Planning for Human Execution of Procedures Using ANML

Mark Boddy
Adventium Labs
Minneapolis, MN
mark.boddy@adventiumlabs.org

Russell Bonasso
TracLabs
Houston, TX
bonasso@traclabs.com

Abstract

Over the past several months, we have been engaged in the definition and implementation of automated planning capabilities for supporting NASA operations personnel in planning and executing operations on the International Space Station (ISS). For this activity, we have chosen to use the *Action Notation Modeling Language* (ANML). In this process, we have exercised much of ANML's considerable flexibility, including exploring several different means of specifying goal decomposition, rather than the task decomposition directly supported in ANML. We have also encountered unexpected semantic ambiguities in the language, for example related to the use of functional fluents with non-numeric ranges. In this paper, we briefly describe the domain, then discuss the modeling challenges arising in this domain and how we have used ANML to address those challenges, and some lessons learned about ANML in the process.

1. Introduction and Motivation

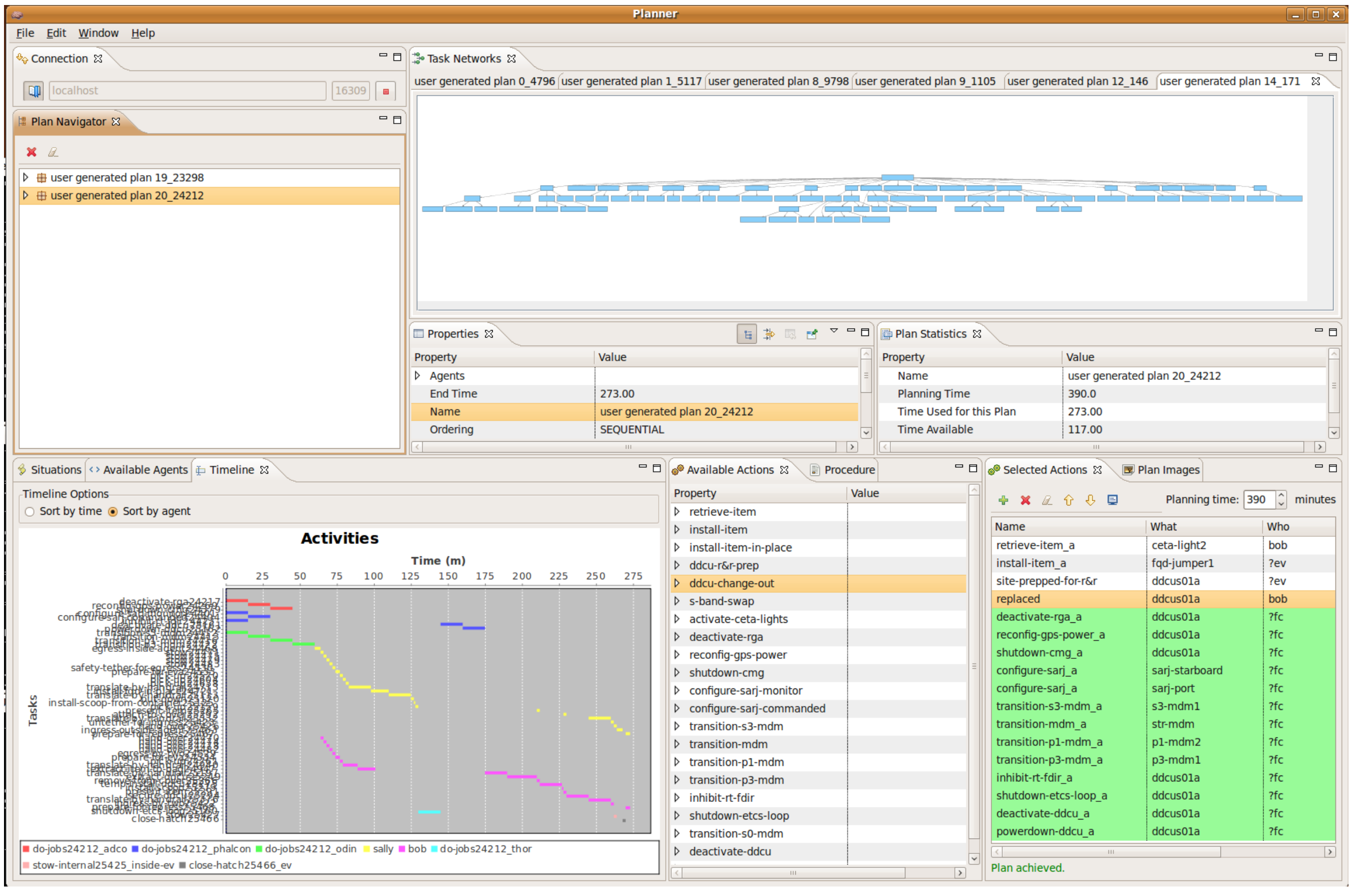
Over the past several months, we have been engaged in the definition and implementation of automated planning capabilities for supporting NASA operations personnel in planning and executing operations on the International Space Station (ISS). Specifically, we have been working on providing tools that support the construction and maintenance of a set of *procedures* detailing how operations must be carried out, and the generation from these procedures of a document detailing what will be done during a given operation (a *flight note*).

In this effort, we have been engaged in implementation and analysis activities aimed at different parts of the operations planning process. First, in consultation with NASA personnel, we identified a target application domain and a reference scenario: Extra-Vehicular Activity (EVA) for the ISS. Through an extended period of interaction with the flight controllers responsible for those operations, we investigated and documented both the scenario and the planning process itself. The results of this analysis are summarized below in Section 3.and reported in more detail in (Bonasso, Boddy, and Kortenkamp 2009).

With those results in hand, we have subsequently been pursuing parallel development of several required capabilities. First, we have identified a set of expressive extensions to the current representation of procedures used by the operations planners, which can be used to provide additional support for the planning process. Specifically, we are designing means to support representation and reasoning about *composing* smaller procedures into larger, more complex operational plans, reasoning in a more principled way about the *duration* required for these operations, and about interactions between different operations, in terms of the *resources* required. The resulting models are a combination of the original ISS procedures, written in the *Procedure Representation Language* (PRL) (Kortenkamp et al. 2008), and an intermediate representation of the additional information described above, which can then be translated into a *planning* model in any of several planning languages, and manipulated using a tool for mixed-initiative planning.

This implementation is still very much a work in progress. At this point, we have defined a structured process for eliciting the additional information required from the domain experts and building the intermediate planning representation described above. This process is described in detail in (Bonasso and Boddy 2010). We have constructed a prototype planning tool. A screenshot of the user interface for that tool is shown in Figure 1. Finally, we have been working on understanding the kinds of planning models and capabilities required to provide effective support. This process has consisted of two parallel efforts. In the first, we have implemented a hand-translated model of the augmented operations procedures, using the AP planner and its input language, pddl-e (Applegate, Elsaesser, and Sanborn 1990). Generating plans with AP provides information about the consistency and coverage of the model we have constructed. Showing the plans generated by this process to the operations planners provides us with feedback as to whether the extended capabilities we are providing are the ones needed.

In parallel, we have been investigating the use of a more recently-developed planning language: NASA's *Action Notation Modeling Language* (ANML) (Smith, Frank, and Cushing 2008). ANML is an attractive target language for several reasons. First, ANML is under active development. This has resulted in a feedback cycle in which some features of the language have been modified, based on the results of our attempts to use it to build models of significant complexity. Second, ANML is sufficiently expressive for us to explore a wide range of alternative modeling choices. Third, ANML is deliberately defined to have a semantics independent of the choice of a particular planner or executive. As discussed briefly below in Section 4.and in more detail in (Smith, Frank, and Cushing 2008), an ANML model can



Plan Navigator:
- user generated plan 19_23298
- user generated plan 20_24212

Task Networks: user generated plan 0_4796 | user generated plan 1_5117 | user generated plan 8_9798 | user generated plan 9_1105 | user generated plan 12_146 | user generated plan 14_171

Properties:

Property	Value
Agents	
End Time	273.00
Name	user generated plan 20_24212
Ordering	SEQUENTIAL

Plan Statistics:

Property	Value
Name	user generated plan 20_24212
Planning Time	390.0
Time Used for this Plan	273.00
Time Available	117.00

Timeline Options: Sort by time [✓ Sort by agent]

Available Actions:

Property	Value
retrieve-item	
install-item	
install-item-in-place	
ddcu-r&r-prep	
ddcu-change-out	
s-band-swap	
activate-ceta-lights	
deactivate-rga	
reconfig-gps-power	
shutdown-cmg	
configure-sarj-monitor	
configure-sarj-commanded	
transition-s3-mdm	
transition-mdm	
transition-p1-mdm	
transition-p3-mdm	
inhibit-rt-fdir	
shutdown-etcs-loop	
transition-s0-mdm	
deactivate-ddcu	

Selected Actions — Planning time: 390 minutes

Name	What	Who
retrieve-item_a	ceta-light2	bob
install-item_a	fqd-jumper1	?ev
site-prepped-for-r&r	ddcus01a	?ev
replaced	ddcus01a	bob
deactivate-rga_a	ddcus01a	?fc
reconfig-gps-power_a	ddcus01a	?fc
shutdown-cmg_a	ddcus01a	?fc
configure-sarj_a	sarj-starboard	?fc
configure-sarj_a	sarj-port	?fc
transition-s3-mdm_a	s3-mdm1	?fc
transition-mdm_a	str-mdm	?fc
transition-p1-mdm_a	p1-mdm2	?fc
transition-p3-mdm_a	p3-mdm1	?fc
inhibit-rt-fdir_a	ddcus01a	?fc
shutdown-etcs-loop_a	ddcus01a	?fc
deactivate-ddcu_a	ddcus01a	?fc
powerdown-ddcu_a	ddcus01a	?fc

Plan achieved.

Figure 1: The ISS EVA Planning Tool Prototype

be viewed as a set of constraints on execution, rather than as input to a particular planning algorithm. This provides a valuable degree of independence from any particular planning algorithm or approach.

In the rest of this paper, we discuss in turn the application domain (Section 2), and a reference scenario (Section 3). We then present a very brief review of some key features of ANML (Section 4) and then provide some examples showing how we have used ANML to model some of the more problematic features arising in this domain. We then conclude with a summary and a discussion of future work.

2. Building Complex Procedures for ISS Operations

At this time, operations planners for various facets of ISS operations are increasingly adopting, or making plans to adopt, a standard syntax and semantics for authoring procedures, known as the *Procedure Representation Language* (PRL) (Kortenkamp et al. 2008).

PRL both captures the form of traditional procedures and allows for automatic translation into code that can be executed by NASA-developed autonomous executives (Bonasso, Kortenkamp, and Thronesbery 2003; Verma et al. 2006). PRL provides for access to spacecraft and habitat telemetry, includes constructs for human centered displays, allows for the full range of human interaction, and allows for automatic methods of verification and validation. Finally, PRL is being developed with a graphical authoring system that enables non-computer specialists to write automated procedures (Kortenkamp, Bonasso, and Schreckenghost 2007). PRL and the tools being developed for creating, modifying, checking, and executing PRL procedures will go a long way in simplifying and standardizing operations planning. However, the generation of flight notes from previously-defined procedures is still almost entirely a manual process.

For many operations planning tasks, including the ISS Extra-Vehicular Activity (EVA) operations that we have addressed, the majority of procedures are simple ones, as shown in the following example of how a single crew member moves from the airlock to outside of the ISS:

```
Procedure (tether)
1. Thermal cover - open
2. Egress AIRLOCK
3. Attach tether to left D-ring extender
4. Verify tether config
```

However, not all procedures are so simple. Procedures can be *fragmentary*: several of them may need to combined to construct a flight note. They may also be *conditional*, in the sense that only parts of a given procedure may be needed for any given operation. Some procedures are *hierarchical*, in the sense that one step of a given procedure may refer to another procedure (or, in some cases, to a step or steps contained in another procedure). Planning for complex operations may require reasoning about resources such as oxygen, fuel, or tools, choosing alternative sub-actions, synchronized or overlapping action by multiple agents, and reasoning about preconditions and effects at planning time. As a language designed primarily to support execution, PRL provides at best minimal support for any kind of reasoning about these features.

3. EVA Mission Scenario

In this section, we summarize the application motivating the work reported in the rest of the paper. For more details, please see (Bonasso, Boddy, and Kortenkamp 2009). When we began this research, the PHALCON (Power, Heat And Light CONtrol) and EVA (Extra Vehicular Activity) flight controllers for ISS had recently conducted a mission wherein one of the EVA tasks was the removal and replacement of a DC-to-DC Converter Unit (DDCU). The main EVA tasks for the six-hour operation were:

1. Crew egress the airlock
2. Retrieve CETA (Crew and Equipment Translation Aid) Light 2
3. Relocate CETA-cart 2 from the P1 truss to the P3 truss
4. Remove and Replace the DDCU 1A on the S0 truss
5. Crew ingress the airlock

We constructed a model of both the planning and procedural elements of this operation. The PHALCON tasks were derived from systems operations data files (SODFs), word files that were part of the repository of flight controller procedures. For the EVA tasks, we needed to go over the written description of the full six-hour scenario because the EVA flight controllers develop each EVA activity essentially from scratch. While the activities of connecting and disconnecting equipment to and from the station are fairly routine, each EVA is unique both in the number and types of tasks and in the starting locations of the equipment and tools used in the tasks. But these written documents have little or no planning information associated with them, such as activity duration, purpose, preconditions or constraints. To derive this planning information, we started with a given EVA or PHALCON procedure, discussed each step with the cognizant flight controller, and then rewrote the procedure to include the information required for planning. We then used this information to construct a planning model for the tasks involved.

From this analysis we distilled seven intermediate sub-procedures and ten leaf-level PHALCON procedures, and thirteen intermediate and 48 leaf-level EVA sub- procedures. An example of the breakout of sub-procedures for EVAs is shown in Figure 2. This breakout shows how to retrieve the CETA light with an agent starting outside the ISS, located at the AIRLOCK. The agent gets the ORU (orbital replacement unit) bag for the light, travels to the light's location, removes the light and stows it in the bag, then takes it back to the airlock. One interesting aspect of this procedure, and of all EVA procedures that involve moving the agents from place to place is the limitation of the safety tether. If the target location is farther away than the 55-foot safety tether can reach, the agent has to travel to the location of another tether, usually of the 85-foot variety, and perform a tether- swap before moving on.

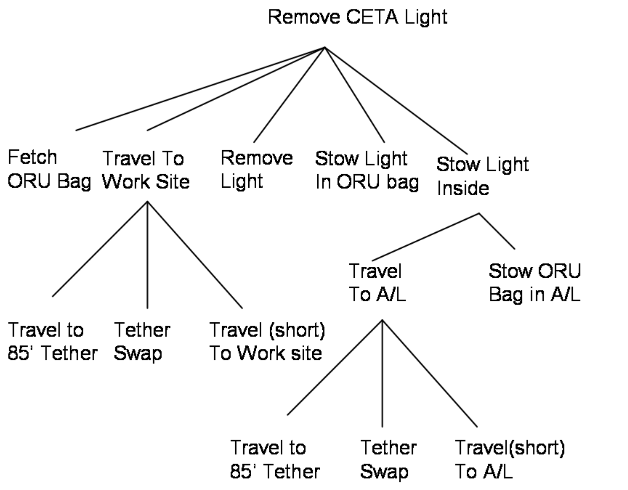


Figure 2: EVA sub-procedures for removing a CETA light

Multi-agent EVA procedures involve two agents working together to accomplish the task. In the case of the replacing the DDCU, for example, during the installation of the spare DDCU, agent1 has the old DDCU attached to his body restraint tether and agent2 holds the spare DDCU on a stanchion mount cover. The coordinated exchange is as follows:

1. Agent2 presents the spare DDCU to agent1.
2. Agent1 grabs the spare via a scoop on the spare and agent2 detaches the spare from the cover.
3. Agent1 inserts the spare in position and then presents the old DDCU to agent2
4. Agent2 attaches the cover on the old DDCU and then stows it
5. Agent1 bolts the spare in place

Clearly, this domain requires the representation of parallel activities by multiple agents, which may be either coordinated or asynchronous.

4. The Action Notation Modeling Language

The *Action Notation Modeling Language* (ANML) is a fairly recent, and to some extent still evolving, attempt to combine the best features of several different languages for defining planning and scheduling problems. ANML supports preconditions and effects, hierarchical task decomposition, complex constraints among tasks, and a flexible specification of goals, rather than just goals of achievement. In this section, we provide just enough of an introduction to ANML to permit the reader to understand the examples provided in the rest of the paper. For a more complete description of ANML's design rationale and usage, including comparison to previous planning languages, see (Smith, Frank, and

```
1  action Stow_external(crew ev1, ev2, object item) [duration] {
2     [start] {located(ev1) == INTRA_VEHICLE;
3              located(ev2) == AIRLOCK};
4     [all] contains {s1 : Stow(ev2 , item);
5                     s2 : Hand_over(ev1 , item)};
6     start(s1) == start(s2);
7     end(s1) == end(s2);
8     [all] located(item) == INTRA_VEHICLE :-> AIRLOCK }
```

Figure 3: A simple ANML action declaration

Cushing 2008). For the most complete definition of ANML syntax of which we are aware, see (ANML). To the best of our knowledge, ANML's semantics are still somewhat in flux, but the basic structure is laid out in the paper cited above.

Figure 3 shows a fairly simple ANML action declaration, illustrating several relevant features. This declares an action called Stow_external which takes three typed parameters: ev1, ev2, and item. The [**duration**] tag indicates that the execution time of this action is not specified, subject to the other constraints placed on this action. Lines 2 and 3 declare a temporally-scoped constraint, specifying required values for two fluents at the time this action starts. Lines 4 and 5 define a decomposition for Stow_external, consisting of two labelled subactions, Stow(ev2,item) with label s1 and Hand_over(ev1,item) with label s2. These labels are used to refer to specific instances of the subactions, so that in lines 6 and 7 we can add constraints such that the actions must start and end at the same times. Finally, line 8 combines a precondition with an effect. The temporal condition [all] indicates that this takes place over the entire extent of the action Stow_external. At the start of the action, the fluent located(item) is tested for equality with the constant INTRA_VEHICLE. The :−> notation means that the value of that fluent is undefined over the extent of the action, then is set to the constant AIRLOCK at the action's end point. ANML does not include an infinitesimal duration as in PDDL 2.1. Instead, temporal conditions can be defined over both open and closed intervals, permitting the modeler to define their own semantics for consecutive durative actions that start and end at the same point.

5. Modeling ISS Operations in ANML

In this section, we show how some of the information required for operations planning, currently represented either in text documents or in the expertise of individual flight controllers, can be effectively represented in ANML.

As discussed in the previous section, ANML is quite flexible and expressive. In the example in that section, much of the complexity involved the temporal extent of durative actions and the times during those actions that conditions may be tested and effects may occur. It is straightforward to use this flexibility to emulate the precondition and effect definitions for durative actions in PDDL, with the single exception of not having an infinitesimal duration. In this section, we will explore some other areas in which ANML's flexibility proves useful.

Combining Idioms

In AI planning, there are three main ways of specifying the relationships among actions required in a valid plan. In *classical planning*, action effects are required to satisfy preconditions of later actions. In *task decomposition*, actions are specified as parts of the expansion of a higher-level action, possibly with some additional constraints among them. In *goal decomposition*, actions must be present to satisfy goals specified in the expansion of a higher-level action. In ANML, any or all of these can be combined in the same model. Equally, any one or two of them will suffice to construct a domain model in most cases.[1] As already shown in Figure 3, task decomposition is directly supported in ANML. As we will show in a slightly more extended example, preconditions and effects can be combined in a very natural way with task decomposition.

In Figure 4, we see definitions for several actions. The action Stow_it (line 1) has no preconditions or effects, but specifies a decomposition into two sub-tasks, Pickup (line 4) and Put_away (line 10). Pickup has a precondition (line 6), a combined precondition and effect involving a single fluent (line 7), and an effect (line 8). Put_away has the same structure, with a different set of conditions and effects. Next we have definitions for two movement actions, Translate_by_handrail and Translate_by_CETA. [2] In both of these actions, the duration required is computed as a function of the locations between which the movement occurs. Translate_by_handrail also contains a bound on the maximum distance over which it can be used. Given the following problem definition, we can use the actions in Figure 3 to generate a plan.

```
// Start and end times:
start := 0;
end := 100;

// Statics:
MAX_HANDRAIL_DIST := 50;
distance(l_21 ,l_72) = 30;
distance(l_21 ,l_17) = 60;
distance(l_72 ,l_17) = 80;
distance(l_17 ,AIRLOCK) = 20;

// Initial situation
[0] located(pgt_31) := l_17;
[0] located(Bob) := l_21;
```

[1]Erol, Hendler, and Nau (Erol, Hendler, and Nau 1994) demonstrate that HTN planning is strictly more expressive than goal regression, but for task hierarchies without recursion or cycles, this does not become an issue.

[2]CETA stands for Crew and Equipment Translation Aid.

```
action Stow_it(crew ev, object item, location stowage) [duration] {
   [all] contains ordered(Pickup(ev,item),Put_away(ev,item,stowage)) }

action Pickup(crew ev, object item) {
   duration := 5;
   [start] located(ev) == located(item);
   [all] possesses(ev,item) == FALSE :-> TRUE;
   [end] located(item) := POSSESSED }

action Put_away(crew ev, object item, location stowage) {
    duration := 10;
   [start] located(ev) == stowage;
   [all] possesses(ev,item) == TRUE :-> FALSE;
   [end] located(item) := stowage }

action Translate_by_handrail(crew ev, location loc1, loc2) {
   distance(loc1,loc2) <= MAX_HANDRAIL_DIST;
   duration := handrail_translation_time(loc1,loc2);
   [all] located(ev) == loc1 :-> loc2 }

action Translate_by_CETA(crew ev, location loc1, loc2) {
   duration := ceta_cart_translation_time(loc1,loc2);
   [all] {located(ev) == loc1 :-> loc2;
          located(CETA) == loc1 :-> loc2} }
```

Figure 4: Combining Task Decomposition and Goal Regression in ANML

```
[0] located(CETA) := 1_72;

// Goal:
Goal [all] contains
    Stow_it(Bob, pgt_31 ,AIRLOCK);
```

The Goal statement simply requires that the action Stow_it(Bob,pgt_31,AIRLOCK) occur, with its entire extent within the planning horizon defined by **start** and **end**. A valid ANML expansion for this problem appears in Figure 5. Shaded actions show the task decomposition. The movement actions were added as required to satisfy action preconditions. Both translation actions are needed at the beginning of the plan, to get Bob to where the CETA cart is, because it is too far from his initial location to use the handrails to get to the initial location of pgt_31.

Goal-Reduction Planning in ANML

For the EVA domain, we sometimes find it useful to use *goal decomposition*, rather than task decomposition. As in the examples above, task decomposition involves expanding non-primitive actions by placing a set of sub-actions and associated constraints in the plan. Goal decomposition involves expanding non-primitive tasks by adding a partially-ordered set of subgoals and associated constraints to the plan. Each subgoal is then established by a task, if necessary. The rationale for using goal decomposition is that in some situations, the planners think in terms of a series of goals to achieve, rather than actions to execute. The primary distinction is that these goals may not require any action in order to be established.

But whereas task decomposition is directly supported in ANML and has a reasonably well-defined semantics based on the conditions, effects, and temporal extent of sub-actions, goal decomposition leaves considerably more for the implementer to define. What is the extent over which the subgoal, once established, must remain true? If a subsequent goal is already established, do the preceding goals still need to be established? For two consecutive subgoals, is it the times at which those subgoals are established that must be ordered, or must we order the actions that establish them? Any planner doing goal decomposition, for example the AP planner previously cited, or Wilkins' SIPE-2 (Wilkins 1990), must provide an answer to these questions. The unique feature of ANML is that, due to the focus on execution semantics, these answers need to be provided in the model itself: we cannot depend on any particular planning algorithm to enforce the desired relationships.

We want to use a mixture of goal and task decomposition, writing something like the following:

```
action foo () [duration] {
[all] contains
    ordered(achieve(possesses(ev, bag)),
    achieve(located(ev, light_place)),
    install(ev,light)) }
```

where the first two "subtasks" in the decomposition specify subgoals to be achieved. Notice that in this model, there are no constraints that the goals, once established, persist over any particular temporal extent. But even with a propositional model, this will not have the desired effect. The problem is that the parameters of the achieve sub-actions in this example will be evaluated and "passed" by value, rather than by reference. So, within the action definition for achieve, the parameter will evaluate to the fluent's value (in the first instance, either TRUE or FALSE), rather than to the fluent itself. Thus, the fluent cannot be affected by the sub-action. This argument exposes another interesting issue: with regard to what time, exactly, is the sub-action's parameter to be evaluated? For ANML, the answer is apparently not yet

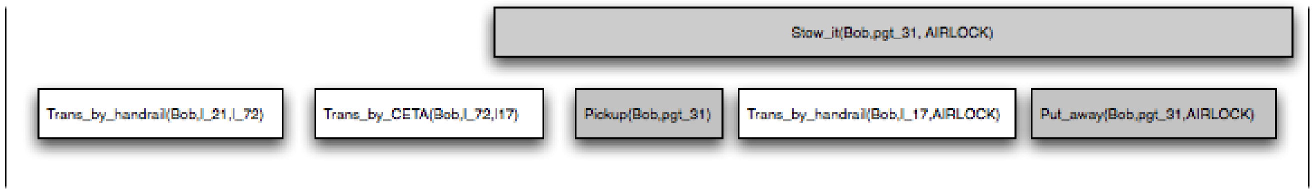

Stow_it(Bob,pgt_31, AIRLOCK)
Trans_by_handrail(Bob,l_21,l_72)
Trans_by_CETA(Bob,l_72,l17)
Pickup(Bob,pgt_31)
Trans_by_handrail(Bob,l_17,AIRLOCK)
Put_away(Bob,pgt_31,AIRLOCK)

Figure 5: ANML plan, generated using both task decomposition and classical planning

specified. This problem does not arise for parameters that evaluate to static or temporally definite relations, or if we fix the time of evaluation, for example using a local variable in the action definition.

We can work around this problem of parameter evaluation as in the following model:

```
action foo () [duration] {
  [all] contains
      ordered(ach_possesses(ev, bag),
              ach_located(ev, light_place),
              install(ev,light)) }
```

where we define a set of actions to achieve sub-goals involving specific predicates. For example, here is the definition for ach_possesses:

```
action ach_possesses(crew agent, object
    item) [duration] {
  {[start] possesses(agent,item) == TRUE} ||
  {[start] possesses(agent,item) == FALSE;
   (start,end) contains
       possesses(agent,item) == UNDEFINED;
   [end] possesses(agent,item) == TRUE} }
```

We still have not managed to constrain the extent over which the sub-goal must remain established. That can be done, but the best way we have found to do it thus far is somewhat messier. Here is an example of an action definition which includes three sequential subgoals, each of which is required to remain established until the start of the action establishing the next subgoal (or the end of the overall action, if there is no next subgoal).

```
action EVA (crew ev, object stringer,
    light, jumper, fqd) [duration] {

  // Establish placeholder interval labels
      for subactions. These may be
      degenerate.

  [all] contains ordered(s1: TRUE, s2:
      TRUE, s3: TRUE, s4: TRUE);

  // Add interval "preconditions" to
      prevent clobbering.

  [start(s2)] possesses(ev,stringer) ==
      TRUE;
  [start(s3)] retrieved(ev,light) == TRUE;
  [start(s4)] installed(ev,jumper) == TRUE;
  [end] installed(ev,fqd) == TRUE;

   // Add only needed actions.

  [start] possesses(ev,stringer) == TRUE ||
   {[start(s1)] possesses(ev,stringer) ==
       FALSE;
    (start(s1),end(s1)) contains
        possesses(ev,stringer) ==
        UNDEFINED;
    [end(s1)] possesses(ev,stringer) ==
        TRUE};

   [end(s1)] retrieved(ev,light) == TRUE ||
   {[start(s2)] retrieved(ev,light) ==
       FALSE;
    (start(s2),end(s2)) contains
        retrieved(ev,light) == UNDEFINED;
    [end(s2)] retrieved(ev,light) == TRUE};

   [end(s2)] installed(ev,jumper) == TRUE
       ||
   {[start(s3)] installed(ev,jumper) ==
       FALSE;
    (start(s3),end(s3)) contains
        installed(ev,jumper) == UNDEFINED;
    [end(s3)] installed(ev,jumper) ==
        TRUE};

  [end(s3)] installed(ev,fqd) == TRUE ||
   {[start(s4)] installed(ev,fqd) == FALSE;
    (start(s4),end(s4)) contains
        installed(ev,fqd) == UNDEFINED;
    [end(s4)] installed(ev,fqd) == TRUE}}
```

Compositional Action Definitions

In this section, we describe one additional use of ANML's modeling flexibility in this domain. For many actions executed by both ISS crew and flight controllers, it is desirable to add an explicit verification step, a "check" that the action has had the desired effect. This is an execution-time check, not a simple confirmation in the planning model that some condition is now satisfied. In the domain elicitation process described in (Bonasso and Boddy 2010), the supported sequence is to define the action itself, then to add a "verify" tag, indicating that such a check is required. There are several ways this may be accomplished, each with different properties.

We start by defining an action to accomplish the final installation of a "DC-to-DC converter unit" (DDCU) as follows:

```
action Secure_ddcu(crew ev,
    dc_to_dc_converter_unit ddcu)
{
  duration := 15;
```

```
    [start] {located(item) == located(ev);
                  inserted(ddcu, located(ddcu))};
    [all] exists (pgt_with_turn_setting pgt)
        {possessed_by(pgt) == ev};
  [end] installed(item) := TRUE
}
```

This is a primitive action with two parameters: a crew member ("ev") and a piece of equipment ("ddcu"). In addition, the crew member needs to have possession of a particular type of tool (a "power-grip-tool with turn setting"). Which specific tool does not matter, as long as ev continues in possession of it throughout the action. The ddcu needs to have been inserted in the appropriate location, and the person needs to be at the same place. The action's duration is specified as 15 minutes, and its final effect is that the ddcu is now "installed." In this specific case, the difference between *inserted* and *installed* is that the fastening bolts have been tightened down.

Now, suppose that in the specification of this action, the user has indicated that the successful installation of the DDCU must be manually verified after the installation is completed, before this action can be considered complete. We can modify the action definition given above to reflect this, in several different ways.

First, here's the declaration for a "check" action:

```
//  Check action for arity-1 predicates

action CHECK(string predicate_name, object
    thing1)
{
 duration := 1
}
```

There are several ways in which to add a requirement that this check be performed to the planning model. One would be to add an additional level to the task hierarchy by redefining Secure_ddcu:

```
action Secure_ddcu(crew ev,
    dc_to_dc_converter_unit item) [duration]
{
 [all] contains
      ordered(Secure_ddcu_1(ev,item),
      Check("installed",item))
}

action Secure_ddcu_1(crew ev,
    dc_to_dc_converter_unit item)
{
 motivated;
 duration := 15.0;

 [start] {located(item) == located(ev);
        inserted(item,located(item));
        exists (pgt_with_turn_setting pgt)
            possessed_by(pgt) == ev};

 [end] installed(item) := TRUE
}
```

Note that in this definition, the duration of the overall action is now unspecified, other than being constrained to be greater than the sum of the durations of Secure_ddcu as originally defined and the CHECK action.

Alternatively, we can decide that the CHECK will be accomplished as part of the orginally-specified 15-minute duration:

```
action Secure_ddcu(crew ev,
    dc_to_dc_converter_unit item)
{
 duration := 15.0;

 [all] contains
      ordered(Secure_ddcu_1(ev,item,duration),
      Check("installed",item))
}

action Secure_ddcu_1(crew ev,
    dc_to_dc_converter_unit item, number
    overall_duration)
{
 motivated;
 duration := overall_duration -
      CHECK_DURATION;          //subtract a
      constant value for CHECK.

 [start] {located(item) == located(ev);
        inserted(item,located(item));
        exists (pgt_with_turn_setting pgt)
            possessed_by(pgt) == ev};

 [end] installed(item) := TRUE
}
```

It is also possible to add the verification action without adding additional level to the hierarchy, again either including the duration of that action within the original specification, or in addition to it. For brevity, we provide only the latter example:

```
action Secure_ddcu(crew ev,
    dc_to_dc_converter_unit item) [duration]
{
 secure_ddcu_duration := 15;

 [start] {located(item) == located(ev);
        inserted(item,located(item));
        exists (pgt_with_turn_setting pgt)
            possessed_by(pgt) == ev};

 [start + secure_ddcu_duration]
     installed(item) := TRUE;

 [start + secure_ddcu_duration, end]
     contains Check("installed",item)
}
```

Another interesting property of this definition is that the verification action is constrained to be after the effects of the original definition of Secure_ddcu, but not necessarily immediately after, yet is still constrained to complete before any action ordered after Secure_ddcu.

6. Summary and Conclusion

All of the examples used in this paper were drawn from the planning model we have been constructing in consultation with the cognizant operations planners. In each case, the presence or absence of a particular modeling requirement has been driven by the application domain. Overall, our experience with ANML has been positive. The language is flexible enough to do a great deal (including getting oneself into trouble). The focus on execution as a criterion for validity makes things significantly simpler and cleaner. The apparent resemblance between ANML and a block-structured, strongly-typed programming language can be misleading at first. But as with any declarative formalism, an ANML model is not "executed" in any meaningful sense. Rather, it describes a set of relationships.

At this point, there appears to be at least one significant issue to be resolved in the ANML semantics, which is how to treat non-numeric functional fluents, in terms of both when, and with respect to what temporal condition, they will be evaluated. The most significant remaining expressive limitation in ANML is that it cannot as currently defined be used to reason about continuous change (discrete changes in continuous values, yes). Removing this restriction would, at least for some of us, significantly increase the appeal of using the language, though it will significantly complicate the job of implementing reasonably complete ANML planners. There are some syntactic shortcuts that could be useful as well, for example supporting the specification of a wider variety of domain axioms, but those are minor points, because they do not involve changes to the fundamental semantics, the way that either continuous change or a more general treatment of functional fluents will.

7. Acknowledgements

This work is supported by NASA, under contract number NNX09C1A7C. We are indebted to NASA-JSC flight controllers Wayne Wedlake and Dave Crook for providing extensive domain expertise for this work. Thanks as well to David Smith and Will Cushing for explanations and discussion of the ANML language, and to Jeremy Frank for discussions regarding the relationship between planning and procedural models. Dave Kortenkamp is our colleague on this project, and has contributed significantly to defining the integration of planning information into PRL.

References

ANML. http://code.google.com/p/anml/.

Applegate, C.; Elsaesser, C.; and Sanborn, J. 1990. An architecture for adversarial planning. *IEEE Transactions on Systems, Man, and Cybernetics* 20(1):186–194.

Bonasso, P., and Boddy, M. 2010. Eliciting planning information from subject matter experts. submission to ICAPS 2010 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS).

Bonasso, P.; Boddy, M.; and Kortenkamp, D. 2009. Enhancing nasa's procedure representation language to support planning operations. In *Proceedings of the International Workshop on Planning and Scheduling for Space (IWPSS)*.

Bonasso, R.; Kortenkamp, D.; and Thronesbery, C. 2003. Intelligent control of a water recovery system: Thryears in the trenches. *AI Magazine* 19–44.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. Semantics for hierarchical task-network planning. Technical report.

Kortenkamp, D.; Dalal, K. M.; Bonasso, R. P.; Schreckenghost, D.; Verma, V.; and Wang, L. 2008. A procedure representation language for human spaceflight operations. In *iSAIRAS 2008*.

Kortenkamp, D.; Bonasso, R.; and Schreckenghost, D. 2007. Developing and executing goal-based, adjustably autonomous procedures. In *AIAA InfoTech@Aerospace Conference*.

Smith, D.; Frank, J.; and Cushing, W. 2008. The anml language. In *International Conference on Automated Planning and Scheduling*.

Verma, V.; Jónsson, A.; Pasareanu, C.; and Iatauro, M. 2006. Universal executive and plexil: Engine and language for robust spacecraft control and operations. In *American Institute of Aeronautics and Astronautics Space Conference*.

Wilkins, D. E. 1990. Can AI planners solve practical problems? *Computational Intelligence* 6(4):232–246.