A COMPARISON OF DECLARATIVE AND HYBRID DECLARATIVE-PROCEDURAL MODELS FOR ROVER OPERATIONS

Russell Knight [(1)], Gregg Rabideau [(1)], Matthew Lenda [(1)], and Pierre Maldague [(1)]

[(1)] Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109
<first name>.<last name>@jpl.nasa.gov

ABSTRACT

The MAPGEN [2] (Mixed-initiative Activity Plan GENerator) planning system is a great example of a hybrid procedural/declarative system where the advantages of each are leveraged to produce an effective planner/scheduler for Mars Exploration Rover tactical planning. We explore the adaptation of the same domain to an entirely declarative planning system (ASPEN [4] Activity Scheduling and Planning ENvironment), and demonstrate that, with some translation, much of the procedural knowledge encoding is amenable to a declarative knowledge encoding.

1. INTRODUCTION

Declarative domain representations facilitate describing what a *proper* (or *good*, when optimizing) solution looks like without having to describe all solutions. This is very attractive for operators of extraterrestrial robotic explorers (spacecraft and rovers) in that the generation of a plan can be separated from the process of certifying that the plan is safe to operate the robotic explorer. In some cases, automated plan generation can take the declarative description and aid in generating the plan as well. This enables faster integration of new constraints and shorter and more effective verification and validation of such systems.

Procedural domain representations are, at the core, executable code that is used to either certify a plan or to generate a plan. Especially with respect to plan generation, constraints that dictate why a plan is constructed a certain way need never be explicitly represented in the domain representation. Procedural representations are usually very fast in terms of execution, but can be difficult to modify when new constraints are introduced and often require more involved and costly verification and validation.

Many planning systems that deal with real problems are hybrid systems, where both declarative and procedural constructs are used in defining a domain. The MAPGEN planning system is a great example of a hybrid procedural/declarative system where the advantages of each are leveraged to produce an effective planner/scheduler for Mars Exploration Rover tactical planning. The work we report here is an exploration of using declarative domain representations for MER tactical planning. The specific planner we adapt to is ASPEN, but the general concept holds for many existing automated planner/schedulers.

Our approach was to develop translators that translate from the core languages used for adapting MAPGEN to ASPEN modelling language. MAPGEN consists of Europa and APGEN [3]. Europa is a constraint-based planner/scheduler where domains are encoded using a declarative model. APGEN is also constraint-based, i.e., it tracks constraints on resources and states and other variables. For APGEN, domains are encoded in both constraints and code snippets that execute according to a forward sweep through the plan. Europa and APGEN communicate to each other using proxy activities in APGEN that represent constraints and/or tokens in Europa.

The composition of a translator from Europa to ASPEN was fairly straightforward, as both are declarative planning systems, and the specific uses of Europa for the MER domain matched ASPEN's native encoding fairly closely. Therefore, we will not delve into translating the Europa modelling language into ASPEN modelling language.

On the other hand, translating from APGEN to ASPEN was considerably more involved. On the surface, the types of activities and resources one encodes in APGEN appear to match one-to-one to the activities, state variables, and resources in ASPEN. But, when looking into the definitions of how resources are profiled and activities are expanded, one sees code snippets that access various information available during planning for the moment in time being planned to decide at the time what the appropriate profile or expansion is. We see that APGEN is actually a forward (in time) sweeping discrete event simulator, where the model is composed of code snippets that are artfully interleaved by the engine to produce a plan/schedule. To address the issue of embedded procedural models, we simulate procedural code as a declarative series of task expansions. Predominantly, we had three types of procedural model to translate: loops, if-statements, and code blocks. Loops and if-statements were handled using controlled task expansion, and code blocks were handled using constraint networks that maintained the generation of results based on what the order of execution would be for a procedural representation.

One great advantage with respect to performance for MAPGEN is the use of APGEN's GUI. This GUI is written in C++ and Motif, and performs very well for large plans. ASPEN's GUI is written in Java, and starts to slow down when working with large plans.

We have demonstrated the system on five shadow operations days where we take the input to MAPGEN and feed it into our system, plan the day, and then compare the days.

2. ARCHITECTURE COMPARISON

MAPGEN takes as input an initial plan. The initial plan is arrived at after the morning planning meeting. The output from the morning planning meeting is the Maestro plan. Maestro is a tool used to view images and plan daily activities. The output of Maestro is read into the skeleton plan generator, which puts a generic set of supporting cast and other details into a skeleton plan. This plan is subsequently edited using the constraint editor to introduce temporal constraints not included in the skeleton plan or Maestro outputs [1]. Fig. 1 shows the flow graphically.

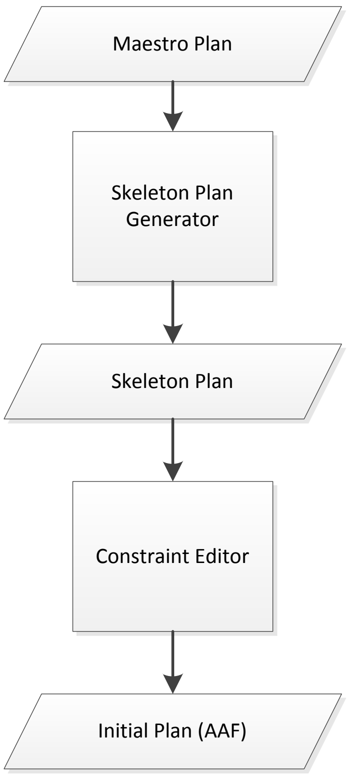


Figure 1. Arriving at an initial plan

The initial plan is in APGEN modelling language (AAF: APGEN adaptation file). We use this as an input to both MAPGEN and ASPEN. The other common inputs are the final state from the previous day's planning session (and subsequent telemetry transmission) and the configuration files for the underlying MMPAT modelling system.

Fig. 2 illustrates the MAPGEN product flow. The Europa Model, APGEN Model, and MMPAT configuration files rarely change. The initial plan and previous day's final state change daily. Here we also see that we may need to invoke the constraint editor during the planning process, especially if we add new activities or change our minds about the ordering of existing activities.

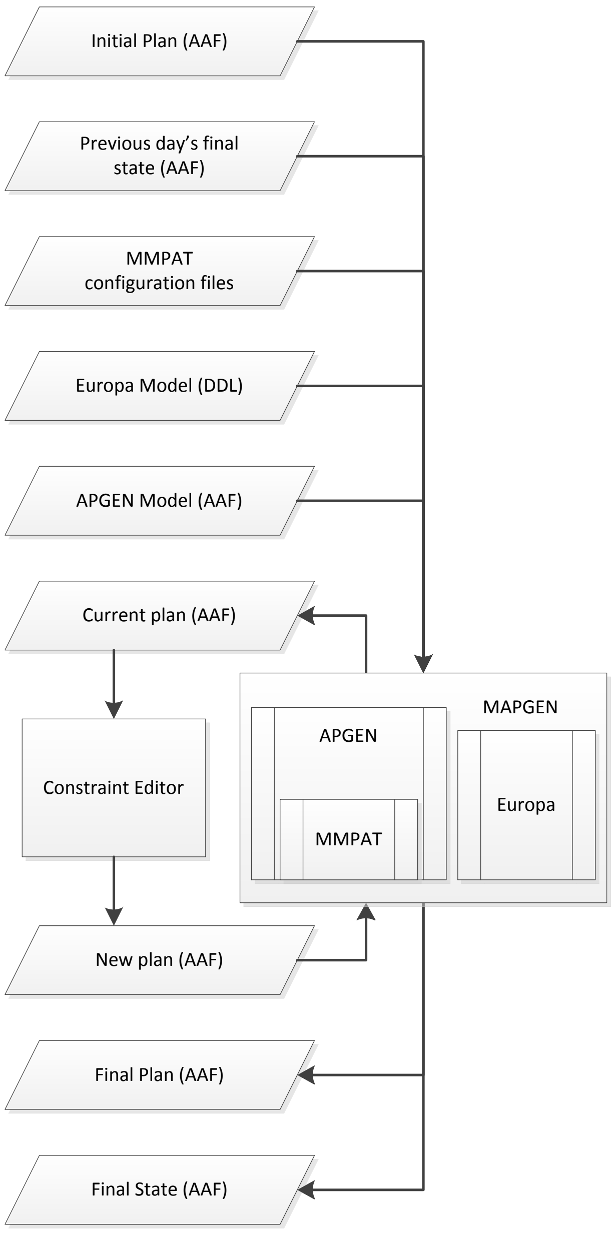


Figure 2. MAPGEN plan generation

Fig. 3 illustrates the plan generation flow while using ASPEN. We see that we use our translated set of models are available and used by ASPEN and that the modification of temporal constraints occurs within the ASPEN tool itself.

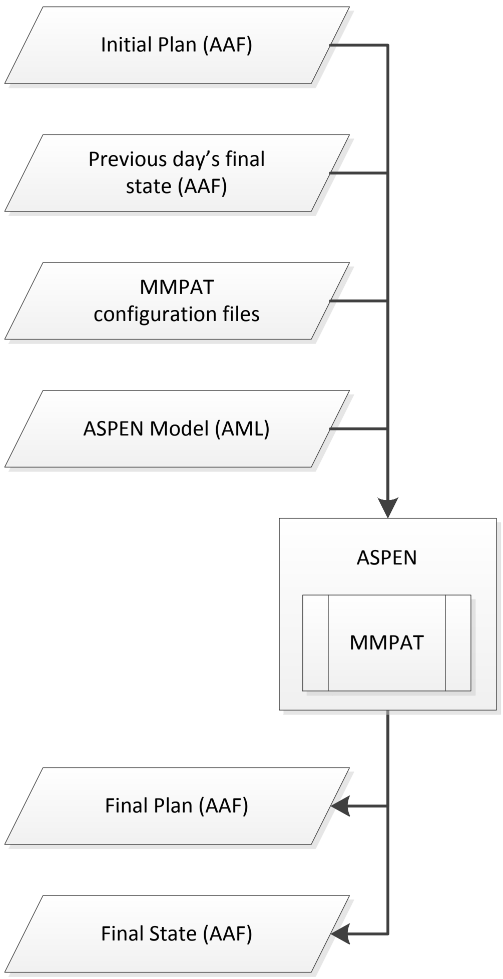


Figure 3. ASPEN plan generation

3. APGEN to ASPEN Translation

Fundamentally, APGEN modeling looks very similar to ASPEN modelling. *Activities* are modelled that levy constraints and post effects on shared states and resources.

APGEN	ASPEN
activity	activity
nonconsumable resource	non_depletable resource
consumable resource	depletable resource
state resource	state_variable

Table 1. APGEN-ASPEN syntax comparison

Note that non-consumable/non-depletable resources model a resource that is used over the interval and then released, e.g., power. Consumable/depletable resources model a resource that is used and not replenished (unless explicitly replenished via some other activity), e.g., energy. State resources/state variables model a single state over time, e.g., whether or not a heater is on or off. Henceforth, we shall refer to shared states and resources as *timelines*. Timelines are profiled, and the combined effects of various simultaneous activities are reflected in the profile of each timeline. Timelines can affect other timelines, allowing for a representation of cause and effect through the shared states and resources. Tab. 1 shows how similar the syntax is for activities and shared states and resources.

For example, if we were to examine the GENRIC_HTR_USE activity in the APGEN adaptation (which represents using any of the heaters onboard the rover), we would see different effects on resources based on the heater_group parameter (which is an input parameter for the activity). If the heater_group is MOBILITY_LEFT_SIDE, then our effect on the power_consumption is a function of MOBILITY_LEFT_HTR_POWER (among others). If it is MOBILITY_RIGHT_SIDE, then our effect on power_consumption is a function of MOBILITY_RIGHT_HTR_POWER (among others).

Similarly, with respect to sub-activities, if we were to examine the HAZCAM_FRONT activity in the APGEN adaptation (which represents using the forward hazard camera), we would see that the number of CAMERA_IMAGE_PAIR sub-activities generated is a function of the camera_selection array that is passed in on the creation of the activity.

The fundamental difference comes when modelling how the timeline profiles are generated, what effects the timelines have on other timelines, and what sub-activities are required in support of existing activities. To produce the resource profiles and to generate all sub-activities, APGEN sweeps forward in simulated schedule time. It creates activities, expands decompositions, and applies resource and state constraints in temporal order. Each clause is stitched together into a single program that is executed and dynamically generated as simulated time sweeps forward. This forward-sweep nature allows for very efficient modelling of resources and states and allows for fast decomposition of parent activities into the supporting cast of children, but it does come at a cost of not being able to modify the past. This often can be overcome by performing some initial modelling to get the values you need when planning a pre-cursor activity and then letting the subsequent activity play out as needed. Of course, one could simply choose a value for the past based on values that one has acquired for the present and then re-run the simulation with the acquired value. This is potentially linearly inefficient in the time that is required to generate a plan, going from linear time to quadratic (in the abstract sense), but the "time of a plan" is the number of propagations one would need to resolve the plan. For MER, multiple-propagation is not necessary as all models were constructed such that a single forward sweep was sufficient to correctly model the profiling of resources and the expansion of activities into sub-activities.

Our challenge then is to represent the resultant reasoning embedded in the execution of these code fragments using a declarative system. In the MER adaptation of APGEN, there were fundamentally only three different types of procedural code: 1) code blocks, 2) if-statements, and 3) loops. We explore each in more detail, using both APGEN and ASPEN modelling language.

Before continuing to the details of the translation, a quick outline of APGEN and ASPEN modelling is in order. Our examples will focus only on the generation of sub-activities.

APGEN includes the ability to model activities. Activities consist of attributes, parameters, resource usage, and decompositions (among others). We will focus on parameters and decompositions. Every activity has the built-in parameters **start** (the start time of the activity), **finish** (the end time of the activity), and **span** (the duration of the activity). Other parameters can be declared at the beginning of an activity. The decomposition is a code snippet that usually describes how to add activities to the schedule and modify the span of the existing activity. The following example is of a FOO_CHILD activity type, along with a FOO activity that consists of two FOO_CHILD activities, ordered back to back.

```
(1)  activity type FOO_CHILD
(2)  begin
(3)    parameters
(4)      A: local integer default to 0;
(5)  end activity type FOO_CHILD
(6)  activity type FOO
(7)  begin
(8)    nonexclusive_decomposition
(9)      child: instance default to "generic";
(10)     child1span: duration default to span;
(11)     call("FOO_CHILD", child) at start;
(12)     child1span = child.span;
(13)     call("FOO_CHILD", child) at start
         +child1span;
(14)     span = child1span + child.span;
(15) end activity type FOO
```

Lines 1-5 declare the child activity and lines 6-14 declare the base activity. Note lines 9 through 13 are to add sub-activities and to adjust the span of the activity. It should be noted that the **finish** parameter is automatically updated. The **call** statements add an activity of the type listed in the first parameter to the schedule.

Compare this to the same model encoded in ASPEN:

```
(1)  activity FOO_CHILD {
(2)    int A;
(3)  }
(4)  activity FOO {
(5)    int child1span, child2start, child2span;
(6)    dependencies =
(7)      child2start<-sum(start_time, child1span),
(8)      duration <-sum(child1span, child2span);
(9)    decomposition =
(10)     FOO_CHILD with (start_time->start_time,
(11)                     child1span<-duration),
(12)     FOO_CHILD with (child2span<-duration);
(13) }
```

Similar to APGEN, ASPEN activities always include a **start_time**, **end_time**, and **duration** parameter. ASPEN does not use mathematic expressions, but instead uses dependency assignments, e.g., x<-y indicates that x is constrained to equal y (but not vice versa). Note that in ASPEN we could forego the math and simply express a temporal relationship between the child activities, but our purpose here is to translate APGEN to ASPEN directly. The dependencies statements describe how to "hook up" the constraint network to various variables. We don't know when the values will be assigned, but we do know that a change in one will cause a propagation of the constraint network much like changing the value of a cell in Excel. The **with** clauses in the decomposition are also used to hook-up the constraint network, but in this case we are hooking up values in external activities. The direction of the arrow tells us which direction the dependency (or equality constraint, in this case) goes. This dictates the direction of propagation. In this example, we would never change the duration of a child activity from the parent, but changing a child activity's duration would result in a change to the duration of the parent activity.

3.1. Code Blocks

Code blocks might seem trivial to convert to declarative representations, but consider the following fragment of procedural code (in standard APGEN, which has a c-like syntax):

```
(1)  A : integer;
(2)  A = x;
(3)  call("FOO_CHILD", A, child);
(4)  A += y;
(5)  call("FOO_CHILD", A, child);
```

Clearly, the following declarative representation would fail:

```
(1)  int A;
(2)  decomposition =
(3)    FOO_CHILD with (A->A),
(4)    FOO_CHILD with (A->A);
(5)  dependencies =
(6)    A <- x,
(7)    A <- sum(A, y);
```

The result of such an ill-advised adaptation would be two sub-activities being generated with the same value being passed to each child, along with the added "feature" of having the value of A change continually and somewhat randomly as the constraint network propagates. Some systems would be able to detect that this can never be resolved and mark any activity introducing such a set of constraints as being faulty.

To address this, we clearly need to consider the value of A as it evolves over time. Specifically, for each step in a procedure where A changes, we need to keep track of the "new" A in that context and build our network accordingly. Thus, the correct declarative representation of the procedure would be the following:

```
int A_2, A_5;
decomposition =
  FOO_CHILD with (A_2->A),
  FOO_CHILD with (A_5->A);
dependencies =
  A_2 <- x,
  A_5 <- sum(A_2, y);
```

Now, should the value of x change, then the value of A_2 would change, and propagate to A_5, and result in a propagation down to both sub-activities of the appropriate value. But, let us draw our attention what happens when y changes. The value of A_2 is left unaltered and only the value of A_5 changes. This gives us an incremental capability to modify parameters in the context of an executing block of code without having to actually execute the block of code.

One challenge for this approach is faithfully representing short-circuiting of evaluation of components of Boolean expressions. Our translator does not explicitly deal with short-circuiting except for reporting warnings of where short-circuit structures exist in the code. For the MER adaptation, this had no impact on the correctness of the final adaptation. This is due to the lack of reliance on avoiding side effects, which is a testament to the overall high quality of the design of the model.

Finally, blocks of code should be seen as modular, copy-able extensions of code (if one thinks of executing a block of code as copying it). If a code block occurs in an IF statement or a loop, a more modular form of representation is called for in the model. To address this, we represent each code block as a separate activity. This allows for multiple copies of the block to exist with the same structure, representing different "executions" of the code block. Thus, extending our example:

```
activity Foo_code_block_1 {
  int x, y; //input parameters
  int A_2, A_5;
  decomposition =
    FOO_CHILD with (A_2->A),
    FOO_CHILD with (A_5->A);
  dependencies =
    A_2 <- x,
    A_5 <- sum(A_2, y);
}
```

3.2. IF Statements

If-statements in APGEN come in the standard c-like syntax:

```
if(<Boolean expression>) <block if true> else
    <block if false> ;
```

In ASPEN, we can easily code each block (both <block if true> and <block if false>) as a code-block activity. We use an enclosing activity for the if-statement using disjunctive decompositions and forcing the selection of which decomposition to use through a feature in ASPEN called the decomposition index. The following is an example of "if(a==1) <true_code_block> else <false_code_block>;".

```
activity foo_if {
  int a;
  dependencies =
    decomposition_index<-if(eq(a,1),0,1);
  decompositions=
    ( foo_if_true_code_block ) or
    ( foo_if_false_code_block);
}
```

Note that the **if**-function behaves similarly to the if-function in Excel. The Boolean **eq**-function returns true when all arguments are equal. Disjunctive decomposition indexing starts with 0 for the first decomposition choice.

3.3. Loops

We might be tempted to believe that the same structure that handles if-statements might be able to handle a loop (and in general we would be correct), but the key missing element for such a structure is the ability to branch back to previous code. Our translation approach is analogous to unrolling the loop into a sequence that has a length that varies based on the loop criteria.

All loops in APGEN can be expressed in the following c-like form:

```
while(<Boolean expression>)<block>;
```

We again take advantage of the decomposition index feature for our translation, but we use a recursion to the same activity type to represent the loop, including the changing of the variable used in the Boolean expression that determines the termination criteria. So, in APGEN we might have:

```
a : integer default to 5;
child: instance default to "generic";
while(a>0){
  call("foo_child", a, child);
  a--;
}
```

The ASPEN equivalent being:

```
(1)  activity foo_while {
(2)    int a, a_5;
(3)    dependencies =
(4)      a_5<-sub(a,1),
(5)      decomposition_index<-if(gt(a,0),0,1);
(6)    decompositions=
(7)      ( foo_while with(a_5->a)
(8)        foo_child with a->a) or
(9)      ( nop );
(10) }
```

The Boolean **gt**-function returns true if the first argument is greater than the second argument. Note that we do not include a block activity as the contents of the block are usually included in the loop activity. Also note the special **nop**-activity used where a decomposition selection is empty.

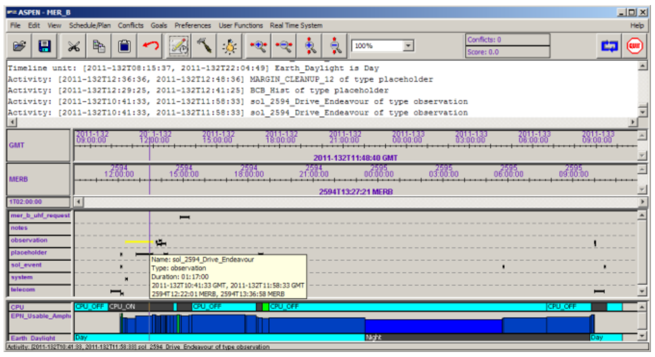
4. COMPARISON

Since some of the inputs and outputs are the same from ASPEN and APGEN (when adapted to MER), our approach was to compare outputs. Not surprisingly, our final adaptation showed little difference in outputs, with the exception of the display of certain profiled information from external power modelling system MMPAT.

In APGEN, MMPAT is integrated tightly into the core and APGEN renders the MMPAT timelines at a fidelity that is adjustable within the model (this also goes to the fidelity with which other APGEN entities generate profiles for other non-MMPAT based timelines) called MMPATfidelity, which for MER is set to between 9 and 12 minutes. APGEN interpolates values between these intervals.

In contrast, ASPEN tracks the worst-case value for these intervals and displays only that, making a kind of stair-step timeline versus a nice, smooth display.

Fig. 4 shows the ASPEN GUI with an example of one of the days that were planned using ASPEN. Fig. 5 shows the MAPGEN GUI of same day planned using MAPGEN. Even though the images are quite small here, it is easy to see the discritization that occurs when displaying the power timelines.



```
Timeline unit: [2011-132T08:15:37, 2011-132T22:04:49] Earth_Daylight is Day
Activity: [2011-132T12:36:36, 2011-132T12:48:36] MARGIN_CLEANUP_12 of type placeholder
Activity: [2011-132T12:29:25, 2011-132T12:41:25] BCB_Hist of type placeholder
Activity: [2011-132T10:41:33, 2011-132T11:58:33] sol_2594_Drive_Endeavour of type observation
Activity: [2011-132T10:41:33, 2011-132T11:58:33] sol_2594_Drive_Endeavour of type observation
```

```
Name: sol_2594_Drive_Endeavour
Type: observation
Duration: 01:17:00
2011-132T10:41:33 GMT, 2011-132T11:58:33 GMT
2594T12:22:01 MERB, 2594T13:36:58 MERB
```

Figure 4. ASPEN display of Sol 2594, MER B.

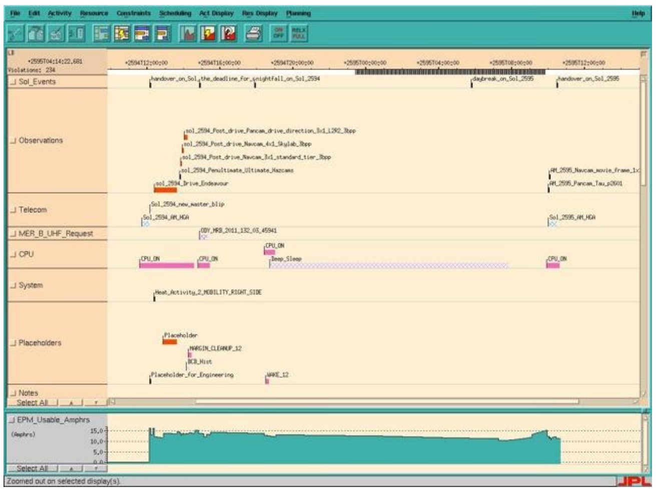


Figure 5. MAPGEN display of Sol 2594, MER B.

Another variance between ASPEN and MAPGEN was that durations for sub-second events were rounded to single second in ASPEN, leading to some small mis-alignments in the output files. This was quite rare and the sub-second durations in MAPGEN were approximations that could be rounded up without causing issues. Subsequent conversion techniques should use ASPEN's ability to model time at a higher fidelity than 1 second to accommodate sub-second events. ASPEN can model temporal intervals as small as a nano-second.

Both ASPEN and MAPGEN allow for the display of various time frames in the same user interface, so Mars time (or MER A time, to be more specific) can be displayed along with UTC or whatever time frame is preferred by the user. More importantly, input files can refer to these time frames and the internal time representation can align and translate these properly. This is particularly useful when aligning DSN operations with rover and orbiter operations.

It should be noted, however, that the MAPGEN GUI is much more responsive than the ASPEN GUI for multiple-day plans when displaying all of the power profiles. This is due in part to the fact that the MAPGEN GUI is embedded Motif, which runs extremely fast, and in part to the design of ASPEN's user interface.

An ASPEN instance starts as a server. The GUI is a java application that connects to the server via a socket. The GUI is responsible for synchronization with the server and maintains a lightweight copy of the schedule database. This allows for an arbitrary number of clients to connect to an ASPEN instance. While this architecture is particularly useful for distributing views of the same plan/schedule and allowing for contemporaneous modification, it is not really necessary for the single-user case that MER represents, and the overhead of copying and synchronizing the plan database across the socket costs us in terms of performance.

On the other hand, introducing temporal constraints between activities in ASPEN is a bit more straight-forward than with MAPGEN. In MAPGEN, a separate tool, called the constraint editor, is invoked and temporal constraints are introduced using the tool. In ASPEN, temporal constraints can be added and deleted directly through the user interface, leading to a more fluid user experience. But, the generation of skeleton plans by the skeleton plan generator that is in common use nowadays obviates the need to edit the temporal constraints very often, so this is not so much of a gain in practice.

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Copyright 2012 California Institute of Technology. Government sponsorship acknowledged.

5. REFERENCES

[1] "Mars Exploration Rover project," NASA/JPL document NSS ISDC 2001 27/05/2001.

[2] Ai-Chang, M., Bresina, J., Charest, L., Chase, A., Chengjung Hsu, J., Jonsson, A., Kanefsky, B., Morris, P., Rajan, K., Yglesias, J., Chafin, B. G., Dias, W. C., and Maldague, P. F., "MAPGEN: Mixed-Initiative Planning and Scheduling for the Mars Exploration Rover Mission", IEEE Intelligent Systems, 2004.

[3] Maldague, P., Ko, A., Page, D., and Starbird, T., "APGEN: A multi-mission semi-automated planning tool." First International NASA Workshop on Planning and Scheduling, Oxnard, CA, 1998.

[4] Rabideau, G., R. Knight, S. Chien, A. Fukunaga, and A. Govindjee. "Iterative Repair Planning for Spacecraft Operations using the ASPEN System", Proc. Intl. Symp. Of Artificial Intelligence, Robotics and Automation for Space, 1999.