

Spike: A Dynamic Interactive Component In a Human–Computer Long–range Planning System

Leslie Zimmerman Foor¹ and Drew J. Asson²

¹Space Telescope Science Institute

²Points of View Software, Ltd.

¹3700 San Martin Drive, Baltimore, MD 21218

²9500 Nightsong Lane, Columbia, MD 21046

foor@stsci.edu, asson@ptsview.com

Abstract

While some systems attempt to build complete solutions into one package, Spike is a dynamic component of a combined human/computer solution to long–range planning of observations for the Hubble Space Telescope (HST). We will focus on its interaction with users as well as with other software components. We suggest that this enables Spike to be an integral part of a robust planning solution.

Introduction

The problem Spike addresses is trying to find times to schedule astronomical observations that are submitted by individuals awarded time on HST. The software attempts to find places during the planning period to schedule these observations. The goal is efficient use of the HST's limited resources as well as timely execution of each individual's observations.

Spike calculates constraints for HST observations and, using those constraints, plans most of the observations for the current and upcoming observing cycles. By allowing the users to specify their own temporal constraints and adjust criteria that determine how the observations are planned, Spike offers a flexible environment for HST planning. Users have real control over the choices Spike makes and can inject their goals into the process before, during, and after Spike is used.

It is useful at this point to discuss the three types of Spike users. The first are the astronomers themselves, who use a version of Spike that provides coarse level suitability information for their observations. They use Spike as a component of RPS2, the software developed at Space Telescope Science Institute (STScI), which is used by astronomers to craft their observing programs. They use Spike only to find suitability for their individual proposal. The second set of Spike users is the program coordinators (PCs), who are the interface between our community users and STScI. They most often use Spike in a single proposal mode as well. The third group of users is the Long Range Planning Group (LRPG). These are the individuals who craft the long range plan (LRP) for

all the observations for HST during the planning period. This third group is the one we refer to below as the users.

Prior to the beginning of the observing cycle, most of the proposals (containing observations) are submitted by the principal investigators for processing. During this processing, the observations are transformed into a hierarchical representation that the HST planning and scheduling systems can use, and which will eventually be up–linked to the telescope for execution. In this hierarchical representation, an observation is referred to as a visit. For each visit, absolute constraints related to physical constraints are calculated. Examples of such constraints are target visibility and avoiding pointing HST near the sun or moon. User specified timing and orientation requirements may also be applied to a visit. An example of an absolute timing requirement would be to restrict a visit to execute between two calendar dates.

Visits can also be grouped into what we call *link sets*. The members of these link sets can be joined by relative timing or orientation links. An example of a relative timing link would be *schedule visit 2 after visit 1 by 10 days*. A relative orientation link could be to orient the telescope 30–40 degrees away from the telescope's orientation of another visit. This would be represented as *orient visit 2 30 to 40 degrees from visit 1*. A visit with no relative links is still considered to be a link set but with only one member.

The range of times when a visit or a group of visits are suitable to be scheduled is referred to as its *constraint windows*. Visits are assigned *plan windows*, which cover a time period of nominally eight weeks where the visit is suitable for scheduling in any of those weeks. A plan window is a good faith commitment to execute a visit during a particular period of time. Plan windows are subsets of a visit's constraint windows. For a more detailed description of the mechanics of visits, link sets, and plan windows, see Kramer and Giuliano (1997). For more information on HST and its various physical constraints, please see Downes and Rose (2001).

Human–Computer Interaction

At one level, Spike provides feedback to users on the constraint and plan windows for visits. This feedback includes both textual and graphical information. Via the graphical user interface (GUI), Spike shows the overall constraint windows for a visit. One can drill down and see how the component constraints, such as target visibility and sun avoidance, combine to generate the overall suitability. This information allows a user to have a more detailed explanation as to why there is no suitability at a particular time. If plan windows have been assigned to a visit, they are also displayed in the GUI.

In addition to these graphical images, several textual reports are also available. Two examples of this are the target orientation and orbital viewing reports. The first report shows the allowed telescope orientation angle ranges per day in the planning interval. This can explain why a visit might not have suitability on a particular day. This information can also be used to help select a day when there is suitability. The second report is a day-by-day reckoning of how many minutes of target visibility are available per HST orbit. If an observation requests a certain amount of exposure time and Spike says there is no suitability at a certain point, one reason may be that there is not enough visibility available on that particular day. This report can provide that explanation.

Guiding and Directing

While these explanatory features are useful, Spike also offers several ways for users to guide and direct Spike's planning capabilities. The system offers three built-in types of control over the planning process. These items are control files, criteria, and critics.

First, we would like to provide a simplified overview of how Spike operates. The system is started and a control file is loaded. All the visits that will be part of the plan are loaded into the system. To load a visit means to construct the constraints that represent the visit, both internal to itself as well as external links to other visits. This information is cached to speed processing in future runs. If desired, a previous long range plan may be loaded that represents plan window commitments that have already been assigned. Adjustments to the previous plan are made if needed, which include checking if a member of a link set has executed and if so, recalculating the constraints on the remaining members of the link set. Then, the scheduler is run which selects any visits that need plan windows and tries to find suitable ones for them. The algorithm is relatively simple. First, choose a link set to schedule. Then, iterate day by day over the planning interval generating potential plan windows of a specific length starting on those days. Finally, score these candidates and select the best one. Note that we do not assign that window into the plan immediately, but wait until all visits have been tried. The reason is that we prefer to let all visits find the plan windows that are best for it

individually, instead of having them compete against each other. After all visits have been run through this phase, the plan window assignments are committed and a repair phase commences which uses an iterative stochastic repair mechanism to resolve oversubscribed regions of the plan. Finally, the new plan is written out to a database and reports are generated on the assigned windows as well as reasons why some visits might not have received plan windows.

A control file provides Spike with general information on HST's orbits and constraints. Tunable parameters describe where to get visit information and how much of it should be included when calculating the constraint windows for a visit. The control file also allows one to configure the nominal length of a plan window, how many orbits should be schedulable in a day, and how large the plan window of the first member of a link set should be.

These control values need not be the absolute truth, but can be what is desired. For example, the number of orbits schedulable in a day might be set lower than the amount that are actually available. This allows the users to incorporate a contingency pad of time into the plan. Different control files can be created and used for multiple purposes, such as building a long-range plan from scratch, regular plan maintenance, and individual proposal processing. In addition, different control files can be loaded at various times while Spike is running, without the need to restart the system from scratch.

Criteria are the main features that enable users to guide the selection of plan windows for link sets. Criteria are plan window scoring functions. Multiple criteria are applied to a single candidate window, and the values are combined in a multiplicative fashion. Each criterion can return a score in the interval [0,1] and the combination of scores is also in [0,1]. Each criterion has a weight assigned to it, which is taken into account when combining the scores. The formula for calculating the overall score is $(1 - weight * (1 - score))$. From this equation, one can see that a score of 0 and weight of 0 means that the overall combined scores will be 0. We call this feature the single criterion veto feature. This feature is useful for absolute criteria, such as the *not before* and *not after* dates. These dates carve out a subset of the entire planning interval where Spike should try to assign plan windows. Outside of this subset of time, windows will not be accepted.

Spike breaks criteria into two groups, one set for selecting a link set to be assigned a plan window, and one for scoring candidate plan windows for that link set. The first set is called the *candidate selection* criteria. It determines how schedulable a link set is within the entire planning interval. It also looks at how much of the link set has already been executed. This is useful only for link sets that contain more than one visit. One desire is to select link sets for planning if part of the link set has already executed. Another heavily-weighted candidate selection criterion looks at the size of the link set. It tries to select the set with the most visits to schedule first.

Once a link set has been selected, the second set of

criteria is used to look at candidate plan windows. These are called *value selection* criteria. These criteria calculate a score based on a variety of factors. One is favoring plan windows earlier in the planning interval rather than later. Another tries to arrange plan windows for link sets in the same proposal near each other in time. An attempt is also made to group plan windows for visits in a link set close together. Yet another tries to maximize the coverage of a plan window over its related constraint windows. The *not before* and *not after* criteria described above are value selection criteria.

Criteria are useful tools for human involvement since they allow users to turn them on and off as well as assign the weight given to the criterion's score. Some also allow the user to configure the scoring procedure, such as whether the criteria applies to internal visits, that is, visits which do not look at an astronomical target but that are used for configuration, calibration, or testing an instrument. Other configuration options include the dates for the *not before* and *not after* criteria and the allowed over-subscription ratio for a period of time during the planning interval.

While criteria provide a powerful method to control the planning process, they can be overwhelming for users to manage. While there may be a dozen criteria applied to a candidate plan window, only one score is generated. Spike users have created domain knowledge regarding how to tweak the weights and scores of criteria to achieve their desired results.

Critics were added to Spike in 2001. Whereas criteria do not change a candidate plan window, critics can change the candidate window before the criteria are applied to it. Critics were created to ensure relative orientation constraints are maintained across members of an orientation link set, to maximize South Atlantic Anomaly (SAA) hiding, and avoid plan window slivers. The SAA is a physical constraint that defines times during each day when HST cannot conduct observing activities. For more information on the SAA, please see Downes and Rose (2001). Slivers are a result of applying a plan window to a visit whose constraint windows are non-contiguous. A plan window can span several of the suitable times with gaps in the plan window where there is no suitability. The sliver critic attempts to minimize the creation of slivers by eliminating them under certain conditions from the candidate plan window. Like criteria, critics can be turned on and off and can be configured. However, unlike criteria, critic manipulation by users tends to be more analytical rather than artistic.

Dynamic software modification

We suggest that one reason for the ease of interaction between users and the Spike system is the fact that it is written in Common LISP. It is deployed on Sun Microsystems workstations using Franz Allegro Common LISP. LISP is an interpreted language, but it includes a native code compiler, which allows for runtime speeds

comparable to C. The interpreter is always available and can be used to run existing components of Spike, add new code either from the keyboard or from a file, and debug problems. Some of our users are comfortable with LISP and have been able to access and modify the software for their needs without affecting the delivered system or creating multiple copies of Spike for each of their cases. The control, criteria, and critic files described above are actually LISP code that are loaded at various stages of Spike's processing. Some of our users who do not have an understanding of LISP still edit these files to adapt Spike to their needs.

While some users interact intimately with Spike through the LISP interpreter, more often a software developer will write code that can be dynamically loaded. An example of this was a situation where a user wanted to be able to get a textual representation of the suitability for over 300 visits in a single proposal. The GUI only allowed him to get it for one visit at a time. While the system had all the information he needed, he could not get it in a format that he could use. Normally, he would have filed a change request and the next official release would contain the report he needed. However, this report was not something that would be used often nor used by many people, and adding it to the system would increase its complexity. In addition, he needed this information in the near-term.

The approach we chose was to write three LISP functions and give them to him in a file. He loaded that file into his currently running Spike process and ran the one interface function that produced his report. He did not need to restart the system or reload his proposal. Note that loading a complicated, 300+ visit proposal may take a long time. We were able to deliver a reporting capability in approximately 20 minutes without the need for a restart or reload. We also did not introduce code into the system that might not be useful for anyone else. Finally, we did not need to create a new image, which is time consuming and wastes disk space.

In addition to the interpreter, the full suite of LISP's debugging tools is available. We see this as an extremely useful benefit since any delivered product, no matter how much testing it undergoes, may have unfound bugs as well as undocumented features. Debugging tools are essential to resolve these problems. LISP offers extreme flexibility on this matter, since all of the language features are available from the point where the software breaks. An issue can be addressed and the process can continue from where the problem was encountered without having to restart the process. The current state of the system can be evaluated as well, so the system does not have to be modified with extra *print* statements, recompiled, redelivered, and rerun on the same information.

As an example of the power of this debugging feature, we recently had a user who had been running an expansive test of Spike. This included processing many visits and was expected to run for six hours. Spike crashed at the very end of the run, just minutes from completion. The

debugger told us where the crash occurred and we understood exactly what had gone wrong. Instead of having to fix the bug and rerun the entire process, we were able to load the software fix right into the LISP interface and restart from that point. The process finished in a few minutes and we saved six hours.

The choice of using LISP was not forced upon our development group and we have written several other systems in it as well. At several times over the last few years, we have reconsidered our language choice. We find that the features that LISP provides are greater than what other languages can offer us. In addition, an export version of Spike has been used on a handful of other observatories, both space-based and ground-based. We feel that the track record of our language choice over both time and across varied projects reinforces the fact that it is a solid language for our project.

Directed human interaction

In addition to the control, criteria, and critics files, Spike users also adjust the scheduling of visits on the long-range plan by adding their own temporal constraints. These constraints are called *User Specified Constraints* (USCs) and are simply timing windows stored in the database. USCs can be combined with other visit constraints to restrict the overall suitability of a visit. This is especially useful when certain external constraints must be taken into account. A recent example of their usefulness is in the planning of observations for the new instruments that were installed during the recent HST servicing mission. USCs were added to those visits so that they would not be planned until after the instruments were commissioned following their orbital verification.

USCs can also be used to override the suitability that Spike has calculated without changing the visit's definition. This allows the users to experiment with potential solutions to scheduling problems to find the optimal solution. Without this ability, the user would have to change the original proposal specification, reprocess the visit, and have Spike recalculate the constraints for the changed visit, just to see if that one potential solution is viable. This "what if" capability highlights the interactive nature of Spike. These types of USCs are also useful when the plan window (and perhaps constraint window) for a visit does not agree with the time generated by the short-term scheduler, which determines the exact execution time for the observations. This situation is discussed in detail in a later section.

Turning from individual visits to groups of visits, a plan was built recently from scratch for the next HST observing cycle. To do this in Spike, the LRPG devised a layering system to improve the scheduling of highly constrained visits and the scheduling of limited temporal resources, such as *SAA-free* time. Some observations require that they be done without a pause for the SAA (i.e., they require *SAA-free* time); however, there are other observations that are not affected by the SAA and can be

scheduled during its passage. Visits that can schedule during the SAA passage are referred to as *SAA-hiders*; visits that can be partially scheduled during the SAA are referred to as *SAA-friendly*.

For the long-range plan (LRP) for this cycle, the LRPG separated visits into categories by instrument and resource restrictions. These categories were scheduled in a layering process—that is, after one category of observations was scheduled, the plan was saved and then the next category was scheduled and the plan saved. This was done until all of the observations were scheduled. There were two categories that each contained three subcategories. The first category consisted of observations that were *SAA-hiders* or *SAA-friendlies*. This category was broken down into three subcategories—highly constrained visits that have only a small window of opportunity for scheduling during the cycle, moderately constrained visits that have restricted windows of opportunity for scheduling during the cycle, and slightly constrained and unconstrained visits that have ample opportunities for scheduling during the cycle. The second category consisted of all visits not included in the first category. As in the first category, these visits were sub-categorized by the extent of their constraints.

Once the categories were determined, the control, criteria, and critics files were modified to generate the plan according to their wishes. USCs were added to the visits to help restrict their suitabilities. Then the visits were loaded in Spike to calculate the suitabilities and schedule them on the long-range plan. The plan was evaluated and the loading and scheduling process was iterated as necessary until a suitable plan was achieved.

In addition to improving scheduling, this layering process breaks down the process of building the plan into discrete pieces that can be evaluated and controlled individually. This allows the LRPG to fine tune each piece until the optimal plan is generated. Thus the users learn how to optimize the process as a whole, and, perhaps, be able to specify future software changes to Spike to have it make smarter scheduling decisions.

The result of this layering process was a relatively even distribution of observations over the planning interval. The *SAA-hiders* and *SAA-friendlies* were scheduled during the SAA passages, allowing more *SAA-free* time for those observations that require it. The even distribution of the plan is important because the pool of observations that can be executed on HST in a particular week is determined by choosing all visits that have plan windows during that week. If not enough visits have plan windows that intersect that week, there could be empty (wasted) time on HST because the pool has less than one week's worth of observations. On the other hand, if a week is very oversubscribed, those visits that cannot be scheduled for execution must be re-planned for a future time, which makes the long-range plan unstable.

Computer–Computer Interaction

For execution of visits on HST, the observations are scheduled on short–term schedules, usually one week in duration. These schedules are generated by a separate scheduling system called the Science Planning and Scheduling System (SPSS). SPSS was the original planning and scheduling system for HST and is the definitive scheduler for observations on the HST. The short term schedules use much finer accuracy in the scheduling and need to schedule the observations with regard to instrument and spacecraft commanding, such as high voltage turn–on/off and availability of TDRS satellites for downloading data. Rather than duplicate the tremendous accuracy that SPSS must use to generate the short–term schedules, Spike can use information from SPSS as additional constraints. Spike is used to create plan windows which are used by SPSS to find pools of candidates for the week.

An example of this is seen in the day–to–day maintenance of the long–range plan. Every night the LRPG runs Spike (through an automatic script) to incorporate changes that may have been made to the observations and update the plan accordingly. One of the most common changes that occur for observations is that they have been executed successfully. Spike knows which visits have executed successfully every day by querying the SPSS database. Once Spike knows that a visit has executed, it may need to adjust the plan windows of visits that may be linked to the executed visits.

Another of the LRPG's nightly scripts runs a tool called "schedulability" that compares the Spike–generated plan windows and constraint windows against SPSS–derived windows. If the plan windows match within a certain percentage, they are considered *verified*. Otherwise, if the plan window comparison does not fall within that percentage, the SPSS window is then used to create a *Verified Constraint Window* (VCW). Spike can use the VCW to override other static constraints and make the suitability in Spike match that in SPSS for an observation. These VCW's are a special case of User Specified Constraints that are generated automatically through the LRPG tools.

Conclusions

The Spike system rejects the typical black box software system by providing an interactive component of an overall long–range planning solution. Users are involved at every step of the process. They provide the initial input, guide and direct the solutions found, and iterate over those solutions until a desired result is produced. Explanation facilities for the user and guidance interfaces for Spike help to create a truly human–centered system.

While a totally software–based system with little human interaction or guidance could possibly be produced, the prohibitive cost in terms of development time and

money make that approach less palatable. Moreover, the addition of new or revised requirements for the system would again make this monolithic piece of software brittle. Commercial software (e.g. COTS) rarely provides the entire suite of functionality needed to solve a problem. There are often specialized pieces of software and human expertise that are available to solve a problem, but they usually are not effectively merged. Spike is an attempt at providing this fusion.

References

Kramer, L., Giuliano, M. 1997. Reasoning About And Scheduling Linked HST Observations With Spike. In *Proceedings Of the NASA Workshop On Planning And Scheduling For Space*. Oxnard, CA.

Downes, R., Rose, S. 2001. Hubble Space Telescope Primer For Cycle 11, Space Telescope Science Institute. http://www.stsci.edu/proposer/cy11/documents/online/primer_cover_.html.