Eliciting Planning Information from Subject Matter Experts

Pete Bonasso* , Mark Boddy*

*Traclabs, Inc, 1012 Hercules, Houston, TX 77059, bonasso@traclabs.cov
**Adventium Enterprises, 111 3rd Ave South, Suite 100, Minneanapolis, MN 55401, mark.boddy@adventiumlabs.org

Abstract

Over the past several months, we have been engaged in layering planning information onto execution procedures for supporting NASA operations personnel in planning and executing activities on the International Space Station (ISS). The procedures are captured in the Procedural Representation Language (PRL). The planning information is to be integrated with the procedural information using a PRL authoring system. This paper describes an initial design for eliciting planning information by the domain experts who created the procedures. The goal is to generate actions in standard planning languages that automated planners can use to generate executable plans. Of particular note is that the resulting action representations support both goal and action HTN decompositions.

Introduction and Motivation

There have been a number of recent efforts, most notably the Automation for Operations (A4O) initiative (Frank 2009), to provide NASA flight controllers with activity planning and execution aids by leveraging maturing execution (Vera et al 2006) and planning technology (e.g., Chien et al, 2003, and Bedrax-Weiss et al 2005). One of those technologies is the development of a procedure representation language (PRL) that both captures the form of traditional procedures and allows for automatic translation into code that can be executed by NASA-developed autonomous executives. 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. As well, PRL is being developed with a graphical authoring system, known as PRIDE, that enables non-computer specialists to write automated procedures (Kortenkamp et al 2007).

Given a set of procedures cast in PRL, one of our current research goals has been to enhance the PRL language to include planning information related to each procedure, i.e., 1) Time for both task duration and for temporal constraints among procedures, b) Resources that are required, produced or consumed by a procedure, c) pre-conditions, post-conditions and other constraints for both a given procedure and among concurrently executing procedures, and d) The decomposition of large procedures into the fundamental actions used to build up a mission plan. Our target flight disciplines have been Extravehicular Activity (EVA) and Power, Heat and Light Control (PHALCON). The two disciplines often work together because spacewalks entail the installation or removal of power equipment around the International Space Station (ISS). (Bonasso & Boddy 2009) details the results of our first year efforts in both "chunking" large EVA and PHALCON procedures into primitives for planning as well as developing PRL representations for time, resources, preconditions and effects that can easily translate into standard planning languages, our target being ANML (Smith & Cushing 2008).

A second major research goal is to design an interaction scheme as an addition to PRIDE that will elicit these planning data from the EVA and PHALCON flight controllers, the same experts who developed the PRL procedures. These subject matter experts have little or no understanding of automated planning technology. This paper describes our initial approach to obtaining from these experts planning information sufficient to be used by automated planners.

Goal versus Action Decomposition

Much of the activity planning done by PHALCONs and virtually all done by EVA flight controllers lends itself to Hierarchical Task Net (HTN) planning. Standard HTN decomposes a task into actions, but some planners, e.g., SIPE (Wilkins & Myers 1998) and AP (Applegate et al 1990), a planner we've used for several NASA applications, use goal decomposition. To illustrate, consider an EVA task to retrieve an external light known as a Crew and Equipment Translation Aid light, or CETA-light. A stripped down action description would be:

Define-action: retrieve-light
  Parameters: ev – crew, light – CETA-light
  Variables: bag – ORU-bag, light-loc – location
  Expansion:
      Sequence
       Pick-up (ev, bag)
       Translate-by-handrail (ev, light-loc)
       Extract-item-to-bag (ev, light, bag)

Basically, the crewmember gets the orbital replacement unit (ORU) bag, travels to the light location and unbolts and stores the light in the bag.

A plan using the above definition will always have three sub-actions. So the first action will still be planned even if the conditions of the initial situation include the fact that the crewmember already possesses the bag.

A goal-decomposition of the above expansion might be:

Expansion:
   Sequence
    Possessed-by (bag, ev)
    Located (ev, light-loc)
    Extracted-into-bag (ev, light, bag)

This form asks the planner to find actions that will bring about the goals (states) in the order specified. However, if any goal already holds, no action need be planned. So if the crewmember already possesses the bag at the outset, only actions for locating the crewmember and getting the light in the bag will appear in the plan. Additionally, a goal decomposition does not specify what action to take to bring about a desired goal, so, in the above example, any action that will position the crew member at the light location can be used, like traveling on a CETA cart or on the space station robotic arm. In essence, an expansion of goals is a template for many action decompositions. In practice there are always actions whose goal/state/intent is just that the action be successfully completed, which is the case for extract-item-to-bag.

While our design favors goal decomposition, our approach to building an interactive aid to elicit planning information will produce a representation from which either or both action and goal decompositions can be derived.

The Interactive Paradigm

We now describe a query-response flow of interaction in PRIDE to obtain the planning information needed to construct complete action descriptions, including actions with decompositions. We assume in this design that all the executable level actions – called procedures – have been defined in files with PRL representations. We also assume a domain ontology is available to the PRIDE system. Obtaining those is a non-trivial effort – we spent a year constructing these for our domain. The examples below are taken from our models of the EVA domain.

Goal Representation

First we analyzed the primitive actions/procedures to develop a set of domain relations that can serve as goals or actions (this set will need to be expanded as users determine there are other relations that should be modeled). Here are examples of a goal and an action relation:

Relation:
   Name: located
   Type: fluent
   Function?: yes
   Args:
     object – thing
     location – geographicarea
  Verb-form: "locate object at location"
  Prefix-form: "object is at location"

Relation:
   Name: extract-item-to-bag
   Type: action
   Function?: no
   Args:
     crew - agent
     object - station-object
     bag - oru-bag
  Verb-form: "crew extract object to bag"
  Prefix-form: "crew has extracted object to bag"

The type field is used to distinguish goals that can have a fluent form and those that are purely actions, that is, there is no corresponding state condition that could be used as a goal. The function field allows planners to take advantage of single-value fluents. For example, rather than

Variables: loc1 – location, crew1 crew 2 - agent
Conditions: located(crew1, loc1)
            located(crew2, loc1)

one can write:

Variables: crew1 crew 2 - agent
Conditions: located(crew1) = located(crew2)

The prefix forms are used to display the relation to the user as either effects or conditions; the verb form, as actions in a decomposition (see Building Decompositions below).

Obtain the Action

A subset of the relations described above corresponds to the PRL procedures assumed to exist for this endeavor. PRIDE will derive the action name as well as the intent of the action from this set. For this example we'll be using the procedures known as `pick-up`, `travel-by-handrail`, `travel-by-SSRMS`, and `extract-item-to-bag`.

So PRIDE will first direct the user to select the procedure to which planning information is to be added. Our user selects `extract-item-to-bag` and the template shown in Figure 1 appears.

Figure 1 Action Template

This is a two-step procedure, written in PRL, wherein the crewmember unbolts the item with a power grip tool and stows it in an ORU bag.

Obtain the Intent

Next PRIDE asks the user: *Select a goal that is the intent of this action?* The user selects from the list of relations described in the section on goal representation above and presented to the user in their prefix form. Our user selects "crew has extracted object to bag" and the template updates as in Figure 2:

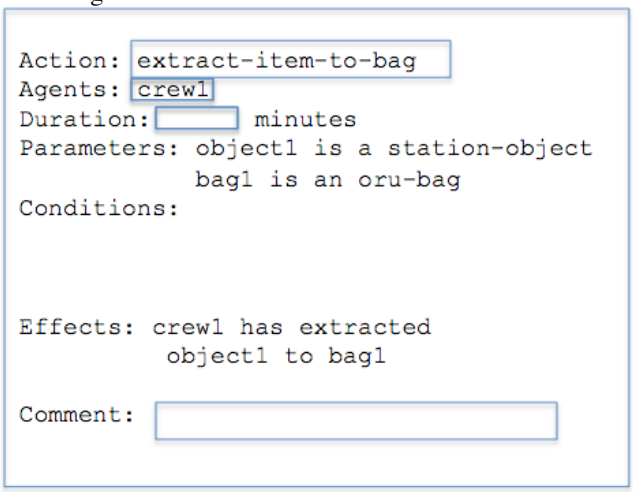

Figure 2 Action Template with Intent

When the relation is selected, PRIDE uses the type information for the relation's arguments to fill in the Agents and Parameters fields. Instances of parameters are constructed from the argument names. The agents are called out separately from the parameters so that other non-planning applications can use that information from the final result.

Determine Needed Tools

Another source of parameters will involve tools used in the procedure. So the user is now asked: *Are any tools needed for this procedure?* A taxonomy of the tools in the EVA domain are presented to the user as shown in Figure

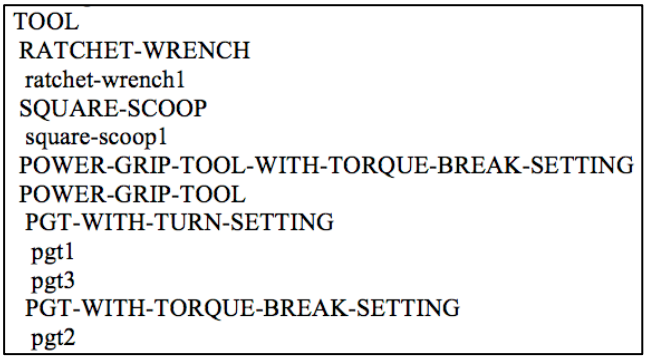

Figure 3 Tool Taxonomy (with instances in lower case)

3. The user selects a power grip tool with a precision ratchet, which shows up as pgt1 in the parameters

Obtain the Decomposition

If this were a new action, the user would be asked at this point to define the decomposition. Since the current action is a primitive, PRIDE will not query for a decomposition (but see Building Decompositions below).

Obtain the Preconditions

Rather than asking the user an open-ended question like,

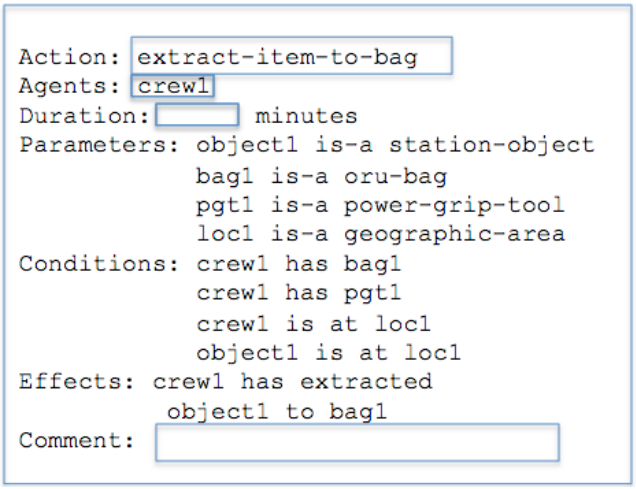

Figure 4 Action Template with Tools and Conditions

What conditions must be true for this procedure to be applicable?, we use a series of "wizard" questions keyed on common conditions such as location, possession and containment. For possession, PRIDE assumes the crew

will be the default possessor and so asks, *Should crew1 possess any items?*, and the user selects from a pop-up menu of the parameters. In this case, user selects bag1 and the pgt1. PRIDE then uses the relational form of possessed-by to construct the appropriate preconditions.

Similarly, PRIDE will ask if any of the crew and/or parameters need to be co-located. In this case, crew1 needs to be at the same place as object1. The same process is used for containment, but we will illustrate that in the effects query below. The resulting template is shown in Figure 4.

As this is a work in progress, we are for now assuming

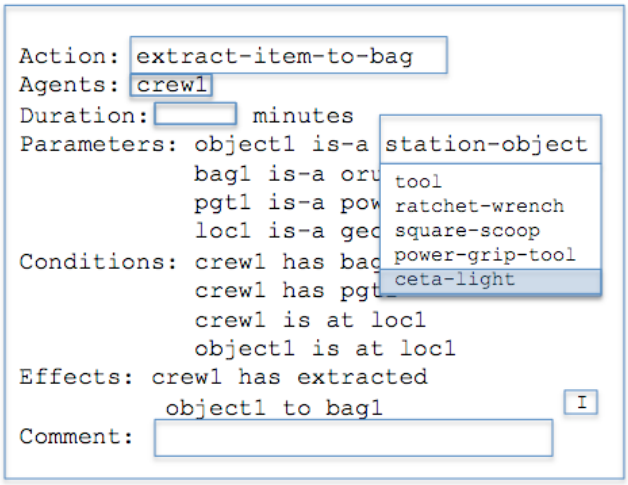

Figure 5 Focusing parameters with a type pull-down menu

all conditions are pre-conditions until we work out how to elicit temporal information from the user or develop some reasonable intelligent "wizard" questions to obtain it.

Focus the Parameters

PRIDE then tells the user, *Use the type drop-down menus to adjust the type of any parameter to be more specific*. A drop-down list under each parameter's type in the template contains all the subtypes for that parameter. The user activates the drop-down for station-object and selects the subtype CETA-light, as in Figure 5.

Obtain Side Effects

Again, rather than asking the user an open-ended question like, *What other effects will be true at the end of this procedure?*, we'll again use a the wizard approach and ask a series of questions keyed on common conditions such as location, possession and containment.

In this example, PRIDE uses the fact that there is a container and an object to ask the question, *At the conclusion of this action will bag1 contain an item?*, and gives a list of parameters less any containers. The user knows that the extracted item will be put in the bag so she checks object1. PRIDE uses the containment relation and the selected parameters to construct the effect as in Figure 6.

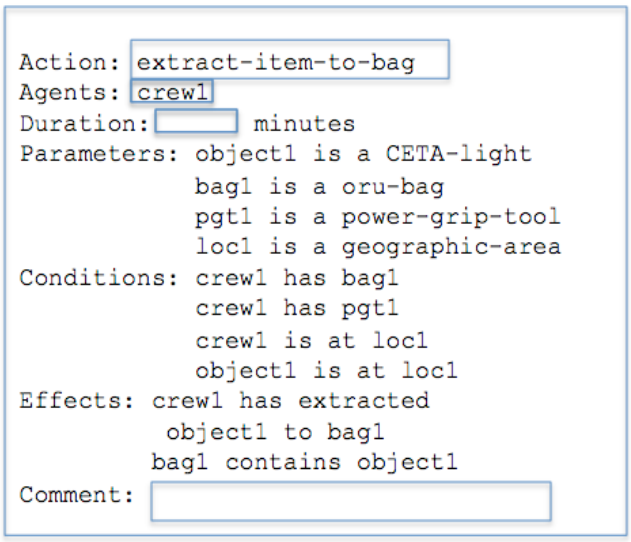

Figure 6 Action Template with Side Effect

Establish Duration

The user is now asked: *How long in minutes will this procedure take?* The user can specify an integer amount of minutes, in this case, 12, or he can specify a computation (see the translate-by-handrail action below).

Provide a Text Description

Finally, the user will be asked to provide a description of

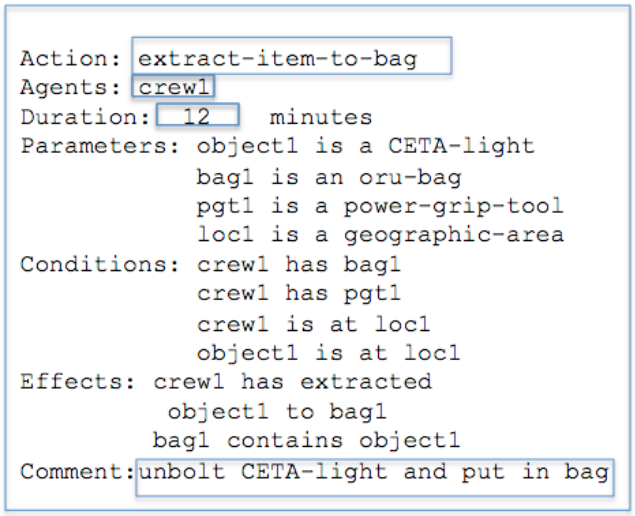

Figure 7 Completed Action Template

the action in free-form English text. Our user enters, "Unbolt the CETA light and place in bag." The final action is shown in Figure 7.

Internal Representation

The main objective of this interactive exercise is to construct an internal representation that can be translated into standard planning languages, such as PDDL and

ANML. Our proposed representation, the instance resulting from the template above, is show below.

```
Action: extract-item-to-bag
Agents: crew1
Duration: 12
Parameters: crew1 – agent
            object1 - CETA-light
            bag1 - ORU-bag
Variables: loc1 - geographicarea
           pgt1 - power-grip-tool
Preconditions: operator: "=="
                var: loc1
                relation: located
                  args: object1
                relation: located
                  args: crew1
               operator: predicate
                relation: possessed-by
                  args: bag1, crew1
               operator: predicate
                 relation: possessed-by
                  args: pgt1, crew1
Effects: relation: extract-item-to-bag
          args: crew1, object1, bag1
         relation: contained-by
          args: object1, bag1
Comment: "unbolt ceta-light and put in bag"
```

Note that any parameters not in the effects are moved to the variables slot. Also, the operator slot allows the definition of functional fluents. This internal representation, along with the set of relations defined earlier, holds sufficient information to generate the following PDDL and ANML actions.

```
(define-durative-action extract-item-to-bag
  :parameters (?crew1 – crew
               ?object1 – ceta-light
               ?bag1 – oru-bag)
  :vars (?loc1 – geographicarea
         ?pgt1 – power-grip-tool)
  :duration 12.0
  :condition
  (and
   (at start (located ?crew1 ?loc1))
   (at start (located ?object1 ?loc1))
   (at start (possessed-by ?bag1 ?crew1))
   (at start (possessed-by ?pgt1 ?crew1)))
  :effect
   (and
    (at end
     (extract-item-to-bag
      ?crew1 ?object1 ?bag1))
    (at end (contained-by ?object1 ?bag1)))
  :comment "unbolt ceta-light and put in
bag"
```

PDDL 2.1 can't take advantage of functional fluents. As well, our current planner, AP, uses only goal decomposition, so a goal form of the action, constructed by the action name and parameters is included in the effects.

```
action Extract_item_to_bag
       (agent crew1, CETA_light object1,
         ORU_bag bag1)
{
 duration := 12
 [start]{located(object1) == located(crew1);
         possessed_by(bag1) == crew1;
         exists (power_grip_tool pgt1) {
                  possessed_by(pgt1) ==
                  crew1}
  };
  [end] contained_by(object1) := bag1
}
```

ANML on the other hand, takes full advantage of functional predicates and can use both goal and action decompositions (e.g., see Building Decompositions below).

Add More Actions

We continue the example by building three more primitives, but we show only the final results (parameters are in italics). The next action is pick-up, whose intent is based on the possessed-by relation.

```
Action: pick-up
Agents: crew1
Duration: 5
Parameters: object1 is a station object,
            loc1 is a geographicarea
Conditions: crew1 is at loc1
            object1 is at loc1
Effects: crew1 has object1
Comment: "Crew untethers item and attaches
to suit."
```

Next we develop a translation action based on the located relation.

```
Action: travel-by-handrail
Duration: function: distance, path1
Agents: crew1
Parameters: loc1 is a geographicarea
            path1 is a path
            loc2 is a geographicarea
Conditions:
  the start location of path1 is loc2
  the end location of path1 is loc1
  crew1 is at loc2
Effects: crew1 is at loc1
Comment: "Crew uses handrails to go to
loc1."
```

Here the duration is computed from the path using a distance function. A list of such available domain functions will reside in the PRIDE system.

Next we develop a similar action based on the crewmember being mounted on the space station remote manipulator system (SSRMS).

```
Action: travel-by-SSRMS
Agents: crew1
Duration: function: GCA, loc2, loc1
Parameters: loc1 is a geographicarea
            arm1 is a robotic-arm
            loc2 is a geographicarea
Conditions: arm1 is located at loc2
            crew1 is on arm1
Effects: crew1 is at loc1
Comment: "Crew GCAs arm to loc1"
```

Here the duration is computed using the Ground Controlled Approach (GCA) function with the start and end locations as arguments.

Building Decompositions

There is no existing PRL procedure for a complex action; by definition they are composed of an ordered set of other complex actions or primitives. Our user wishes to build a retrieve action wherein a crewmember obtains a bag, travels to a worksite and extracts a CETA-light to the bag. She will go through the steps as before, with certain differences alluded to earlier because this is a new action with a decomposition.

Action Name and Intent. The user will create an action name for the new action and PRIDE will generate a relation based on an assumption that at least one crewmember is involved and at least one object. If there is no object involved in the preconditions, effects or decomposition, PRIDE will excise it from the final internal representation. In this case the user types in "retrieve item" and PRIDE generates

Relation:
  Name: retrieve-item
  Type: action
  Function?: no
  Args:
    crew - agent
    object - station-object
 Verb-form: "crew retrieve item object"
 Prefix-form: ""

and the action template that appears is:

```
Action: retrieve-item
Agents: crew1
Parameters: object1 is a station-object
Effects: crew1 retrieve item object1
```

Decompositions. As mentioned earlier, a new planning action may include a decomposition, so PRIDE asks, *Does this action have a decomposition?* In this case, the user answers in the affirmative and PRIDE presents a list of verb forms for existing relations. A subset of that list is shown below:

1) "locate object at geographicarea "
2) "object possess another object"
3) "object contain another object"
4) "crew extract CETA-light to bag"

The user selects 2) and 1), focusing object to crew, and 4), which results in the following template:

```
Action: retrieve-item
Agents: crew1
Duration: derived
Parameters: object1 is a CETA-light
            loc1 is a geographicarea
            bag1 is an ORU-bag
Expansion: sequential
            crew1 possess bag1
            locate crew1 loc1
            crew1 extract object1 to bag1
Effects: crew1 retrieve item object1
```

The default ordering is sequential, but is associated with a pull-down menu that includes `unordered`, `simultaneous` and `parallel`.

Note that the parameter object1 has been further specified by the addition of the extract action where the object type was specified as a CETA-light. As well, parameters from the actions other than object1 and crew1 are added to the variables list. Finally, the duration is set to `derived`, since it will be an accumulation of the durations of the actions in the decomposition.

Tools. In this first pass at our design we do not query for tools in a complex actions; the bottom-up approach to building actions should cover the needed tools at the primitive level.

Preconditions and Effects. In this first pass at our design we do not allow side effects for a decomposition; the bottom-up approach to building actions should cover the needed effects at the primitive levels.

For tasks with decompositions, the usual suspects for preconditions – e.g., `crew1 has bag1`, are brought about by the actions in the decomposition. But PRIDE can reason about some aspects of this action and ask, *Is loc1 the location of object1 or bag1?* The user selects object1.

After adding a text description the action is:

```
Action: retrieve-item
Agents: crew1
Duration: derived
Parameters: object1 is a CETA-light
            loc1 is a geographicarea
            bag1 is an ORU-bag
```

```
Conditions: object1 is at loc1
Expansion: sequential
             crew1 possess bag1
             locate crew1 loc1
             crew1 extract object1 to bag1
Effects: crew1 retrieve item object1
Comment: "crew gets bag, goes to loc and
extracts light."
```

Internal Representation. The internal representation for the above complex action is:

```
Action: retrieve-item
Agents: crew1
Duration: derived
Parameters: crew1 - agent
            object1 - ceta-light
Variables:  bag1 - oru-bag
            loc1 - geographicarea
Conditions: operator "=="
               var: loc1
               relation: located
                 args: object1
Expansion:
  Order: sequential
    Tasks:
      relation: possessed-by
        args: bag1, crew1
      relation: located
        args: crew1, loc1
      relation: extract-item-to-bag
        args: crew1, object1, bag1
Effects:  relation: "retrieve-item"
          args: crew1, object1
Comment: "Crew gets bag and goes to loc and
extracts light."
```

Here are the resulting PDDL and ANML actions.

```
action Retrieve_item (agent crew1,
                      ceta_light object1,
                      oru_bag bag1)
[duration]
{location current_location := located(bag1);

[all]contains
  ordered(ach_possessed_by(bag1,crew1),
       ach_located(crew1,current_location),
      extract_item_to_bag(crew1,object1,bag1
    ))
}
```

The first two items in the expansion are goals. ANML uses an achieve action for each goal, e.g.,

```
Action ach_possessed_by(station_object item,
crew agent)[duration]
{
[start] possessed_by(item, agent) == TRUE ||
{[start] possessed_by(item, agent) == FALSE;
  [end] possessed_by(item, agent) == TRUE}}
}
```

that can be interpreted as: if the state doesn't hold at the start, find an action that will bring it about.

PDDL uses the goal form for all the actions in the decomposition:

```
(define-durative-action retrieve-item
  :parameters (?crew1 - crew
               ?object1 - ceta-light))
  :vars (?bag1 - oru-bag
         ?loc1 - geographicarea)
  :condition (at start (located ?object1
                        ?loc1))
  :expansion
    (sequential
      (possessed-by ?bag1 ?crew1)
      (located ?crew1 ?loc1)
      (extract-item-to-bag ?crew1 ?object1
         ?bag1)
:effect (at end (retrieve-item ?crew1
         ?object1))
:comment "crew gets bag, goes to loc and
extracts light."
```

Resulting Plans

For planning, the PDDL or ANML actions are selected as tasks to be planned. So the user could, for example, ask the planner to plan `bob retrieve ceta-light1` and a resulting plan might be:

```
sequence
 bob pick-up medium-oru-bag2
 bob travel-by-handrail to ceta-light1-loc
 bob extract ceta-light1 to medium-oru-bag2
```

Given an initial situation where Bob already possessed an ORU-bag, however, the plan would be:

```
sequence
 bob travel-by-handrail to ceta-light1-loc
 bob extract ceta-light1 to medium-oru-bag1
```

Given a starting situation where Bob possessed an ORU bag and was positioned on the SSRMS, the plan would be:

```
sequence
 bob travel-by-SSRMS to ceta-light1-loc
 bob extract ceta-light1 to medium-oru-bag1
```

Thus, the decompositions serve as templates of several different action combinations that could bring about a top-level goal.

Relation to Other Work

The bulk of the efforts in knowledge engineering for planning deal with AI programmers eliciting planning

information from domain experts, and then using KE aids to model and validate this information. Examples are (Frenandez et al 2004) and (Simpson 2007). The effort in this paper is aimed at developing planning actions from an existing set of executable procedures, by asking the procedure authors – non-AI-programmers – leading questions about the procedures. Our hope is that, through a set of focused questions to these non-AI users, we can obtain planning actions that can be used to generate valid, though possibly inefficient plans.

Like our work here, related KE efforts target standard planning languages like PDDL, NDDL and OCL. Besides PDDL, we have selected the ANML planning language, because it is based on strong notions of action and state, uses a variable/value model, supports rich temporal constraints (Smith & Cushing 2008 mention ongoing development of an ANML to NDDL translator), and provides simple, convenient idioms for expressing the most common forms of action conditions, effects, and resource usage. The language supports both generative and HTN planning models in a uniform framework and has a clear, well-defined semantics.

(Boddy and Bonasso 2010) is a companion paper that discusses the semantics of ANML including goal versus action decompositions.

Summary and Future Work

With the interactive paradigm described in this paper, we believe we can enable non-AI programmers to construct primitive and complex actions from existing procedures that can be used by AI planners to generate executable plans. Our design favors goal-based HTNs as templates for multiple methods of bringing about top-level goals. We are in the process of coding an interactive plug-in to our PRIDE system that will execute this paradigm.

What we've reported on here is of course the tip of the iceberg. Our approach using "wizard" questions will quickly become too restrictive as our domain models become more complicated. So, allowing the user to break out of the restricted question set and manage the action templates directly will involve the development of many more checks and balances to help the user avoid inadvertent errors.

And as with all KE for planning efforts, the planning models developed by this interactive paradigm must be validated. We envision running our AP planner endowed with the PRIDE-developed action set, on a set of situations to obtain valid plans. We then need to develop schemes for failure diagnosis and feeding back the results of that diagnosis to the authoring system when the planner cannot find valid plans. Initially, we will construct a single thread of that closed loop, concentrating on one or two types of authoring errors, using our existing planning aid (Bonasso et al 2009).

References

Applegate, C., C. Elsaesser, and J. Sanborn. 1990. An Architecture for Adversarial Planning. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(1): p. 186-194.

Bedrax-Weiss, T., et al., 2005. *EUROPA2: user and contributor guide*, NASA Ames Research Center.

Bonasso, Pete and Boddy, Mark. 2010. Planning for Human Execution of Procedures Using ANML. In submission to ICAPS 2010 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS). Toronto, Canada.

Bonasso, Pete, Boddy, Mark, Kortenkamp, D. 2009. Enhancing NASA's Procedure Representation Language to Support Planning Operations. In Proceedings of IWPSS09, Pasadena, CA.

Chien, S., et al. Autonomous Science on the Earth Observer One Mission. In i-SAIRAS 2003. 2003. Nara, Japan.

Frank, Jeremy. 2009. Automaton for Operations. http://ti.arc.nasa.gov/news/a4o-demo-for-hdu/

Simpson, R. M. 2007. Structural Domain Definition using GIPO IV. *The Knowledge Engineering Review*, 22, 117-134. Cambridge University Press

Susana Fern´andez, Daniel Borrajo, Raquel Fuentetaja, Juan D. Arias and Manuela Veloso. 2004. PLTOOL: A Knowledge Engineering Tool for Planning and Learning. *The Knowledge Engineering Review*, Vol. 00:0, 1–24. Cambridge University Press

Kortenkamp, D., R.P. Bonasso, and D. Schreckenghost. 2007. Developing and Executing Goal-Based, Adjustably Autonomous Procedures,. in AIAA InfoTech@Aerospace Conference.

Smith, D.E. and W. Cushing. 2008. The ANML Language, in iSAIRAS. Los Angeles, CA.

V. Verma, A. Jónsson, C. Pasareanu, and M. Iatauro, Universal Executive and PLEXIL: Engine and Language for Robust Spacecraft Control and Operations, American Institute of Aeronautics and Astronautics Space 2006 Conference.

Wilkins, D. and Myers, K. 1998. A Multi-agent Planning Architecture, in Artificial Intelligence Planning Systems, Pittsburg, PA.