Automated Scheduling for TerraSAR-X/TanDEM-X

Christoph Lenzen, Maria Th. Wörle, Falk Mrowka, Michael P. Geyer
German Space Operations Center, DLR Oberpfaffenhofen, 82234 Wessling, Germany
Rüdiger Klaehn
Heavens Above, Pfingstrosenstrasse 2, 81337 München, Germany

Abstract

In 2007 the satellite TSX-1 of the mission TerraSAR-X has been launched. Its primary payload is an active radar instrument, which shall supply radar images on request for commercial and scientific users. With a maximum load of up to 1000 datatake requests per day and an order deadline of six hours before uplink, the command generation process had to be fully automated. The complexity of the satellite and the evolving knowledge of its constraints exposed further challenges on implementation of the scheduling process.

In 2010 a copy of TSX-1 has been launched: the TDX-1 satellite. When operating in close formation (i.e. distances of 250m-400m) this pair of satellites may execute stereoscopic datatakes which allow the generation of a digital elevation model. However several inter-satellite constraints need to be taken into account when operating at such short distances. Additionally we have two distinct missions, which need to be merged together: First, we have the old TerraSAR-X mission, for which the customers usually ingest their high-priority orders very late, for example for disaster monitoring. Second, we have the TanDEM-X mission, whose goal is a complete coverage of the earth in 3D with best possible accuracy. For this mission the datatakes are calculated up to one year in advance.

This paper shows what techniques have been developed in order to cope with the challenging requirements of combining two missions and handling a two-satellites-system.

TerraSAR-X planning problem

The TSX-1 satellite carries a newly developed SAR instrument whose integration in the bus was not trivial. Therefore commanding of the satellite has to be done at a rather low level. The most important tasks to schedule are:

- file creation
- instrument wake up from sleep-level
- datatake execution
- instrument go to sleep-level
- downlink
- file deletion
- antenna mode switching (there exist two types of datatakes, for which the memory module needs to be reconfigured)
- attitude mode switching (the radar instrument needs a certain view angle, therefore the satellite has two attitudes which allow data taking: left- and right-looking)

A lot of constraints in between tasks of same as well as different types exist. The main ones are listed in the following; all together we have about 70 constraints:

- Uplink, datatake and downlink have an individual but fix duration. Downlinks may be split. All other tasks have a task specific fix duration.
- Uplink before file creation, file creation before datatake, etc.
- No two datatakes at the same time.
- No file creation and file deletion in parallel to datatake or downlink.
- No more than nine telecommands during one second (during datatake commanding we have up to 7 commands).
- No more than 55 file deletions in parallel. File deletions which overlap must have same start time, so they are merged to one telecommand
- Partial file deletions must not overlap with other file creations or (partial) file deletions at all
- Limited on-board memory
- In between two datatakes the instrument has to be set into a certain sleep-level. The depth of this sleep-level depends on the gap size:
  - Gap size < 15sec : Sleep-level 0 (SL0)
  - Gap size < 60sec : Sleep-level 1 (SL1)
  - Gap size < 15min : Sleep-level 2 (SL2)
  - Gap size >= 15min: Sleep-level 3 (SL3)
- Nominal datatakes need right-looking mode, left-looking datatakes need left-looking mode.
- Maximum duration in left-looking mode during one maneuver: 170sec
- Duration of turning to left-looking and turning back: 250sec
- 15min after start of turning to left-looking mode the satellite has to be in right-looking mode.
- During each time window of size one orbit, no more than 180sec of data taking is allowed.

- Replay only during visibility of the respective downlink station
- X-band-transmitter must be switched on during replay.
- During each time window of size one orbit, X-band-transmitter must not be on for longer than 2800sec.

TerraSAR-X mission goals

The TerraSAR-X mission has been established as a public private partnership, consisting of the public partner, who represents the scientific community, and the private partner, whose concern is the commercial exploitation of the satellite. You can find a detailed description in [1, 2].

The TSX-1 satellite has been designed for about 500 datatakes per day, which means that Mission Planning has to consider about 1000 datatakes per day, since not all datatakes will be schedulable. Obviously Mission Planning had to be automated.

Most of the requested datatakes are known several days or weeks in advance. However one major goal of the TerraSAR-X mission is to supply data for current urgent events, in particular for disaster monitoring. A main requirement therefore is a quick response to incoming orders, which we achieve by generating a new timeline for each uplink session, i.e. twice a day. Order deadline for incoming orders usually is six hours before the respective uplink passage.

In addition to the six hours deadline, a special service has been requested to allow modifications of already commanded datatakes via additional ground stations. The deadline for these updates is reduced to one hour before uplink.

Algorithm

The commercial and the scientific user groups need to share the satellite. First challenge therefore was to find a suitable mechanism of how to distribute the satellite resources. A sophisticated mechanism of pricing orders with respect to priority, resource consumptions and time criticality has been developed, together with a simple priority based greedy algorithm, see [2, 3]. However management decided that pricing of orders would not be necessary. Fortunately the user groups managed to arrange with each other, in particular because commercial success was known to result in a successor satellite for both parties.

Thus the base algorithm is quite simple:

1. use the timeline of the preceding planning run as starting point
2. for each scheduled datatake, sorted by (low priority first / late order date first):
   a. if possible, unschedule file creation, datatake, downlink and file deletion
   b. if a. is not possible, try to unschedule datatake and downlink, move file deletion to the earliest possible time.
3. for each not scheduled datatake, sorted by (high priority first / early order date first):
   try to schedule the datatake, together with downlink, file deletion, file creation, uplink and sleep-level switchings

This approach does neither optimize the sum of priorities nor the number of datatakes, but it has two major advantages:

I. For each unscheduled datatake, we can supply a list of more important, scheduled datatakes, with which this datatake would be in conflict.
II. The algorithm is fast enough to obey the six hours deadline.
III. For the one hour update, we can restrict steps 2 and 3 to the places of the desired modifications

Although the base algorithm is quite simple, there are several special features, which had to be implemented:

Sleep-level switching

Initially, the sleep-level transitions should have been added after scheduling all datatakes. However there exist constraints in between sleep-levels and datatakes, which may cause a conflict in certain situations. Therefore scheduling of sleep-level transitions had to be added into the main algorithm.

For this purpose we reused an update mechanism deep inside the scheduler: You can let an update method be triggered whenever a certain modification takes place. In this case the modification is a new or deleted datatake timeline entry. This modification causes the surrounding sleep-level switchings to be adapted and checked to be conflict free – if not the datatake is rejected.

Downlinks

The downlink of a datatake may last for a few minutes. We therefore have to support splitting it into several fragments. However the last segment of a preceding fragment must be repeated at the beginning of the succeeding fragment, in order to allow the two pieces to be merged properly.

Furthermore, we have a set of allowed downlink station groups for each datatake request. The scheduler shall choose the best downlink station group and schedule all downlink fragments for the ground station opportunities of this group.

In order to find the best ground station group, the scheduler tries all of them and chooses the result with earliest end time of the last downlink fragment end. This assures that memory on board the satellite is deallocated as early as possible.

File deletion

According to the satellite hardware configuration, a file deletion must not take place during a datatake and it must not take place during a downlink. However, when scheduling the first datatake, there would be no constraint indicating that the fileDeletion of this datatake can not be scheduled during the opportunity of the second datatake. The algorithm therefore may place the fileDeletion such that the second datatake can't be scheduled any more.

We know that there is enough time for file deletion, because a constraint exists not to activate the instrument for more than 700sec per orbit (~95min). We therefore decided to pick certain times in advance. File deletion will only take place at these dedicated times. Usually these times are chosen right behind the downlink stations, and in any case such that no downlink stations are visible and such that preferably no datatakes are visible. In the last three years we never had a case where a low priority datatake was blocked due to this mechanism.

Although we have similar constraints with file creation and downlinks, we do not have them with file creation and datatakes. Therefore a preprocessing as for file deletions is not necessary.

Backtracking

The base algorithm itself calls for a backtracking technique: in case the algorithm detects the unfeasibility of a datatake when some timeline entries have already been generated. In case of a failure due to sleep-level switchings, all datatake specific timeline entries already exist, and all of them must be removed. Even in case of no failure, we use backtracking when scheduling the downlinks: As mentioned above, we try out all downlink groups and select the best one.

Although we are coding in a conventional object oriented language (.NET/C#), the core operations are strict functions without side effects. At a higher level, we strictly distinguish between the model objects and their states. For example the resource 'Power' is a model object, whose identity will never change, but its resource profile will be modified together with the timeline. The current state of the resource at a certain time is given by an immutable structure. Inside this structure the resource profile is stored, together with other properties such as name and parameters. As an element of an immutable structure, a resource profile must be immutable, too. Of course, the resource profile of Power has to be modified whenever a power consuming task is scheduled. But this is done by replacing the immutable state of Power with a shallow copy, where only the resource profile is replaced by the result of the calculation.

All entities of our modeling language (tasks, groups, resources and the project) are built up this way and are derived from a special base class, which supports the following transactional semantics:

```
using(trans = new Transaction())
{
   bool success;
   // here you can try your subpath and set success
   if(success)
      trans.Accept();
}
```

When generating the *new Transaction()*, we save a snapshot of all current states of all model objects. Due to the fact that these states are immutable, we only need to save the references to these states. Inside the using block, we may continue our algorithm, e.g. by adding timeline entries or constraints. Before we exit the using block, we have to decide whether to take over the modifications we have done or to roll back to the state at the beginning of the transaction. In order to take over the modifications, one has to execute the 'Accept()' method of the transaction. In order to roll back one can execute the 'Reject()' method. Similar to database transactions, default is to roll back. This allows wrapping the transaction block into a try-catch block and to be sure that in any exceptional case the current state of the system is well defined, namely the one before entering the transaction.

This approach is similar to 'Software Transactional Memory', see [6, 7].

In our planning software, we use this transaction together with a *NoSolution* exception. This *NoSolution* can be thrown inside a subpath and caught outside the *using(new Transaction())* block. For example we can use the following pattern:

```
try
{
   using(trans = new Transaction())
   {
      bool success;
      // here we try to schedule the datatake, including
      // uplink, file handling, downlink, sleep-levels
      ScheduleDatatake(dt);
      trans.Accept();
   }
}
catch(NoSolution ns)
{
   // log the reason why the NoSolution has been thrown
   // the reason is supplied in ns.Message
}
```

Whenever the *NoSolution* gets thrown inside the *ScheduleDatatake* method (e.g. caused by failing to adapt the sleep-levels), we exit the *using(trans = new Transaction())* block and all modifications inside *ScheduleDatatake* are discarded.

Although throwing exceptions in nominal workflow should be avoided, we do not face a performance issue here, because the effort to schedule a datatake is by far greater than throwing and catching an exception.

Modification handling

As mentioned in the previous section, we use a certain mechanism in order to assure execution of code whenever something happens which we want to observe. This has some resemblance with event handling. However it is not quite the same. With events, you usually get notified when something happens on a different thread. Here we want to have a piece of code being executed at once when something happens. The thread in which it is executed must wait until the code returns.

The separation of model object and its state allows this kind of monitoring quite easily: Whenever a model object of the scheduler gets a new state (e.g. because a timeline entry is added to a task or a resource is given a new resource profile), a certain list of modification handlers is processed. There exist predefined modification handlers, such as the one which recalculates the resource profiles when a timeline entry is added or a new constraint is defined. But you can also add modification handlers yourself, e.g. in order to keep the sleep-level switchings up to date.

Multithreading

With respect to performance, this semi-functional approach proved to be sufficiently fast. The CPU load is absolutely irrelevant compared with the time used for the dominating part, the execution of the resource profile calculations. It even allows a very good multithreading approach:

Whenever a new thread is started, a copy of the current state is generated (as the states themselves are immutable, nothing has to be copied except for the mapping of objects to their states). The new thread now can execute its operation. When finished, the set of modifications can be incorporated into the state of the parent thread or when multiple threads have been executed in parallel, one can select in between different modes:

- Merge calculations, i.e. take over all modifications. An exception would occur if the same object was modified by more than one thread.
- Use best result, i.e. check which result is better and only use this one's modifications.

The multithreading concept itself proved to work without problems; however our current implementation of profiles is based on a tree structure, which means that small memory allocations are executed extremely often. The garbage collector of .NET 3.5 unfortunately is single threaded; therefore parallelization didn't show the expected performance benefit. To solve this, we are currently working on an alternative implementation based on arrays.

How to hit a moving target

When designing and implementing the TerraSAR-X mission planning system, we faced the problem that many details of the satellite were not yet known. Many changes had to be incorporated during the project's implementation phase, and even in the early operational phase, new constraints occurred, not to mention adaptation of constraints.

When designing the scheduler for the TerraSAR-X mission, we had two possibilities, either to implement a lightweight specialized tool, which can only serve this mission, or we could refactor our existing planning tool suite to match the requirements of the new mission. Fortunately we decided to refactor our generic planning tools. The GSOC modeling language, which is used in these tools, strictly distinguishes between the structure (tasks, which shall be scheduled and grouping of tasks), the resources (state of the modeled system), the constraints (how the tasks interact with other tasks and resources) and the algorithm. Therefore most of the newly appearing requirements could be implemented just by adding a new constraint. Sometimes we needed to add a new resource. Both modifications did not affect the algorithm or other components of the Mission Planning System (MPS) at all, so no great effort was involved in this place. Modifications of other parts of the MPS were necessary only when we had to introduce a new task, e.g. a file creation. For a description of the modeling language, see [5].

Of course there existed the need to adapt certain parameters during the mission. In order to serve this requirement, we started with a configuration file, in which we stored all values, for which we supposed that they might get modified in course of time. However this proved to be insufficient: the configuration file grew large and the exact meaning of the different values became hard to understand for all but the developer. Furthermore new constraints could not be defined, because only existing values could be adapted. This led to a wide range of software versions.

To solve this, we decided to design an XML schema, which allows defining constraints according to our generic modeling language. An XML file which obeys this schema can be used to define constraints on a planning project inside our planning tools. Almost all constraints of the TerraSAR-X mission have been translated to entries of such an XML file. This means that all of these constraints have been removed from the mission specific code and are now generated according to the XML file. Thus all of these constraints are now completely configurable and removable and even new constraints may be defined just by modifying the XML file.

Of course this means that this XML file itself must be treated with special care – however activation of a modified constraint specifications file is much less time consuming than recompiling and installing a new software version.

We still have some parameters, which do not concern the constraints themselves. But we have managed to ban the volatile part into almost full configurability.

Challenges of TanDEM-X scheduling

The TDX-1 satellite and its mission were added to the TSX-1 satellite and its mission three years after launch of TSX-1. For details on the combined mission and the technical needs for the following, see [4]. When starting the commissioning phase, we had two satellites in far formation. At this time the only additional complexity was to select one satellite per data take. However as the satellites approached, different scenarios with different constraints had to be considered:

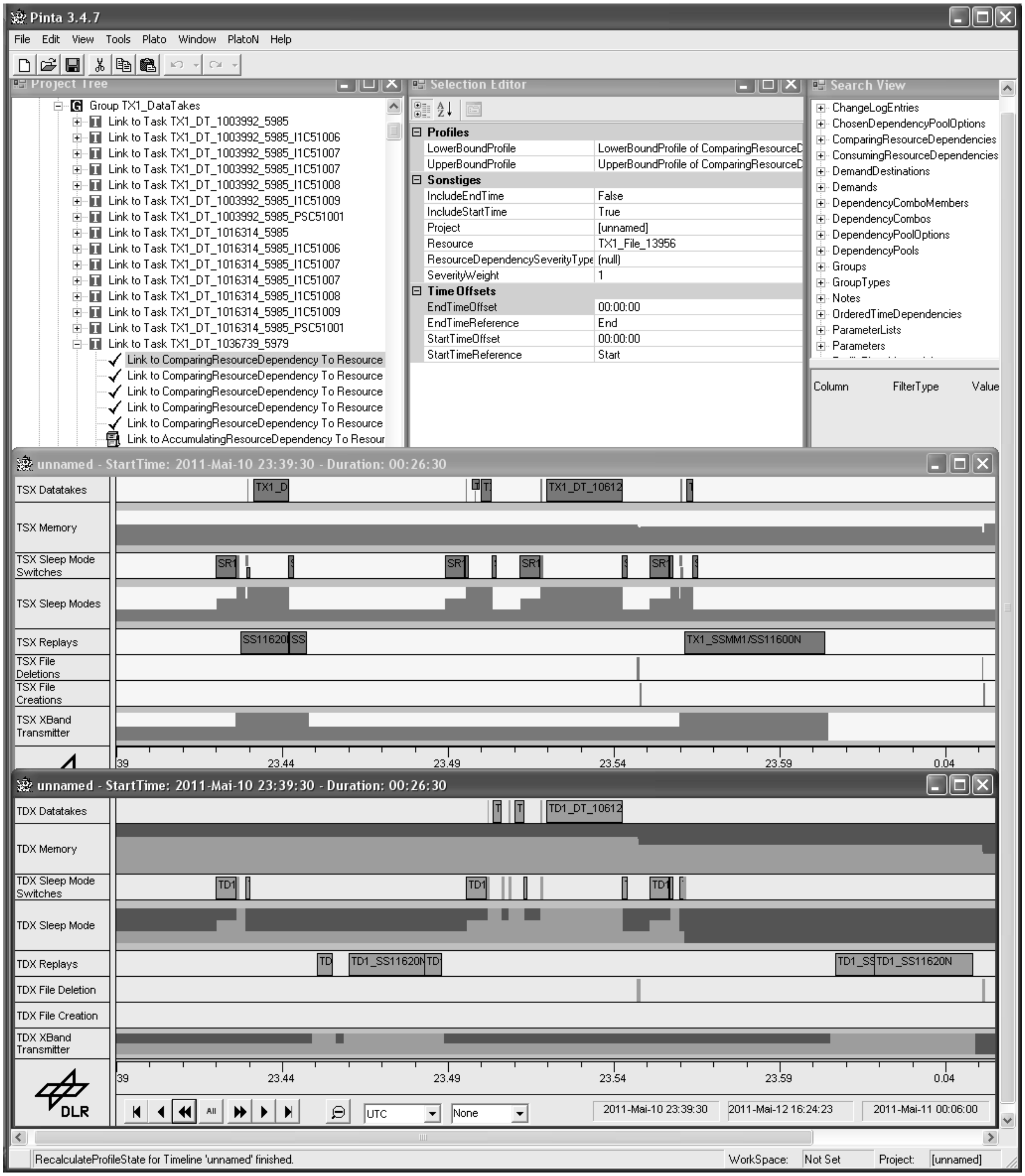
- separated formation
    - parallel downlink of both datatakes is possible
    - during one pass, reception of data from both satellites with only one antenna is impossible
- near formation:
    - no parallel downlink of the two satellites, because their signals would interfere
    - during one pass, reception of data from both satellites with only one antenna is impossible
- close formation:
    - no parallel downlink
    - no parallel execution of distinct datatakes
    - Reception of data from both satellites is possible with only one antenna during one pass, but a handover margin of ten seconds must be considered.
    - proper SyncHorns must be selected on both satellites in order to allow synchronization of the two satellites during bistatic datatakes and during SyncWarnings (see below)
    - exclusion zones: neither satellite may activate the instrument whenever the other satellite might get hit by the radar beam

These scenarios had to be addressed by their individual modeling.

Special attention had to be paid to the exclusion zone constraint in close formation. This is a mission critical constraint as violating it may damage the satellite which gets hit by the radar beam of the opponent satellite. Therefore an additional check is made on board the satellite. However this check relies on both satellites being in their predicted position. To be sure of this, we need to obey another rule:

- datatakes may only be scheduled in case a synchronization check is scheduled no longer than 53 minutes beforehand

This so called ‘SyncWarning’ would deactivate both satellites’ instruments in case it fails.



Group TX1_DataTakes
- Link to Task TX1_DT_1003992_5985
- Link to Task TX1_DT_1003992_5985_I1C51006
- Link to Task TX1_DT_1003992_5985_I1C51007
- Link to Task TX1_DT_1003992_5985_I1C51007
- Link to Task TX1_DT_1003992_5985_I1C51008
- Link to Task TX1_DT_1003992_5985_I1C51009
- Link to Task TX1_DT_1003992_5985_PSC51001
- Link to Task TX1_DT_1016314_5985
- Link to Task TX1_DT_1016314_5985_I1C51006
- Link to Task TX1_DT_1016314_5985_I1C51007
- Link to Task TX1_DT_1016314_5985_I1C51007
- Link to Task TX1_DT_1016314_5985_I1C51008
- Link to Task TX1_DT_1016314_5985_I1C51009
- Link to Task TX1_DT_1016314_5985_PSC51001
- Link to Task TX1_DT_1036739_5979
  - Link to ComparingResourceDependency To Resource
  - Link to ComparingResourceDependency To Resource
  - Link to ComparingResourceDependency To Resource
  - Link to ComparingResourceDependency To Resource
  - Link to ComparingResourceDependency To Resource
  - Link to AccumulatingResourceDependency To Resour

Profiles	
LowerBoundProfile	LowerBoundProfile of ComparingResourceD
UpperBoundProfile	UpperBoundProfile of ComparingResourceD
Sonstiges	
IncludeEndTime	False
IncludeStartTime	True
Project	[unnamed]
Resource	TX1_File_13956
ResourceDependencySeverityType	(null)
SeverityWeight	1
Time Offsets	
EndTimeOffset	00:00:00
EndTimeReference	End
StartTimeOffset	00:00:00
StartTimeReference	Start

- ChangeLogEntries
- ChosenDependencyPoolOptions
- ComparingResourceDependencies
- ConsumingResourceDependencies
- DemandDestinations
- Demands
- DependencyComboMembers
- DependencyCombos
- DependencyPoolOptions
- DependencyPools
- Groups
- GroupTypes
- Notes
- OrderedTimeDependencies
- ParameterLists
- Parameters

Column	FilterType	Value

RecalculateProfileState for Timeline 'unnamed' finished. WorkSpace: Not Set Project: [unnamed]

Picture 1: Screenshot of several scheduled datatakes, downlinks, file creations, file deletions and sleep level modes, together with some resources

Generic techniques for similar problems

Given these new requirements, we could have followed the old approach we used for sleep-level switching, which would have meant to implement one modification handler for each feature, i.e.

- Whenever a downlink is scheduled, check whether enough antennas are available – note that this is formation dependent.
- Whenever a datatake is scheduled, take care that a SyncWarning is scheduled beforehand and if not, adapt the SyncWarning timeline entries.
- Whenever a datatake or a SyncWarning is scheduled, take care that the SyncHorns are in proper constellation.

For sure this direct approach would have worked, but we would not have had any benefit for future missions. We therefore decided to implement a new feature, called 'RepairOnExit'. This feature solves all of the described issues generically, including the sleep-level switching.

The idea is to allow automatic scheduling of a supplier task: We formulate the constraints in a way that the scheduler can detect tasks which can serve to solve potential conflicts on these constraints. Scheduling of the dependent task takes place inside a using-block, in which the respective constraints are replaced by helper constraints, which reflect the potential supply of the supplier tasks. This prevents the need for repairing when adding the supplier task in the next step. When scheduling inside the using statement is complete, the dispose mechanism takes care of removing the helper constraints and reactivating the ignored constraints. To solve the conflicts on these constraints, one or multiple repairer tasks are scheduled as needed. If deconflicting fails, the transaction of this repair-on-exit statement is rejected, i.e. no modification takes place:

```
var config = .. // here we specify what constraint shall be
                // repaired by scheduling a supplier task
using(new RepairOnExit(config)) // here we calculate
                // where a supplier task may be scheduled
{
  ExecuteCode(); // here the specified constraint is
                 // replaced by constraints, which
                 // reflect the places where a supplier
                 // can be scheduled
  Transaction.Current.Accept(); // implicit transaction
                                // must be accepted
} // here we try to schedule repairer timeline entries,
  // in order to deconflict the specified constraint
```

Let's have a more detailed look at this mechanism for the SyncWarnings as an example:

There exist 36 different SyncWarning tasks, since there exist six sync horns on each satellite. For each sync horn pair we need a dedicated SyncWarning task

1. Before scheduling a datatake, a new transaction is started
2. The SyncWarnings around the time when the datatake may be scheduled, are removed
3. The times are calculated when a SyncWarning may be scheduled
4. The constraint 'Datatake needs SyncWarning' is replaced by a constraint reflecting the result of 3.
5. The datatake is scheduled, together with uplink, file creation, downlink, file deletion and sleep-level switchings. If this is not possible, the whole transaction is rejected and we exit this block. Otherwise we proceed as follows:
6. The constraint from 4. is removed and the one of 3. is reactivated
7. If the constraint 'Datatake needs SyncWarning' is violated, one or more SyncWarnings are scheduled to deconflict all conflicting 'Datatake needs SyncWarning' constraints. If this succeeds, the transaction is accepted. If not it is rejected.

Note that in step 7, we only add SyncWarnings in case we do have a conflict. This allows an easy mechanism for activating and deactivating these features due to different constellations:

We introduce four tasks 'farFormation', 'separatedFormation', 'nearFormation' and 'closeFormation'. At any time, exactly one of them is scheduled, namely the one whose formation is active at that time. The generic constraint definition allows adding a constraint for the first three of these tasks: They all supply the SyncWarning supplied resource in the same way as the SyncWarning does. This means that whenever we are not in close formation, the resource is supplied and no SyncWarnings are commanded – completely without reconfiguration. The only input needed is the information of the satellite scenario.

Preferred TSX-1 downlink

The two satellites are very similar. However they differ quite significantly in mass memory: the newer TDX-1 satellite may store twice as much data as the TSX-1 satellite. If we would treat the two satellites equally, we can expect the TSX-1 memory to be full although 50% of the TDX-1 memory is still available. For the TerraSAR-X mission, this is not such a big issue, because on the one hand, the high priority datatakes are already scheduled and on the other hand the TDX-1 satellite may also execute the datatake in most cases. For the TanDEM-X mission however the situation is different: each datatake for this mission must be executed on both satellites synchronously. Whenever one satellite's memory is full, no such pair of bistatic datatakes can be executed any more. The TanDEM-X mission therefore would not benefit at all from the increased capabilities of the TDX-1 satellite.

We therefore had to introduce a concept which somehow first dumps the TSX-1 satellite. We started to introduce a rescheduling of downlinks when TSX-1's limit is reached. However we ended up in complex code and a bad performance. In the end, we changed our concept the following way:

While scheduling the bistatic datatakes for the TanDEM-X mission, we schedule the datatakes not in the order of their priorities (which is equal anyway for all of them), but we schedule them in the order of their opportunities. After one bistatic pair of datatakes has been scheduled, we unschedule the downlink of the TDX-1 satellite and store the downlink in a queue. As soon as the TDX-1 memory is filled by more than N% (e.g. N = 50), the 'oldest' element in this queue is taken and scheduled. After scheduling the last bistatic TanDEM-X datatake, all remaining TDX-1 downlinks are scheduled before the TerraSAR-X datatakes are considered. This results in the following sequence:

- Bistatic datatake pair 1:
    - o datatakes scheduled at 12:00
    - o TSX-1 downlink scheduled at 12:30
    - o TDX-1 downlink postponed
- Bistatic datatake pair 2:
    - o datatakes scheduled at 12:10
    - o TSX-1 downlink scheduled at 12:32
    - o TDX-1 downlink postponed
- …
- Bistatic datatake pair 10:
    - o datatakes scheduled at 13:00
    - o TSX-1 downlink scheduled at 13:30
    - o TDX-1 downlink postponed
- TDX-1 meory filled by more than N%
    - o TDX-1 downlink for datatake pair 1 scheduled at 12:38
- Bistatic datatake pair 11:
    - o datatakes scheduled at 13:10
    - o TSX-1 downlink scheduled at 13:32
    - o TDX-1 downlink postponed
- TDX-1 memory filled by more than N%
    - o TDX-1 downlink for datatake pair 2 scheduled at 12:40
- …

As long as there exist enough unused downlink opportunities for scheduling the TDX-1 downlinks, scheduling them takes place before the opportunities of all bistatic datatakes, which have not yet been considered. Therefore the TDX-1 downlinks can be scheduled without affecting the TSX-1 downlinks. This way the TDX-1 memory is only dumped when TSX-1 is empty.

This concept will work unless there don't exist enough downlink opportunites. In this case the downlinks of TDX-1 would 'overtake' the currently scheduled datatakes, which means that new TSX-1 downlinks and old TDX-1 downlinks compete for the downlink opportunities. But this is correct, because when the TDX-1 memory is filled by more than N%, we expect both satellites to have equal memory left. So why should TSX-1 be preferred any more?

Nevertheless the high-priority bistatic datatakes are requested in a systematic way, which assures that there exists enough downlink capacity for the TanDEM-X mission. We therefore did not observe this situation at all.

Remaining flexibility

The high-priority bistatic datatakes of the TanDEM-X mission are ordered in a systematic way, which assures that these requests may be scheduled. Of course we still need to serve the TerraSAR-X mission, whose orders are unknown until up to six hours before uplink. We therefore need a mechanism, which on the one hand allows high-priority TerraSAR-X datatakes to block TanDEM-X datatakes and on the other hand assures that the TanDEM-X mission can be completed.

This is achieved by the Remaining Flexibility concept: Before starting the algorithm, we analyze the bistatic datatakes of the TanDEM-X mission. Each datatake has a time window when it must be executed. Inside this time window, we have one opportunity all eleven days. Usually a datatake has a time window of about 45 days, i.e. five opportunities. For each of these TanDEM-X datatakes of the current three days scheduling horizon, we check how many opportunities are left which are not yet occupied by other TanDEM-X datatakes. This number specifies the flexibility during the TanDEM-X datatake's opportunity.

At the beginning of the scheduling process, all high-priority TerraSAR-X datatakes, which overlap with a region of flexibility less or equal 1, are blocked and queued for later consideration. When the last TanDEM-X datatake has been scheduled, all of these blocked TerraSAR-X datatakes get a second chance, just in case the TanDEM-X datatake could not be scheduled for any other reason.

Background sequence

Of course we have to schedule additional tasks, such as housekeeping dumps and transponder switchings. These tasks can always be schedulable somewhere. Therefore they are not considered during the main timeline generation step. Nevertheless we still need to consider certain constraints to tasks of the main timeline; therefore we schedule them in a post processing step.

The requirements to schedule these background sequence tasks have been as volatile as the constraints of the TSX-1 satellite. So we implemented this algorithm as generic as possible. Its basic functionality is to take opportunities and add an offset to their start time or their end time. At this place a timeline entry is generated.

In order to maintain a conflict free timeline, the resulting timeline entry is deconflicted by moving into a specified direction. A transmitter-on command for example would be moved to an earlier time, if the number of telecommands in the calculated second slot exceeded its maximum value. A dumpStart command on the other hand would be moved to the succeeding second slot.

You can also specify a minimum distance in between the opportunities, when these should be merged. This way you can avoid switching off and on a transmitter at a short distance.

Of course it does not make sense to schedule a dump command if the transmitter is not switched on. To assure this doesn't happen, you may group commands together and specify different levels.

For example consider two groundstation opportunities which are at a very short distance to each other or even overlapping. The first level rule would merge these opportunities and schedule transmitter on and off before and after the pair of opportunities. The second level rule does only apply if the first level rule has succeeded to add the transmitter switchings. In this case the second level rule would consider these opportunities separately. For the first one the dump start and dump stop commands would be generated. For the second one the dump start would be scheduled not earlier than the preceding dump stop and according to the second opportunity.

Similar to the constraint definitions, this algorithm is completely configurable via a well defined XML file.

Performance

When starting with the TerraSAR-X mission, we were thinking about implementing some optimization criterion, which might be used for optimization. However for this mission, optimization was not desired. Instead a simple rule was preferred.

It turns out that global optimization might not be possible at all for this mission within the available time. Even our simple algorithm currently takes one hour calculation time for a short-term timeline, which covers three days. The workload is about 3000 datatake alternatives, i.e. one datatake for the TSX-1 satellite and one for the TDX-1 satellite. The dominating factor of the large work load is the power resource model.

For the TerraSAR-X mission, we had a simple rule that no more than 180sec datataking may take place in any 95 minutes window. For the TanDEM-X mission we have to reach the limits of the satellites, therefore a detailed power and thermal analysis has been performed. We now have 15 gliding windows; the smallest one is the 95min window, for which the maximum datatake workload is now extended to 400sec. The largest one is 15*95min window with a maximum workload of 15*210sec. However the calculation effort is caused by the power model, which directly models the state of discharge of the battery. This model supports the battery capacity, i.e. whenever the battery is full, further energy supply by the solar arrays get lost. Using this model means that adding a power consumption results in propagating the whole future, at least until the next time where the battery is completely refilled.

But also without the power calculation, optimization would be a hard thing. Tests by us and others have indicated that the problem is too complex to tackle using more generic stochastic approaches.

Prospects

The technique of specifying constraints and algorithms via XML files has proven to be very helpful. For future missions we therefore intend to implement a similar approach for structure generation. This way, a simple mission might only need to make use of the generic tools together with a properly adapted configuration.

Our modeling language is quite straight forward and allows intuitive modeling of constraints. In order to use more generic algorithms, one may write a project analyzer which identifies the bottlenecks of the planning problem and extracts them into a simplified model. A generic algorithm may be applied to this simplified model and the result (e.g. a selection of datatakes, which shall be scheduled) can be used as input for the scheduling algorithm on the complete model. However it seems to be extremely challenging to write a good generic project analyzer. A mission specific project analyzer on the other hand would drastically reduce our ability to quickly react to changing constraints, which is vital especially for a one of a kind mission such as TerraSAR-X/TanDEM-X.

References

[1] Maurer E., Mrowka, F, Braun A., Geyer M. P., Lenzen C., Wasser Y., 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

[2] Braun, A., Geyer, M.P., Wickler, M., Foussal, C.I., "Autonomous End-to-End Planning System for TerraSAR-X," AIAA Space Ops 2004

[3] Geyer, M. P., Braun, A., Foussal, C.I., Lenzen, C. and Köhler, A., "Tailoring the TerraSAR-X Mission Planning System to PPP needs," AIAA Space Ops 2006

[4] Dr. Michael P. Geyer, Dr. Falk Mrowka, Christoph Lenzen, “TerraSAR-X/TanDEM-X Mission Planning – Handling Satellites in Close Formation” AIAA Space Ops 2010

[5] Christoph Lenzen, Dr. Falk Mrowka, Dr. Andreas Spörl, “Scheduling Formations and Constellations”, electronic poster at the AIAA Space Ops 2010

[6] Nir Shavit and Dan Touitou. Software Transactional Memory. Proceedings of the 14th ACM Symposium on Principles of Distributed Computing, pp. 204–213. August 1995.

[7] Tim Harris, Simon Marlow, Simon Peyton Jones, and Maurice Herlihy. Composable Memory Transactions. ACM Symposium on Principles and Practice of Parallel Programming2005 (PPoPP'05). 2005.

Acknowledgments

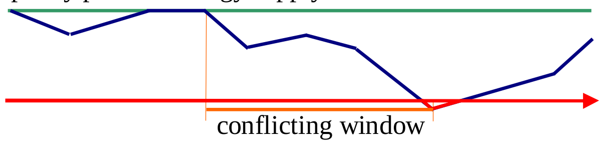
The TanDEM-X project is partly funded by the German Federal Ministry for Economics and Technology (Förderkennzeichen 50 EE 1035) and is realized in a public-private partnership by DLR e.V. and Astrium EADS

Appendix: Simplification of the power model

Performance analysis showed that the power model exposes a bottleneck in terms of computing time. Reason for this is the following: when you add a power consuming timeline entry, you have to recalculate the power resource beginning from the new timeline entry until the succeeding time where the battery capacity is reached. In case of a full timeline this may be the end of the scheduling horizon. In addition to avoiding this calculation effort of $O(n^2)$, you may want to remove the upper limit feature in order to be compliant to an alternative scheduling engine. In the following section, we want to show how you can do this using an approximation by gliding windows:

I. Exact modeling via gliding windows

In case you have a conflict on the power resource, you can find a time interval, which starts with fill-level = upper limit and where the consumption is greater than the battery capacity plus the energy supply:



Therefore we know that you can replace the charge level resource by gliding windows of all sizes where the upper bound of the gliding window of size Δ is

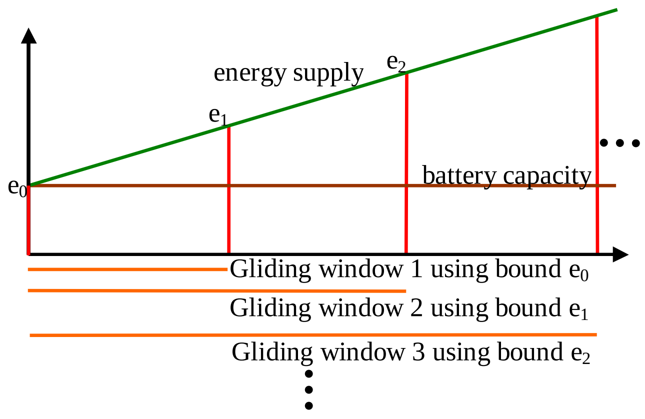
$$\text{bound}(\Delta) = \text{battery capacity} + \text{energy supply during } \Delta$$

II. Restricting to a finite set of gliding windows

In order to assure that a gliding window of size Δ is obeyed, it is sufficient to show that a gliding window of greater size and equal or lower upper bound is obeyed. You therefore can restrict to a series of gliding windows where the limit of the n-th window is given by

$$\text{bound}(\Delta_0) = \text{battery capacity}$$
$$\text{bound}(\Delta_n) = \text{battery capacity} + \text{energy supply}(\Delta_{n-1})$$

e_3

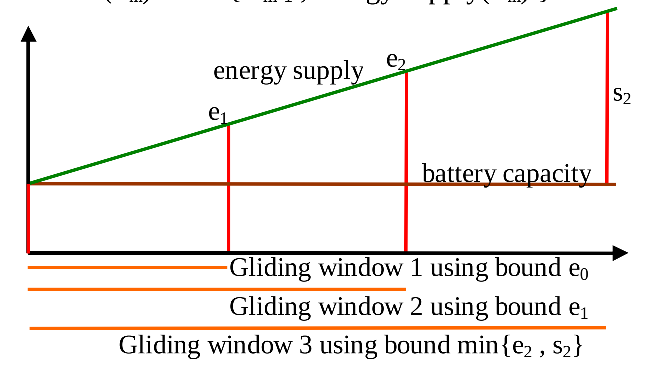

energy supply
e₂
e₁
e₀
battery capacity
Gliding window 1 using bound e₀
Gliding window 2 using bound e₁
Gliding window 3 using bound e₂

But still we don’t achieve a computational benefit, because the biggest window would have size of the planning horizon, thus leading to a computational effort of $O(n^2)$.

Bound of the greatest gliding window

In order to solve this issue, you need to break off the chain of gliding windows at some index m. You can do so if you obey the following rule:

$$\text{bound}(\Delta_m) = \min\{\, e_{m-1} , \text{energy supply}(\Delta_m) \,\}$$


energy supply
e₂
e₁
s₂
battery capacity
Gliding window 1 using bound e₀
Gliding window 2 using bound e₁
Gliding window 3 using bound min{e₂ , s₂}

Proof:

Assume you have a conflict and all your gliding windows are obeyed. By construction of the gliding windows you can find a time window (a , b) of size greater than your greatest gliding window size Δ_m, where you consume more energy than the sum of the battery capacity plus the energy supply during this time window.

You may assume that the upper limit is not reached within the inner part of this time interval, because otherwise you can restrict to the time interval (a_1 , b), where

$$a_1 = \sup\{x \text{ in } (a , b) \text{ and chargeLevel}(x) = \text{upperLimit}\}$$

According to the definition of the greatest gliding window, $\text{chargeLevel}(a + \Delta_m) >= \text{chargeLevel}(a)$. Therefore you know that during the time window $(a + \Delta_m , b)$, the energy consumption is at least as high as during (a , b). You can iterate this until you reach a time window $(a + k^*\Delta_m , b)$ with $b – (a + k^* \Delta_m) <= \Delta_m$. This contradicts the definition of the greatest gliding window, where the energy consumption is restricted to a lower level than used during (a , b).

In the most simple case, you choose one window:

$$\Delta = \text{duration to fill the empty battery}$$
$$\text{bound}(\Delta) = \text{battery capacity}$$