Maintenance Request Generation, Planning, and Scheduling for Highly Reusable Space Transportation

Steve Chien, Dennis DeCoste, Alex Fukunaga,
Gregg Rabideau, Tobias Mann, David Winkler
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109-8099
firstname.lastname@jpl.nasa.gov

Abstract

This paper describes a preliminary concept demonstration of an automated capability to analyze telemetry data to generate maintenance requests and use this information to generate a maintenance plan. In this capability, real-time downlinked telemetry data from a reusable spacecraft would be analyzed to determine appropriate maintenance goals. These goals would then be used to develop a maintenance plan to be implemented upon arrival of the spacecraft. This capability would allow pre-positioning of resources and would lead to shorter turnaround times for the vehicle, increasing the fleet utilization rate and thusly leading to lower costs.

The Highly Reusable Space Transportation (HRST) program targets development of technologies leading to highly reusable space transportation systems which will provide extremely low cost access to space [HRST95, HRST96]. As part of this program, we have been developing and demonstrating advanced scheduling systems for the rapid generation and revision of plans for maintenance and refurbishment of highly reusable launch vehicles. In this application, real-time telemetry downlinked either during flight or immediately after flight would be analyzed to automatically generate a set of maintenance requests. These maintenance requests would then be transformed into a refurbishment plan by an automated planning and scheduling system which would account for available equipment and resources as well as the intricacies of the refurbishment procedures of the highly complex propulsion systems (this operations flow is shown below in Figure 1).

The end target for automating this process is to allow a turnaround of several days for the HRST spacecraft to support a flight frequency on the order of one flight per 1-2 weeks. As a comparison, the space shuttle refurbishment process currently takes approximately 65 days with a flight frequency of once per 4 months. It is worth noting that while automated planning and scheduling is already used extensively in ground processing for the Space Shuttle [Zweben 1994, Deale 1994], one difference is that highly reusable space transportation is being designed with the central goals of: reducing the turnaround time for the unit, simplifying determination of required maintenance activities, and simplifying the required maintenance activities themselves. If the maintenance schedule can be generated using in-flight telemetry then the refurbishment process can be speeded even further by allowing for downlinking of requests for pre-positioning of equipment and resources to minimize schedule delay.

Once the actual maintenance plan has been generated, the planning tool continues to be of use in two ways. First, in many cases there can be several mutually exclusive maintenance activities which can be performed next. Via lookahead and critical path analysis, automated scheduling software can determine the next activities to enable the minimal makespan (overall schedule execution time). Second, as unexpected events arise (such as equipment failures, resource unavailabilities, and schedule slippage), the automated scheduling software has the ability to revise the schedule so as to minimize schedule disruption (movement of activities and resources from their original assignments) and schedule slippage (delay of the completion of the overall refurbishment).

In order to test and validate this technology

[0]This paper describes research conducted by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.
Appears in the Working Notes of the 1997 Workshop on Planning and Scheduling for Space Exploration and Science.

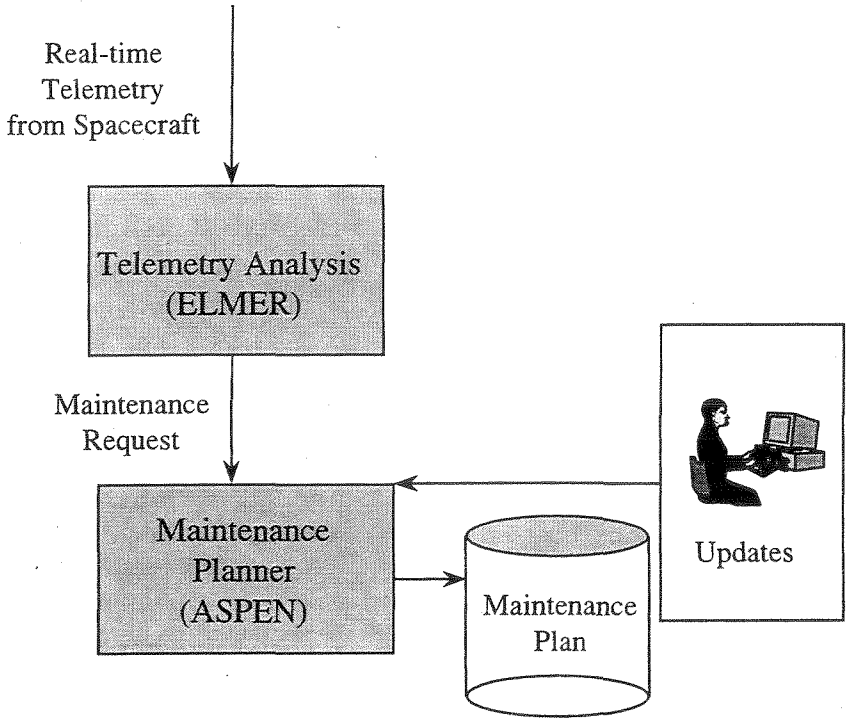

Figure 1: Information Flow for Automated Maintenance Request Generation and Planning

we built a preliminary proof of concept prototype which was demonstrated in the fall of 1996. This prototype consisted of two main modules (as previously described in Figure 1):

1. The telemetry analysis module, which analyzes real-time telemetry and produces a maintenance request; and

2. the maintenance plan generation module, which accepts the maintenance request and generates an appropriate maintenance plan.

For generating maintenance requests, we used the *Envelope Learning and Monitoring using Error Relaxation* (ELMER) system [DeCoste 1997a, DeCoste 1997b]. ELMER uses statistical machine learning techniques to learn and refine input-conditional limit functions from historic and/or simulated data. These limit functions define context-sensitive upper and lower boundaries, within which future data is expected to fall.

To test our prototype, ELMER was feed approximately 100 minutes of data for 12 sensors of the attitude control subsystem from a Space Shuttle mission. Unfortunately, the data we had did not contain abnormalities that were specifically relevant to the HRST domain. Thus, to functionally demonstrate our prototype, we defined (artificial) mappings from the abnormalities detected by ELMER in the Shuttle data to maintenance requests relevant to our HRST planner models.

In the development of the maintenance plan generation module for the prototype, we used the ASPEN planning and scheduling system [Fukunaga et al. 1997]. The ASPEN planning and scheduling system is a flexible, domain independent planning and scheduling application framework designed to support a wide range of planning and scheduling applications. For the HRST prototype, we acquired data on maintenance procedures for the LO2 and LH2 propulsion systems for the proposed Rockwell International led prototype for the X-33 Reusable Launch Vehicle program. While these procedures are not likely to be the same procedures used for HRST (since HRST is one generation beyond RLV) they are likely to have the same sorts of interactions, resources, and characteristics as other space transportation systems maintenance scheduling data.

Maintenance Request Generation Module

In our HRST prototype, ELMER is used essentially to classify each datum in the incoming engineering time-series data stream (i.e. for each sample time and each sensor) as either abnormal (i.e. outside the learned bounds) or normal. Each class of abnormality (e.g. a particular sensor being abnormally high) is then automatically mapped to a corresponding maintenance request, based on a simple manually-specified lookup table.

Traditional approaches such as expert systems could also provide such mappings. However, the key advantage of our approach is that ELMER allows operational definitions of our predeclared classes of symptoms to be automatically learned, based on historic data, to reflect context-sensitivity. Learning symptom detectors instead of manually specifying them can potentially be both cheaper and more accurate.

Given the large volumes of testbed and inflight data that are becoming increasingly common in many spacecraft domains, there is much promise that such learning can be feasible. The primary issue is a fundamental need to appropriately balance the cost of analyzing false alarms against the risks of missing significant faults.

Below we summarize some of the key ideas behind the ELMER techniques, with some emphasis on how it tries to avoid false alarms while still retaining sufficient accuracy.

Motivations Behind ELMER

In practice, automated monitoring of spacecraft relies heavily on limit-sensing and simulation. *Limit-sensing* compares time-evolving sensor values against high and low alarm limits. For the sake of minimizing the run time cost of monitoring, the numbers of false "nuisance alarms", and the costs of (manual) limit-determination, these limits are typically predefined, static, and relatively wide (e.g. "red-lines"). Alternatively, *simulation* dynamically computes tight limits — at the cost of requiring dynamic models that are expensive to create, test, maintain, and execute. Unfortunately, limit-sensing tends to be too imprecise (missing alarms) and simulation tends to be *too precise* (false alarms).

To help overcome those disadvantages, we developed ELMER. ELMER incrementally generates successively tighter high/low limit functions (*envelopes*), essentially moving from the wide static limits typical of limit-sensing toward the precise predictions of simulations, while avoiding unacceptable false alarm rates.

ELMER is motivated by our intuitions that simple limit-sensing *can* perform well, *if* we find limits that are *context-sensitive functions* and we limit-sense across *multiple perspectives* of the raw data. The intuition behind using multiple perspectives (e.g. learning bounds on both a raw sensor and on its derivative) is that that can allow each bound can be relatively loose (to reduce false alarms) yet collectively they might still detect most underlying faults. Supporting that intuition is that fact that many faults manifest themselves as many anomalous data across many times. Many tasks do not necessarily require detecting a fault as earlier as possible, especially if doing so risks costly false alarms.

In our early exposure to Space Shuttle operations, for example, we discovered that operators often slightly nudged the red-lines for some sensors up or down between flights, to account for both nuisance false alarms and for empirical realizations that some limits were excessively wide. ELMER can be viewed in part as a means for automating such a process via machine learning techniques.

Summary of ELMER Techniques

We define the *bounds estimation problem* as follows:

Definition 1 (Bounds estimation) *Given a set of patterns $\mathcal{P}$, each specifying values for inputs $x_1, ..., x_d$ and target y generated from the true underlying function $y = f(x_1, ..., x_D) + \varepsilon$, learn high and low approximate bounds $y_L = f_L(x_1, ..., x_l)$ and $y_H = f_H(x_1, ..., x_h)$, such that $y_L \leq y \leq y_H$ generally holds for each pattern, according to given cost functions.*

We allow any $1 \leq l \leq d$, $1 \leq h \leq d$, $d \geq 1$, $D \geq 0$, making explicit both our expectation that some critical inputs of the generator may be completely missing from our patterns and our expectation that some pattern inputs may be irrelevant or useful in determining only one of the bounds.

For each envelope sensor S, we compare its actual sensed value (y) at each time t+1 to its envelope predictions (y_H and y_L). To simplify discussion, we will usually discuss learning only high bounds y_H; the low bounds case is essentially symmetric. An *alarm* occurs when output y_H is below the target y, and a *non-alarm* occurs when $y_H \geq y$. We will call these alarm and non-alarm patterns, denoted by sets $\mathcal{P}_a$ and $\mathcal{P}_n$ respectively, where $\mathcal{N} = |\mathcal{P}| = |\mathcal{P}_a| + |\mathcal{P}_n|$.

ELMER relaxes the standard least squared regression cost function to include parameters which allow balancing the cost of (false) alarms against the cost of imprecision, as defined in Figure 2.

We can favor non-alarms (i.e. looseness) over alarms (incorrectness) by using values for the penalty (P) factors where $P_{H_a} > P_{H_n}$. This is analogous to the more common use of nonstandard loss functions to perform risk minimization in classification tasks.

$P_{H_a} = P_{H_n} = 1$ and $R = R_{H_a} = R_{H_n}$ gives us the standard class of Minkowski-R errors, where R=2 gives standard sum-of-squares error and R=1 gives classic *robust estimation* (which reduces the effects of outliers).

We focus here on linear regression (i.e. $y_H = \sum_{i=1}^{h} w_i x_i$) to perform bounds estimation, both for simplicity and because our larger work stresses heuristics for identifying promising explicit nonlinear input features, such as product terms. Nevertheless, this approach generalizes to nonlinear regression as well.

Since there is no general closed-form solution using such asymmetric cost, we use an iterative Newton method to perform batch linear regression. Conveniently, performance of the Newton method becomes analogous to that of the classic closed-form solution of singular-value decomposition (SVD) in the special case of symmetric cost (i.e. P_{H_n}=1, P_{H_a}=1, R_{H_n}=2, R_{H_a}=2 and P_{L_n}=1, P_{L_a}=1, R_{L_n}=2, R_{L_a}=2.). Even for asymmetric R and P values, we have found the weights resulting from SVD to be useful as initial seed weights. For example, such seeding allows P_{H_n}=0 to give a non-trivial result for lower-bounding. Otherwise, with initial weights of all 0, P_{H_n}=0 would result in zero error (since $E_{|L|}$ would be 0) and thus training would immediately stop with the constant result of y_L=0.

The complexity of linear regression is cubic in the number of inputs. For our application domains the number of sensors and transforms we wish to consider as inputs is typically in the hundreds or thousands. Thus, in practice ELMER must typically use heuristics to focus on promising candidate inputs. Indeed, given that the main goal in ELMER is not to necessarily fit the data precisely, suboptimal but cheaper and reasonable input subsets can often suffice.

We have found some simple greedy heuristics similar to classic forward subset selection techniques to be useful in practice. Specifically, we compute scores for each candidate sensor based on their correlations with the error residual of the current regression network. We add the candidate with highest correlation to the active input set and then rerun the batch Newton method to learn new (tighter) envelopes. We stop adding new inputs when the error on the validation data set starts to increase, since that suggests that overfitting is beginning to occur.

More fundamentally, our approach requires search over the space of possible and R and P values. Currently we have no optimal method of performing this search. Instead, we spot check combinations of various commonly useful values and pick the best result. For example, these choices typically include $R_{H_n} \in \{1, 2\}$, $R_{H_a} \in \{2, 10, 20\}$, $P_{H_n} \in \{1, 0.1, 0.01, 1e{-}3, 1e{-}4, 1e{-}5, 1e{-}10, 1e{-}15, 0\}$, and $P_{H_a} \in \{1, 1000\}$. The selection criteria involves not only the relative alarm versus non-alarm errors, but also other user preferences, such as acceptable levels of false alarm rates for each sensor. Once again, this does not result in optimal envelopes, but it does provide a cheap way to find reasonable ones. Furthermore, we have also been developing promising alternative techniques which avoid this search over R and P space altogether.

The Maintenance Scheduler

The maintenance planner portion of our prototype used the ASPEN planning and scheduling system. We now briefly describe the ASPEN planning and scheduling framework, the ASPEN modeling language, and the specific RLV model encoded.

The ASPEN System

ASPEN (Automated Scheduling and Planning ENvironment) is a reusable, configurable, generic planning/scheduling application framework designed to enable rapid development of automated scheduling systems for NASA applications,. An application framework [Pree 1995] is a class library (i.e., a reusable set of software components) that provides the functionality of the components found in prototypical instances of a particular application domain. Frameworks anticipate much of an application's design, which is reused in all applications based on the framework. In order to facilitate code reuse, ASPEN (as with many frameworks) incorporates multiple "design patterns". This implies a significant reduction in the amount of code necessary to implement successive systems.

The reusable components provided by ASPEN include:

- An expressive constraint modeling language to allow the user to naturally define the application domain;
- A constraint management system for representing and maintaining operability and resource constraints, as well as activity requirements;

$$e_H = \begin{cases} P_{H_n}(y_H - y)^{R_{H_n}} \text{ if } y_H \geq y \\ P_{H_a}(y_H - y)^{R_{H_a}} \text{ if } y_H < y \end{cases}, \qquad e_L = \begin{cases} P_{L_n}(y_L - y)^{R_{L_n}} \text{ if } y_L \leq y \\ P_{L_a}(y_L - y)^{R_{L_a}} \text{ if } y_L > y \end{cases}$$

$$E_H = \frac{1}{N}\sum_{p\in\mathcal{P}} e_H,\ E_{|H|} = \frac{1}{N}\sum_{p\in\mathcal{P}} |e_H|, \quad E_L = \frac{1}{N}\sum_{p\in\mathcal{P}} e_L,\ E_{|L|} = \frac{1}{N}\sum_{p\in\mathcal{P}} |e_L|$$

Figure 2: Asymmetric cost functions for high and low bounds.

Parameters: $P_{H_a}, P_{H_n}, P_{L_a}, P_{L_n} \geq 0$; $R_{H_a}, R_{H_n}, R_{L_a}, R_{L_n} \geq 1$.

- A temporal reasoning system for expressing and maintaining temporal constraints;
- A set of search engines for constructing, repairing, and optimizing plans/schedules;
- Linear Programming utilities for optimizing schedule preferences; and
- A graphical interface for visualizing plans and schedules (for use in mixed-initiative systems in which the problem solving process is interactive).

ASPEN is currently being utilized in the development of an automated planner/scheduler for commanding the New Millennium EO-1 satellite and a naval communications satellite, as well as for prototype schedulers for the ground maintenance for the Reusable Launch Vehicle and a design analysis tool for the Pluto Express spacecraft.

Models in ASPEN

In order to use the ASPEN scheduling system, a modeling language is used to specify domain-specific constraints and activities. Figure 3 shows part of a domain model specified in the ASPEN modeling language.

Within the ASPEN modeling language, the principal elements are: activities, resources, and states. Activities are the basic items being scheduled. In the RLV application, activities are maintenance steps required to refurbish the hardware, or setup steps necessary to support such operations. For example, each of the subsystems has refurbishment activities for various subsystems. Resources are finite elements whose capacity is limited and hence their use must be coordinated. Within the RLV domain, resources might consist of particular pools of workers with specific maintenance skills, physical locations which allow access to a piece of hardware (for example only two people may fit within a bay which allows access to a particular subsystem), or power limitations (not all maintenance equipment could be powered up at once). States correspond to physical or operational states of a system relating

```
Activity prevalve_removal {
  duration = [15, 20]
  slot subsystem ss
  after prevalve_prep with (ss = subsystem)
  before prevalve_replace with (ss = subsystem)
  reservation hydraulic_lift use 1
  reservation prevalve must_be purged
  reservation prevalve_area must_be illuminated
}

Resource hydraulic_lift {
  type non-depletable
  capacity 1
}

State_Variable prevalve {
  states purged unpurged
  transitions purged to_and_back unpurged
}
```

Figure 3: Sample of ASPEN modeling language (part of the Reusable Launch Vehicle maintenance model). This describes an activity for removing the prevalve of an engine subsystem.

to feasibility of activities. For example, in the RLV application, a state variable could model whether a hatch is open, whether a piece of equipment is powered on, or whether a valve is open.

To ground these concepts, the figure above shows a maintenance activity from the RLV model representing a prevalve removal step. This step must be preceded by a prevalve preparation step and followed by a prevalve removal step. This step also requires a hydraulic lift (a resource requirement) as well as two states (that the area be illuminated and that the prevalve be purged).

The exact maintenance request is derived from the engineering analysis performed by ELMER. Thus, depending on the performance data of the RLV, different maintenance requests would be generated, and hence different planning and scheduling problems posed to ASPEN. ASPEN would receive a maintenance request from the ELMER module, and then construct a plan/schedule using the available

resources.

The RLV Maintenance Scheduling Domain

In the prototype demonstration of the maintenance planning element, we utilized test maintenance procedures developed by Rockwell International during the Phase 1 competition of the Reusable Launch Vehicle Program which ended in July 1996. Specifically, we used the maintenance procedures developed for the LO2 and LH2 propulsion systems for the X-33 Reusable Launch Vehicle. These procedures were then modeled in the ASPEN planning and scheduling system [Fukunaga et al, 1997]. The procedures derived for maintaining and refurbishing the test articles provided a rich testbed for Space Propulsion System Maintenance Scheduling. In all, the testbed involved refurbishment of 2 major systems with 16 subsystems, with a total of approximately 700 lowest level activities in a complete (all subsystems) refurbishment plan. The model that we developed for schedule generation involved: 576 activity types, 6 resources, and on average 6 state, resource, and precedence constraints per activity. In this application we allowed maintenance requests to request either refurbishment of specific subsystems or major systems. In order to schedule the maintenance requests the ASPEN system used a forward sweeping greedy dispatch algorithm which used knowledge of the required precedences of activities in the plan. The resultant scheduler was able to generate schedules for smaller refurbishment problems (8 subsystems, 358 activities) in approximately 8 minutes.

Discussion and Conclusions

The current prototype does not substantially validate the viability of automated maintenance request generation and maintenance planning. Rather it merely serves to illustrate the possibility of such a concept. In order to validate the concept, a more comprehensive demonstration would need to be performed with several key differences. First, the request generation should use actual relevant engineering data, i.e. the same data which currently would be manually analyzed to determine maintenace requirements. Second, realistic mappings from engineering data to maintenance requests should be used. Finally, a more detailed and realistic model of maintenance activites should be implemented. A demonstration in this context would provide significant evidence of the promise of such an approach.

Other work in the area of maintenance scheduling includes work on space shuttle ground processing [Deale et al. 1994, Zweben et al 1994] and testbed work on RLV maintenance scheduling [Allen 1996].

This paper has described a proof of concept demonstration of automated analysis of engineering data to generate maintenance requests for highly reusable space transportation and automated planning and scheduling of such requests using an automated planner. This demonstration highlights the possibility of such an approach to reduce turnaround times for space transportation systems - thus allowing a higher utilization rate, reduced fleet size, and hence reduced operations costs. While the initial prototype was successful, significant further work is needed to validate the viability of the concept.

Acknowledgements

The authors gratefully acknowledge the assistance of John Allen of NASA Ames for his assistance in obtaining HRST testbed data and information.

References

[Allen 1996] J. Allen, Personal Communication, NASA Ames Research Center, 1996.

[Deale et al, 1994] M. Deale, M. Yvanovich, D. Schnitzius, D. Kautz, M. Carpenter, M. Zweben, G. Davis, and B. Daun, *Space Shuttle Ground Processing Scheduling System,* in M. Zweben and M. Fox (eds.), Intelligent Scheduling, Morgan Kaufman, 1994.

[DeCoste, 1997a] D. DeCoste, Automated learning and monitoring of limit functions. Proceedings of the Fourth International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-97), Japan, July 1997.

[DeCoste, 1997b] D. DeCoste, Mining multivariate time-series sensor data to discover behavior envelopes. Proceedings of the Third Conference on Knowledge Discovery and Data Mining (KDD-97), Newport Beach, CA, August 1997.

[Fukunaga et al, 1997] A. Fukunaga, G. Rabideau, S. Chien, and D. Yan, ASPEN: A Framework for Automated Planning and Scheduling of Spacecraft Control and Operations, Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space (I-SAIRAS97), Tokyo, Japan, 1997.

[HRST95] Proceedings of the 1995 Technical Interchange Meeting on Highly Reusable Space Transportation, Huntsville, AL, July 1995.

[HRST96] Proceedings of the 1996 Technical Interchange Meeting on Highly Reusable Space Transportation, Huntsville, AL, August 1996.

[Pree 1995] W. Pree, *Design Patterns for object oriented software development.* ACM Press, 1995.

[Zweben et al. 1994] M. Zweben, B. Daun, E. Davis, and M. Deale, *Scheduling and rescheduling with iterative repair,* in M. Zweben and M. Fox (eds.), Intelligent Scheduling, Morgan Kaufman, 1994.