A Detailed Example of a SPAR-Based Plan Representation

Stephen T. Polyak

Artificial Intelligence Applications Institute (AIAI)
Institute for Representation and Reasoning (IRR)
Division of Informatics, The University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, United Kingdom
Steve_Polyak@ed.ac.uk

Introduction

The Shared Planning and Activity Representation (SPAR)[1] is being developed to contribute to a range of purposes including domain modelling, plan generation, plan analysis, plan case capture, plan communication, behaviour modelling, etc. The purpose of this paper is to illustrate one example of a SPAR-based plan and activity representation. The example is intended to provide a common point of reference for SPAR discussions based on a simple blocks world domain. Many other examples may be developed to illustrate other domains which could require different modelling approaches. This paper uses a specialised sorted first-order language (The Common Process Language, CPL) to present a SPAR-compliant representation of a plan for stacking blocks in various stages of a planning process. This planning process and domain is described along with a set of extensions required to annotate the plan artifact with planning rationale.

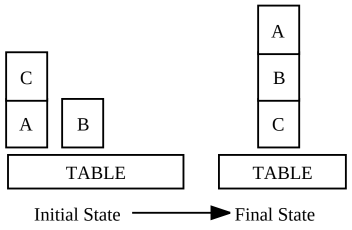
Blocks World Domain

The first step is to outline the world or domain for which this block-stacking plan applies. This paper draws on a simple domain which is described in the UM Nonlin users manual. The domain (which we will abbreviate as BBW, for basic blocks world) is essentially composed of two higher level operator schemas (i.e. makeon and makeclear) and a primitive action schema (i.e. puton) as shown in the UM Nonlin operator macros listed below. The main activity relatable objects are blocks along with a table which is specified to always be available as the destination of the primitive puton action.

```
(opschema makeon
:todo (on ?x ?y)
:expansion (
(step1 :goal (cleartop ?x))
(step2 :goal (cleartop ?y))
(step3 :action (puton ?x ?y))
)
:orderings ( (step1 -> step3)
( step2 -> step3))
:variables (?x ?y))
—–
(opschema makeclear
:todo (cleartop ?x)
:expansion (
(step1 :goal (cleartop ?y))
(step2 :action (puton ?y ?z))
)
:orderings ((step1 -> step2))
:conditions (
(:use-when (on ?y ?x) :at step2)
(:use-when (cleartop ?z) :at step2)
(:use-when (not (equal ?z ?y)) :at step1)
(:use-when (not (equal ?x ?z)) :at step1)
)
:variables (?x ?y ?z))
—–
(actschema puton
:todo (puton ?x ?y)
:expansion (
(step1 :primitive
(puton-action ?x ?y)))
:conditions (
(:use-when (cleartop ?x) :at step1)
(:use-when (cleartop ?y) :at step1)
(:use-only-for-query (on ?x ?z) :at step1)
)
:effects (
(step1 :assert (on ?x ?y))
(step1 :assert (cleartop ?z))
(step1 :delete (cleartop ?y))
(step1 :delete (on ?x ?z))
)
:variables (?x ?y ?z)
:duration 1)
```

[1] Details on the Shared Planning and Activity Representation (SPAR) are available via the WWW at: http://www.aiai.ed.ac.uk/∼arpi/spar/

The objectives which we will address here are based on Sussman's anomaly problem. In brief, we have a specification of an initial world state in which blocks A and B are on the table and block C is on A and we wish to achieve a world state in which A is on B and B is on C. This is depicted in Figure 1 using a UM Nonlin problem description.



(defun sussman-blocks ()
(store-always-ctxt '((cleartop table)))
(store-init-ctxt '((on c a) (on a table) (on b table) (cleartop c) (cleartop b)))
(plan sussman-blocks
(g1 :goal (on a b))
(g2 :goal (on b c))
))

Figure 1: A Specification of the Sussman Anomaly Problem.

Planning Process

In a very simple model of a planning process, we can envision an initial task formulation step which is followed by an iteration of plan refinements which continue until all plan flaws have been addressed. In this paper, we are interested in describing the SPAR-based representation of the plan at both the end of the initial task formulation step and at the end of a sample plan refinement step. In this case, we will consider a plan refinement step to be equivalent to one planning "cycle" in the UM Nonlin planner.

We will use the Common Process Language (CPL) as a concrete presentation of the SPAR-based representations. An overview of the CPL language is available from the author.

Task Assignment

The block stacking plan begins its life with some specification of the objectives which apply to it. For example, the following set of constructs which contain the plan objectives which may be communicated from a task assigning agent to an automated planner (UM Nonlin in this case).

```
//Specifying the sorts
SORT spar-plan={P1}
SORT spar-timepoint={TP1,TP2}
SORT spar-objective-specification={OS1}
SORT spar-always-constraint={AC1}
SORT spar-output-constraint={OUT1-OUT5}
SORT spar-input-constraint={INP1-INP2}
SORT spar-objective={Obj1,Obj2}
SORT spar-include-objective-constraint={IOC1,IOC2}
SORT bbw-block={Ba,Bb,Bc}
SORT bbw-table={T1}

//Designating the plan's temporal scope
process.start-timepoint(P1)=TP1
process.finish-timepoint(P1)=TP2

//Assigning the objective spec to plan, P1
process.objective-spec(P1)=OS1

//Adding the always true constraint
constraint.expression(AC1)="(cleartop T1)"
member(AC1,OS1)

//Adding the objectives
objective.pattern(Obj1)="goal (on Ba Bb) at TP2"
include-objective(IOC1)=Obj1
member(IOC1,OS1)

objective.pattern(Obj2)="goal (on Bb Bc) at TP2"
include-objective(IOC2)=Obj2
member(IOC2,OS1)

//Adding the goal world state specifications
constraint.expression(INP1)="(on Ba Bb) at TP2"
constraint.expression(INP2)="(on Bb Bc) at TP2"
member(INP1,OS2)
member(INP2,OS2)

//Adding the initial world state specifications
constraint.expression(OUT1)="(on Bc Ba) at TP1"
constraint.expression(OUT2)="(on Ba T1) at TP1"
constraint.expression(OUT3)="(on Bb T1) at TP1"
constraint.expression(OUT4)="(cleartop Bc) at TP1"
constraint.expression(OUT5)="(cleartop Bb) at TP1"
member(OUT1,OS2)
member(OUT2,OS2)
member(OUT3,OS2)
member(OUT4,OS2)
member(OUT5,OS2)
```

Plan Preparation

In SPAR, a plan relates an objective-specification to an activity-specification. What we have in our plan description so far, is simply the objective-specification. Before we begin planning, we will perform a plan preparation step based on the objectives. In this step we will create the counter part activity-specification and translate the objectives into issues to be handled. The plan representation, prior to plan refinement is represented as follows.

```
//Specifying the sorts
SORT spar-plan={P1}
SORT spar-timepoint={TP1,TP2}
SORT spar-start={S1}
SORT spar-finish={F1}
SORT spar-objective-specification={OS1}
SORT spar-activity-specification={AS1}
SORT spar-always-constraint={AC1}
SORT spar-output-constraint={OUT1-OUT5}
SORT spar-input-constraint={INP1-INP2}
SORT spar-objective={Obj1,Obj2}
SORT spar-include-objective-constraint={IOC1,IOC2}
SORT spar-include-node-constraint={INC1,INC2}
SORT spar-issue-constraint={IS1,IS2}
SORT spar-ordering-constraint={OC1}
SORT bbw-block={Ba,Bb,Bc}
SORT bbw-table={T1}

//Designating the plan's temporal scope
process.start-timepoint(P1)=TP1
process.finish-timepoint(P1)=TP2

//Assigning the objective spec to plan, P1
process.objective-spec(P1)=OS1

//Assigning the activity spec to plan, P1
process.activity-spec(P1)=AS1

//Adding the bounding nodes
include-node(INC1)=S1
start.timepoint(S1)=TP1
include-node(INC2)=F1
finish.timepoint(F1)=TP2
member(INC1,AS1)
member(INC2,AS1)
constraint.expression(OC1)="before(TP1,TP2)"
member(OC1,AS1)

//Adding the always true constraint
constraint.expression(AC1)="(cleartop T1)"
member(AC1,OS1)

//Adding the objectives
objective.pattern(Obj1)="goal (on Ba Bb) at TP2"
include-objective(IOC1)=Obj1
member(IOC1,OS1)

objective.pattern(Obj2)="goal (on Bb Bc) at TP2"
include-objective(IOC2)=Obj2
member(IOC2,OS1)

//Adding the goal world state specifications
constraint.expression(INP1)="(on Ba Bb) at TP2"
constraint.expression(INP2)="(on Bb Bc) at TP2"
member(INP1,OS2)
member(INP2,OS2)

//Adding the initial world state specifications
constraint.expression(OUT1)="(on Bc Ba) at TP1"
constraint.expression(OUT2)="(on Ba T1) at TP1"
constraint.expression(OUT3)="(on Bb T1) at TP1"
constraint.expression(OUT4)="(cleartop Bc) at TP1"
constraint.expression(OUT5)="(cleartop Bb) at TP1"
member(OUT1,OS1)
member(OUT2,OS1)
member(OUT3,OS1)
member(OUT4,OS1)
member(OUT5,OS1)

//Listing the current issues
constraint.expression(IS1)="goal (on Ba Bb) at TP2"
constraint.expression(IS2)="goal (on Bb Bc) at TP2"
member(IS1,AS1)
member(IS2,AS1)
```

End of Plan Cycle 1

Planning begins based on the initial representation described above. We do not assert that this is the actual internal representation used by UM Nonlin, but rather that the following CPL example is an externalisation of the planning knowledge following the end of cycle 1. For example, this knowledge could be automatically generated by UM Nonlin at the end of each cycle by translating its internal data structures to the descriptions shown below.

In the first cycle, the issue or flaw, IS1, is selected to be addressed. Nonlin selects this issue based on a simple linear selection criteria (i.e. it considers the issues to be in a list data structure rather than the more general set of CPL). The selected mode of refinement involves adding a new node which represents a "makeon" action. It is expanded by a "makeon" process (which corresponds to the makeon opschema). This causes issue IS1 to be deleted and three new issues to be added. Two of the issues correspond to new world state requirements on a new "puton" action in the "makeon" process and the third is the pending expansion of the non-primitive "puton" action.

We will present the externalisation of this new plan artifact and also present the decision rationale which accompanies it. The elements of the decision rationale are an extension to the core SPAR concepts and are described in the CPL paper.

```
//Specifying the sorts
SORT spar-plan={P1}
SORT spar-timepoint={TP1-T6}
SORT spar-start={S1}
SORT spar-finish={F1}
SORT spar-begin={Beg1}
SORT spar-end={End1}
SORT spar-action={A1-A2}
SORT spar-process={Proc1}
SORT spar-objective-specification={OS1}
SORT spar-activity-specification={AS1-AS2}
SORT spar-always-constraint={AC1}
SORT spar-output-constraint={OUT1-OUT5}
SORT spar-input-constraint={INP1-INP4}
SORT spar-objective={Obj1-Obj2}
SORT spar-include-objective-constraint={IOC1-IOC2}
SORT spar-include-node-constraint={INC1-INC6}
SORT spar-issue-constraint={IS2,IS3,IS4}
SORT spar-ordering-constraint={OC1-OC6}
SORT bbw-block={Ba,Bb,Bc}
SORT bbw-table={T1}

//Designating the plan's temporal scope
process.start-timepoint(P1)=TP1
process.finish-timepoint(P1)=TP2
```

```
//Assigning the objective spec to plan, P1
process.objective-spec(P1)=OS1

//Assigning the activity spec to plan, P1
process.activity-spec(P1)=AS1

//Adding the bounding nodes
include-node(INC1)=S1
start.timepoint(S1)=TP1
include-node(INC2)=F1
finish.timepoint(F1)=TP2
member(INC1,AS1)
member(INC2,AS1)
constraint.expression(OC1)="before(TP1,TP2)"
member(OC1,AS1)

//Adding the makeon action
include-node(INC3)=A1
member(INC3,AS1)
activity.pattern(A1)="makeon Ba Bb"
activity.expansion(A1)=Proc1
activity.begin-timepoint(A1)=T3
activity.end-timepoint(A1)=T4
constraint.expression(OC1)="before(TP1,TP3)"
member(OC2,AS1)
constraint.expression(OC1)="before(TP4,TP2)"
member(OC3,AS1)

//Adding the makeon expansion process
process.pattern(Proc1)="makeon ?x ?y"
process.expands(Proc1)=A1

//Designating the process temporal scope
process.start-timepoint(Proc1)=TP3
process.finish-timepoint(Proc1)=TP4

//Assigning the activity spec to process, Proc1
process.activity-spec(Proc1)=AS2

//Adding the bounding nodes
include-node(INC4)=Beg1
start.timepoint(Beg1)=TP3
include-node(INC5)=End1
finish.timepoint(End1)=TP4
member(INC4,AS2)
member(INC5,AS2)
constraint.expression(OC4)="before(TP3,TP4)"
member(OC4,AS2)

//Adding the puton action
include-node(INC6)=A2
member(INC6,AS2)
activity.pattern(A2)="puton Ba Bb"
activity.begin-timepoint(A2)=T5
activity.end-timepoint(A2)=T6
constraint.expression(OC5)="before(TP3,TP5)"
member(OC5,AS2)
constraint.expression(OC6)="before(TP6,TP4)"
member(OC6,AS2)

//Adding the puton world state requirements
constraint.expression(INP3)="(cleartop Ba) at TP5"
member(INP3,AS2)
constraint.expression(INP4)="(cleartop Bb) at TP5"
member(INP4,AS2)

//Adding the always true constraint
constraint.expression(AC1)="(cleartop T1)"
member(AC1,OS1)

//Adding the objectives
objective.pattern(Obj1)="goal (on Ba Bb) at TP2"
include-objective(IOC1)=Obj1
member(IOC1,OS1)

objective.pattern(Obj2)="goal (on Bb Bc) at TP2"
include-objective(IOC2)=Obj2
member(IOC2,OS1)

//Adding the goal world state specifications
constraint.expression(INP1)="(on Ba Bb) at TP2"
constraint.expression(INP2)="(on Bb Bc) at TP2"
member(INP1,OS2)
member(INP2,OS2)

//Adding the initial world state specifications
constraint.expression(OUT1)="(on Bc Ba) at TP1"
constraint.expression(OUT2)="(on Ba T1) at TP1"
constraint.expression(OUT3)="(on Bb T1) at TP1"
constraint.expression(OUT4)="(cleartop Bc) at TP1"
constraint.expression(OUT5)="(cleartop Bb) at TP1"
member(OUT1,OS1)
member(OUT2,OS1)
member(OUT3,OS1)
member(OUT4,OS1)
member(OUT5,OS1)

//Listing the current issues
constraint.expression(IS2)="goal (on Bb Bc) at TP2"
member(IS2,AS1)
constraint.expression(IS3)="goal (cleartop Ba) at TP5"
member(IS3,AS2)
constraint.expression(IS4)="goal (cleartop Bb) at TP5"
member(IS4,AS2)
constraint.expression(IS5)="expand A2"
member(IS5,AS2)
```

End of Plan Cycle 1 - Rationale

Along with the plan artifact knowledge produced above we can record and communicate the decision rationale which details the design process that was navigated by the planner during this plan cycle. In particular, we use a design space analysis ontology which indicates the questions, options, and criteria involved. This representation, which utilises a SPAR extension, describes the space of decisions in addition to the space of behavior described above.

```
//Specifying the sorts
SORT dr-rationale={DR1}
SORT dr-rationale-specification={RS1}
SORT dr-question={Q1-Q3}
SORT dr-option={Opt1-Opt7}
SORT dr-criteria={Crt1-Crt4}
SORT dr-include-constraint={RC1}
```

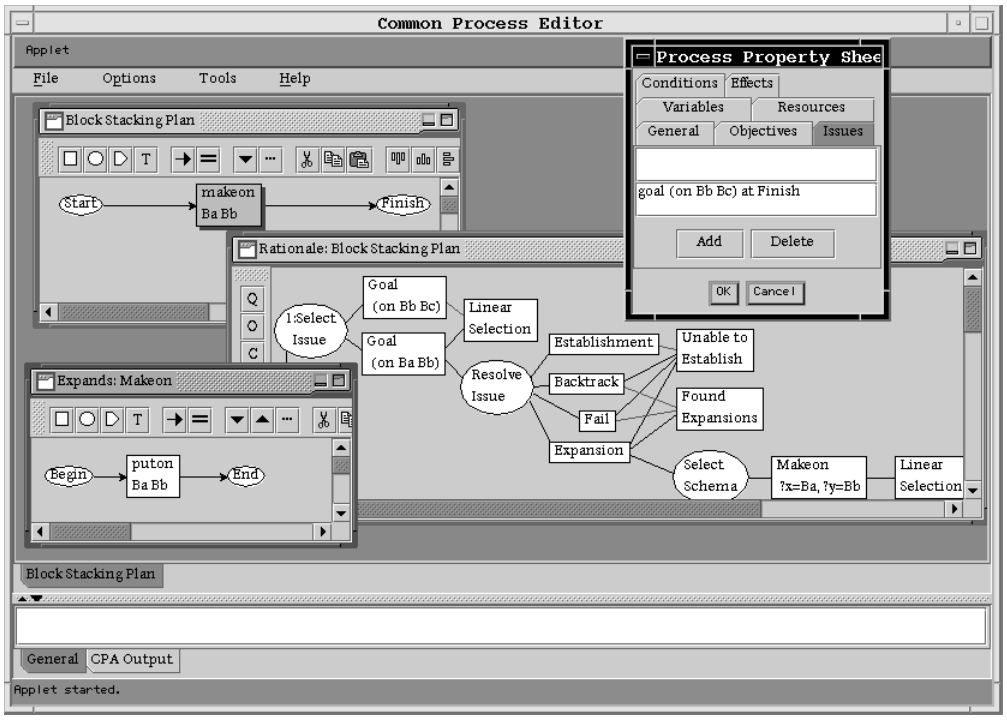


Figure 2: Common Process Editor Presentation

```
decision-rationale(P1)=DR1
rationale-spec(DR1)=RS1

rationale.label(Q1)="1:Select Issue"
has-option(Q1,Opt1)
has-option(Q1,Opt2)
rationale.label(Opt1)="goal (on Bb Bc)"
rationale.label(Opt2)="goal (on Ba Bb)"

selected(Opt2)
rationale.label(Crt1)="linear selection"
supports(Crt1,Opt2)
detracts(Crt,Opt1)

sub-question(Opt2,Q2)
rationale.label(Q2)="Resolve Issue"
has-option(Q2,Opt3)
has-option(Q2,Opt4)
has-option(Q2,Opt5)
has-option(Q2,Opt6)
rationale.label(Opt3)="Establishment"
rationale.label(Opt4)="Backtrack"
rationale.label(Opt5)="Fail"
rationale.label(Opt6)="Expansion"

selected(Opt6)
rationale.label(Crt2)="Unable to Establish"
rationale.label(Crt3)="Found Expansions"
supports(Crt2,Opt4)
supports(Crt2,Opt5)
supports(Crt2,Opt6)
supports(Crt3,Opt6)
detracts(Crt2,Opt3)
detracts(Crt3,Opt4)
detracts(Crt3,Opt5)

sub-question(Opt5,Q3)
rationale.label(Q3)="Select Schema"
has-option(Q3,Opt7)
rationale.label(Opt7)="makeon ?x=Ba, ?y=Bb"

selected(Opt7)
rationale.label(Crt4)="Linear Selection"
supports(Crt4,Opt7)

include-item(RC1)=Q1
member(RC1,RS1)
```

CPE Visualisation

While the CPL representation provides a rich representation of the plan and can be used to communicate knowledge between planning tools, it is certainly not appropriate as a human-inspectable artifact. In our work on a Common Process Framework, we developed tools which can import this CPL knowledge for process visualisation and process editing. Figure 2 represents the graphical presentation of the plan knowledge following the conclusion of plan cycle 1 as described above.