessity to balance the wish for optimally filled timelines and the following of strict priority rules on the one hand and the wish to avoid too frequent changes of once given information about previous planning states and to be able to interact in the decision making process on the other hand.

Nevertheless we are convinced that the Incremental Planning framework with its sophisticated, thoroughly elaborate design, the scalability, configurability and therefore reusability and robustness will result in major benefits.

Currently, the implementation is ongoing after the detailed design of the Incremental Planning System has been finished last year. The first mission which shall make use of it in full complexity will be EnMAP (an Earth observation mission with planned launch date 2016 [19]), but the usage of one or the other component might already take place earlier in other fully- or semi-automated planning applications.

## References


[1]Lenzen, C., Wörle, M. T., Mrowka, F., Geyer, M. P., and Klaehn, R., "Automated Scheduling for TerraSAR-X/TanDEM-X", *7th International Workshop on Planning and Scheduling for Space (IWPSS 2011)*, Darmstadt, Germany, June 8-10, 2011.

[2]Mrowka, F., Geyer, M. P., Lenzen, C., Spörl, A., Göttfert, T., Maurer, E., Wickler, M., and Schättler, B., "The Joint TerraSAR-X/TanDEM-X Mission Planning System", *Symposium Proceedings, pp. 3971-3974, IGARSS 2011*, Vancouver, Canada, July 24-29, 2011.

[3]Geyer, M. P., Mrowka, F., and Lenzen, C., "TerraSAR-X/TanDEM-X Mission Planning – Handling Satellites in Close Formation", *SpaceOps 2010, 11th International Conference on Space Operations*, Huntsville, AL, April 25-30, 2010.

[4]Maurer E., Mrowka F., Braun A., Geyer M. P., Lenzen C., Wasser Y., and Wickler M., "TerraSAR-X Mission Planning System: Automated Command Generation for Spacecraft Operations", *IEEE Transactions on Geoscience and Remote Sensing*, Vol. 48, No. 2, 2010, pp. 642-648.

[5]Maurer, E., Geyer, M. P., Braun, A., Mrowka, F., Lenzen, C., Wasser, Y., and Köhler, A., "TerraSAR-X Short Notice Planning", *SpaceOps 2008, 10th International Conference on Space Operations*, Heidelberg, Germany, May 12-16, 2008.

[6]Göttfert, T., Lenzen, C., Wörle, M. T., and Mrowka, F., "Robust Commanding", *SpaceOps 2014, 13th International Conference on Space Operations*, Pasadena, CA, May 5-9, 2014.

[7]Jonsson, P., and Bäckström, C., "Incremental Planning", in Ghallab, M., and Milani, A.: *New trends in AI Planning, Proceedings 3rd European Workshop on Planning*, IOS Press, 1995.

[8]Drummond, M., Swanson, K., Bresina, J., and Levinson, R., "Reaction-First Search", *Proceedings 13th International Joint Conference on Artificial Intelligence (IJCAI '93)*, pp. 1408-1414, Chambery, France, 1993.

[9]Sarkarati, M., "A Generic Science Operation Planning Concept for Planetary Missions and its Implementation on the First ESA Lunar Mission SMART-1", Dr.-Ing. Dissertation, TU Berlin, Germany, 2010.

[10]Damiani, S., Dreihahn, H., Noll, J., Niézette, M., and Calzolari, J. P., "Automated Allocation of ESA Ground Station Network Services", *5th International Workshop on Planning and Scheduling for Space (IWPSS 2006)*, Baltimore, MD, October 22-25, 2006.

[11]Chien, S., Knight, R., Stechert, A., Sherwood, R., and Rabideau, G., "Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling", *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, Breckenridge, CO, USA, April 14-17, 2000.

[12]Chien, S., Knight, R., Rabideau, G., Estlin, T., Sherwood, R., and Stechert, A., "Using Incremental Replanning to Improve the Responsiveness of Planning and Scheduling", URL: http://www-aig.jpl.nasa.gov/public/papers/hw/ieeecasper/CASPER journal.doc [cited 24 March 2014].

[13]Myers, K. L., "Towards a Framework for Continuous Planning and Execution", *Proceedings of the AAAI Fall Symposium on Distributed Continual Planning*, 1998.

[14]"Planning Modelling Language", URL: http://www.dlr.de/rb/en/Portaldata/38/Resources/dokumente/gsoc_dokumente/mb/GSOC_Modelling_Language.pdf [cited 31 March 2014].

[15]Lenzen, C., Wörle, M. T., Mrowka, F., Spörl, A., and Klaehn, R., "The Algorithm Assembly Set of Plato", *SpaceOps 2012, 12th International Conference on Space Operations*, Stockholm, Sweden, June 11-15, 2012.

[16]Lenzen, C., and Wörle, M. T., "Onboard Planning and Scheduling Autonomy within the scope of the FireBird mission", *SpaceOps 2014, 13th International Conference on Space Operations*, Pasadena, CA, May 5-9, 2014.

[17]Wille, B., Wörle, M.T., and Lenzen, C., "VAMOS – Verification of Autonomous Mission Planning On-board a Spacecraft", *19th IFAC Symposium on Automatic Control in Aerospace*, Würzburg, Germany, September 2-6, 2013.

[18]Wörle, M. T., and Lenzen, C., "Ground Assisted Onboard Planning Autonomy with VAMOS", *8th International Workshop on Planning and Scheduling for Space (IWPSS 2013)*, Moffett Field, CA, March 25-26, 2013.

[19]Müller, R., Bachmann, M., Chlebek, C., Krawczyk, H., Miguel, A., Palubinskas, G., Schneider, M., Schwind, P., Storch, T., Mogulsky, V., and Sang, B., "The EnMAP Hyperspectral Satellite Mission. An Overview and Selected Concepts", *Third Annual Hyperspectral Imaging Conference, Instituto Nazionale di Geofisica e Vulcanologia*, Rome, Italy, May 15-16, 2012.