In-Situ Domain Modeling with Fact Routes *

Daniel Bryce
SIFT, LLC.
dbryce@sift.net

Pete Bonasso
Traclabs, Inc.
bonasso@traclabs.com

Khalid Adil
Traclabs, Inc.
khalid@traclabs.com

Scott Bell
Traclabs, Inc.
scott@traclabs.com

David Kortenkamp
Traclabs, Inc.
korten@traclabs.com

Abstract

Engineering plans and the domain models that underly them is a significant challenge. Research on knowledge engineering for planning has developed many ways to produce both plans and domain models, but most work treats these as separate tasks. We propose that it is more natural to combine plan synthesis with domain modeling. We describe a new planning and modeling tool, called Conductor, that is based upon representing plan steps and fact routes. Conductor uses a visualization metaphor derived from metro maps to display facts as transit routes and step preconditions as stations. The visualization helps quickly convey how a plan modifies the state and appeals to the metro metaphor to support user engagement in modeling.

Introduction

Developing plan authoring tools is a challenge. Providing support beyond the level of a text-editor requires some form of domain model that describes the semantics of steps. However, acquiring and maintaining the domain model often requires an expert. Users may not have access to such modeling experts, but typically have a suitable understanding of the task, the steps to achieve it, and the relevant state variables. Users need tools that can easily accept their knowledge and provide planning support based upon that knowledge.

We present Conductor, a visual planning tool that enables users to add, remove, and rearrange steps, as well as annotate the plan with their knowledge about the state of the world. We propose a new form of domain model knowledge called a fact route that specifies a fact's life cycle. Conductor uses the visual metaphor of a metro map (Figure 1) to treat state facts as transit routes, and how facts interact with steps as stations. Fact routes are conceptually simple and they provide useful, but incomplete, information about the domain model. For example, a fact route of the form $a_i \xrightarrow{p:(a_k,\ldots,a_l)} a_j$ states that the fact p is true between steps a_i and a_j, and is used by $a_k, \ldots, a_l$ as a precondition. The fact route reveals that a_i adds p, a_j deletes p, no step between a_i and a_j deletes p, and that p is a precondition of $a_k, \ldots, a_l$. The fact route fails to state whether any other step between a_i and a_j also adds p. It also allows the user to omit steps that use p as a precondition. In this way, a fact route is a bundle of causal links (but it is not clear which causal links). Explicitly stating the causal links or the precondition, adds, and deletes would be more informative, but at the cost of usability and the peril of user error.

Conductor uses the Marshal model maintenance system (Bryce, Benton, and Boldt 2016) to reason about the incompletely specified domain model. Marshal treats the fact routes as observations of the incomplete model, and develops possible interpretations of the model that are consistent with them. Conductor presents Marshal's interpretations so that the user can optionally dispel incompleteness and correct errors.

Modeling fact routes appeals to a user's intuition about how facts persist over time without necessarily requiring that they encode how the fact is related to each step. Presenting only the impactful model omissions and errors helps keep the user on task without requiring a complete and correct model. Conductor and Marshal help ensure that the plan is internally consistent given the information provided by the user.

Conductor is different than contemporary planning tools because it focusses on acquiring the aspects of the domain model that are most natural for users to express. Conductor does not require a complete and correct domain model, but is able to structure interactions with a user so that it can acquire one. In contrast with prior works that treat modeling and planning as distinct activities, Conductor takes a least-commitment approach to modeling that is more accessible to non-experts.

In the following, we discuss background on incomplete models, describe fact routes and how they inform the planning model, and present Conductor's interface and interaction modalities. We then explain how Marshal provides Conductor the interpretations of an incomplete model and how Conductor elicits model refinements. We end with a discussion of related work and a conclusion.

*This work was conducted under NASA contract NNX15CA19c.

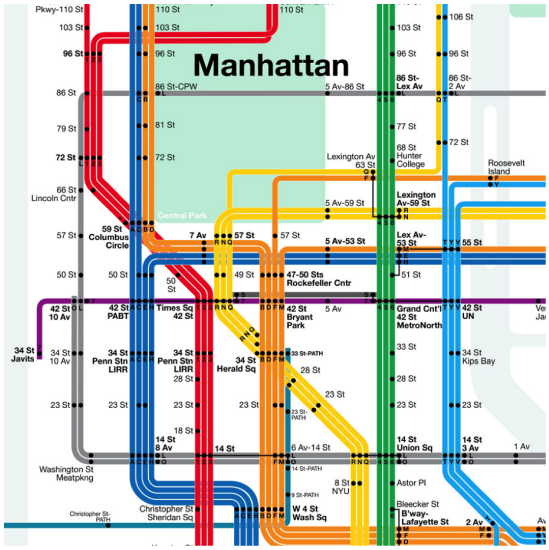

Figure 1: Manhattan Metro Map

Figure 2: Conductor Fact Routes

Example Plan

We illustrate Conductor with an example plan to brew a cup of coffee using the Aeropress^{TM}brewer (Figure 3). Figure 2 illustrates the plan in Conductor, where each step aside from the "Initial State" and "Goal" steps are provided with the instructions for the brewer. The figure also illustrates the fact routes added by a Conductor user. The plan involves placing the brewer on a cup, adding coffee and water, waiting for the coffee to brew, stirring the coffee, and then plunging to extract the coffee (as shown in Figure 3).

Figure 3: Aeropress Coffee Brewer

Background

As a user creates a plan and annotates it with fact routes in Conductor, Marshal is able to develop interpretations of the domain model and critiques of the plan. In the following, we define plans, domain models, fact routes, and open conditions.

Plans: A plan π is a sequence of actions $(a_1, \ldots, a_n)$. For convenience, we assume that the initial state and goal are represented by actions a_0 and a_{n+1}.

Domain Model: We represent the domain model with a grounded (propositional) STRIPS model M. The grounded STRIPS planning model M defines the tuple (P, A), where P is a set of state propositions (facts), and A is a set of actions. Each action $a \in A$ defines the tuple $(\mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a))$, where each element of the tuple is a subset of P. Marshal (and Conductor, by proxy) may never fully represent the domain model. Marshal maintains knowledge about the model, and each interpretation of this knowledge corresponds to a different model M.

Fact Route: A fact route $a_i \xrightarrow{p:(a_k,\ldots,a_l)} a_j$, corresponds to the case where p originates in a_i, terminates in a_j, and visits steps $a_k, \ldots, a_l$ (also called stations). Originating in a step corresponds to the step adding the fact. Terminating in a step either corresponds to the step deleting the fact, or that the step is the goal step. Visiting a step corresponds to the step requiring the fact as a precondition. A fact route is legal for a plan π if in the plan: a_i precedes a_j, and each $a_k, \ldots, a_l$ succeeds a_i and either precedes a_j or is a_j. Conductor enforces that the user creates only legal fact routes.

Open Condition: An open condition (a_i, p) denotes that p is a precondition of a, $p \in \mathsf{pre}(a)$, and p is not true prior to executing a_i. An open condition occurs if no prior action adds p, or some action a_t before a_i deletes p and no third action between a_t and a_i adds p.

Figure 4: Conductor displays how each fact originates, is used as a precondition, and terminates.

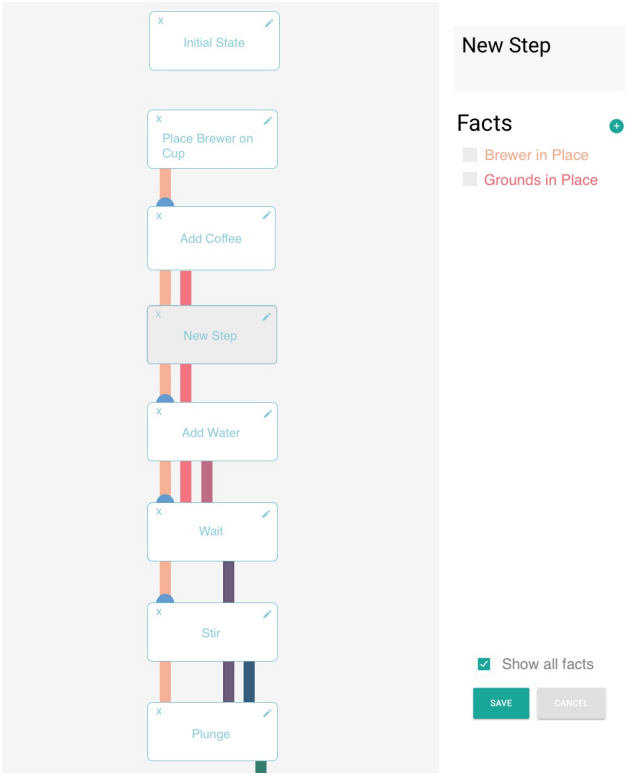

Figure 5: Users add steps to Conductor and can edit their details. The details panel allows users to change the step name, and the fact routes impacting or impacted by the step.

Conductor

Conductor allows the user to perform several modifications to a plan, including adding and removing steps or fact routes. These modifications inform Marshal about the domain model and help it develop its interpretations. Marshal then develops a set of open conditions affecting the plan and notifies the user via Conductor. In this section, we describe how a user can interact with Conductor.

Overview: Conductor displays a plan as a sequence of steps (white boxes) that start at the top and proceed to the bottom. Figure 4 illustrates an optional view that provides the details of each fact route. For example, it illustrates the following fact routes (among others):

$$\text{Add Coffee} \xrightarrow{\text{Grounds in Place:(Wait)}} \text{Wait}$$

$$\text{Add Water} \xrightarrow{\text{Water in Place:(Wait)}} \text{Wait}$$

$$\text{Wait} \xrightarrow{\text{Brewed:(Plunge)}} \text{Plunge}$$

The plan view on the left of the figure illustrates the fact routes by the vertical colored lines. Each fact route flows from the bottom of the originating step to the top of the destination step. Each step visited by the fact route includes a blue semi-circle station on the top of the step that is overlaid on the route. For example, the "Grounds in Place" fact route is shown as the second fact route from the left in orange-red.

The side panel also shows a list of fact routes for facts, including the step where they originate (dot with line at bottom), visit a station (blue text), do not visit a station (grey text), and where they terminate (dot with line at top). We use the terminology "terminate" to capture both the case where a fact becomes false (is deleted) or reaches the goal.

Adding and Editing a Step: Figure 5 illustrates a procedure after the user has elected to add a step, initially titled "New Step", between "Add Coffee" and "Add Water". The edit step panel to the right allows the user to modify the details of the step, such as the step name, and facts relevant to it. Because the facts "Brewer in Place" and "Grounds in Place" correspond to fact routes crossing the new step, they are listed as relevant facts. We discuss modifying the fact routes below. When the user adds a step to a plan π (transforming it to π', Conductor generates an observation (π, π') for Marshal.

Removing a Step: Figure 6 illustrates a plan before and after removing the "Place Brewer on Cup" step. There was previously a fact route from this step to the Goal, that visited several intermediate steps. When the user removes a step in plan π (transforming it to π', Conductor generates an observation (π, π') for Mar-

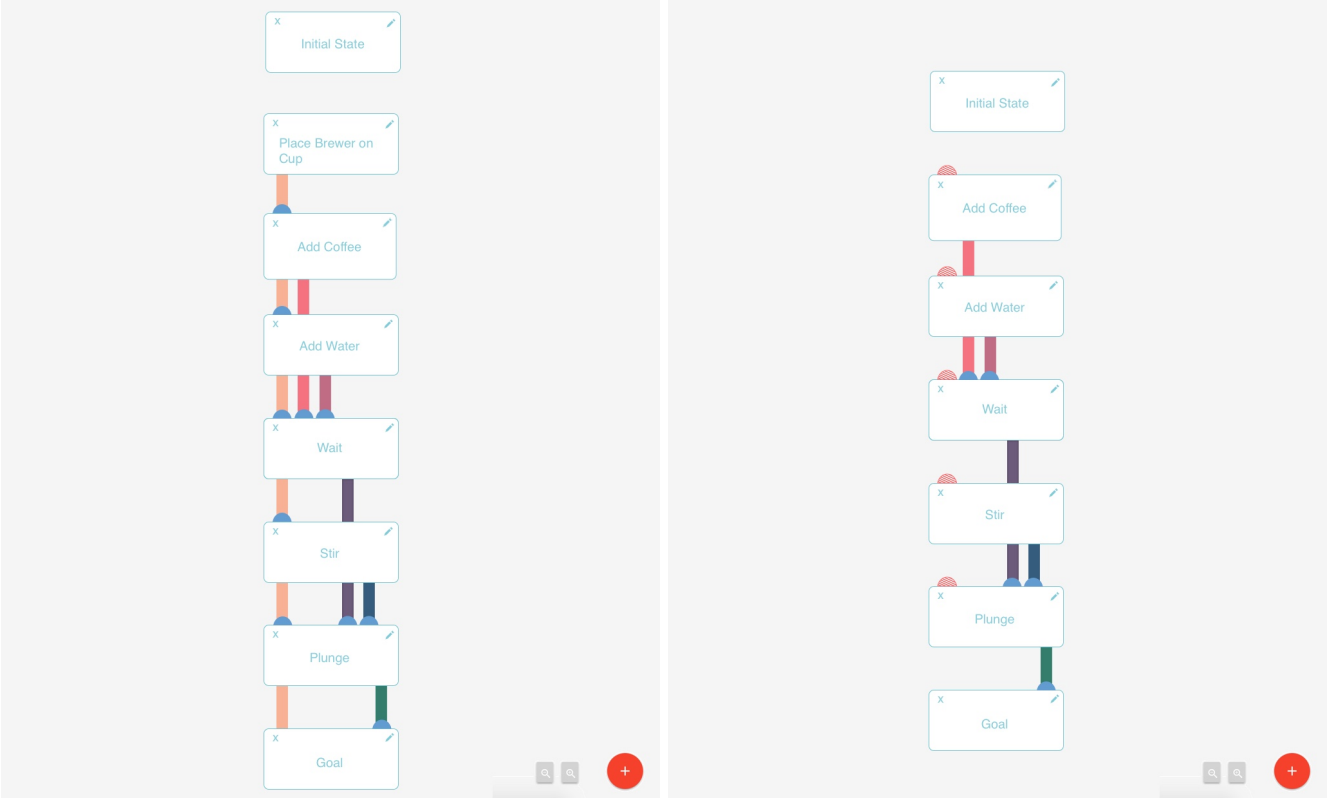

Figure 6: Users can remove steps in Conductor, which can disrupt the fact routes. The left illustrates the procedure before removing the "Place Brewer on Cup" step and the right illustrates after. Marshal computes the impact on the fact routes and marks any open conditions in red.

shal. In response, Marshal recomputes the fact routes and open conditions. Each of the steps with a red station denoting that it requires the fact as a precondition, now has an open condition. The image on the right illustrates each open condition as a red semi-circle on the corresponding step.

Adding, Editing, and Removing an Fact Route: Figure 7 illustrates adding a fact route to a procedure. For example, the user adds the fact route:

$$\text{Add Water} \xrightarrow{\text{Water in Place:(Wait)}} \text{Wait}$$

The left-most image illustrates the plan prior to adding the fact route, and after the user clicks the edit (pencil icon) on the "Add Water" step and adding a new fact (clicking the blue "+" icon, and entering the name of the fact). By default, facts added to a step in this fashion originate in the step (as denoted by the circle with a line at the bottom next to the "Water in Place" fact) and terminate at the specified endpoint. Next, the user must edit the "Wait" step (right-side of the figure), where the "Water in Place" fact has been automatically populated in the fact list as an end point (denoted by the circle with a line at the top). The user clicks the precondition checkbox to state that its a precondition of the step. The user can optionally remove a fact route as well by clicking the trash can in the fact view. When the user adds a fact route of the form $a_i \xrightarrow{p:(a_k,\ldots,a_l)} a_j$, Conductor generates an observation $(a_i \xrightarrow{p:(a_k,\ldots,a_l)} a_j, \text{true})$ for Marshal. Similarly, removing a fact route results in an observation $(a_i \xrightarrow{p:(a_k,\ldots,a_l)} a_j, \text{false})$ for Marshal. Fact route edits are described by a pair of observations that correspond to removing the prior fact route and adding the new fact route.

Labeling an Open Condition: Figure 8 illustrates a case where a user addresses open conditions. The user may either dismiss the open condition (clicking the red "?" button and selecting ignore), meaning that it is not a precondition of the step, or establish a fact route that satisfies the open condition. When user dismisses the open condition it results in an observation to Marshal of the form $((a_i, p), \text{false})$. Otherwise, modifying a fact route results in the fact route observations described above.

Possible Model Features: Figure 9 illustrates possible modifications to the domain model identified by Marshal. The left-most image illustrates a case where Marshal has identified several possible open conditions, denoted by the blue striped pattern over the possible stations and the blue "?" in the step details panel. Clicking the "?" will allow the user to confirm or deny

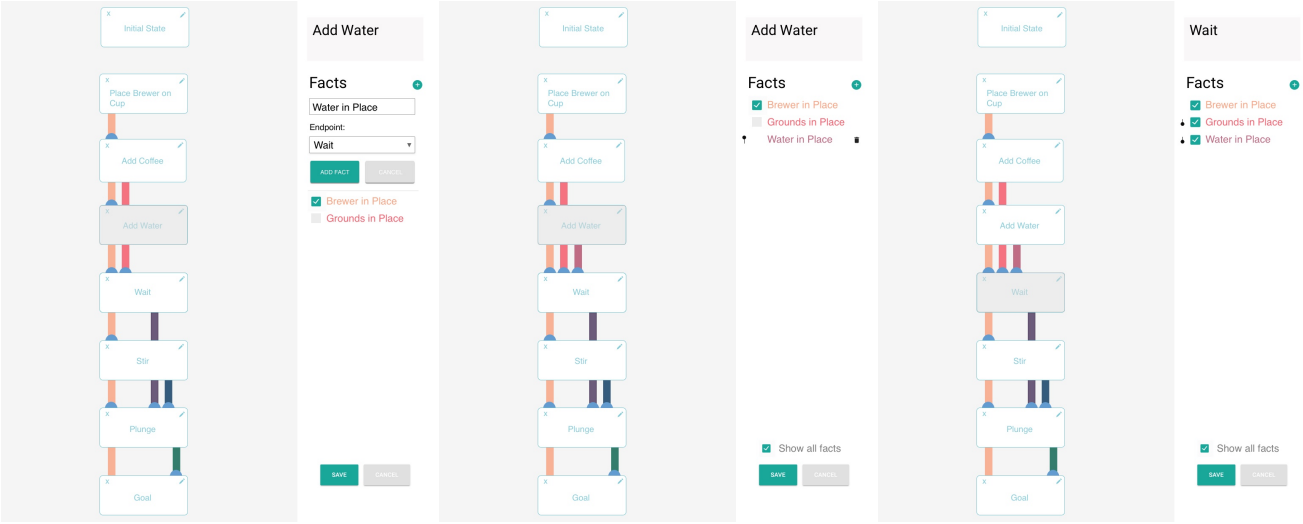

Figure 7: Users can add fact routes of the form $a_i \xrightarrow{p:(a_k,\ldots,a_l)} a_j$ in Conductor by adding a fact to a step (left, before, and center, after), and then terminating the route and adding stations (right).

the existence of the stations and it will result in either an open condition (red station) or removal of the station. The center image illustrates that Marshal has identified a possible add effect for the initial state step, along with the stations (as before). The possible add effect means that there is a possible fact route originating in the initial state, and the striped fill on the route highlights that it is hypothesized by Marshal. The rightmost image illustrates how Conductor communicates that Marshal hypothesizes that "Add Coffee" deletes (is the terminus) of the "Brewer In Place" fact route originating at the initial state. The fact route is first solid and then has a striped fill to indicate that it may continue or not, depending on the hypothesized delete effect.

Marshal

Marshal observes modifications to the plan and fact routes, updates its interpretations of the domain model, and then notifies Conductor of any new fact routes and plan flaws. The observations are as follows:

- (π, π'): a plan π and its modification π'.
- $(a_i \xrightarrow{p:(a_k,\ldots,a_l)} a_j, \ell)$: a fact route with truth label $\ell \in \{\text{true}, \text{false}\}$.
- $((a_i, p), \ell)$: an open condition for fact p at action a_i with truth label ℓ.

Marshal processes these observations to update its interpretations of the domain model. For each plan modification (π, π'), where π' adds an action to, or removes an action from π, Marshal develops explanations of the change. For example, adding an action a_i to π can be explained by a_i adding a fact p, which is an open condition in the plan. This translates into modifying the model interpretations to capture that a_i adds p.

Similarly, observing a fact route $(a_i \xrightarrow{p:(a_k,\ldots,a_l)} a_j, \text{true})$ will cause Marshal to explain the fact route and modify its interpretations of the model. One possible explanation is that action a_i adds fact p, actions $a_k, \ldots, a_l$ use p as a precondition, and action a_j deletes p.

Observing a label for an open condition is handled in a similar fashion. Explanations for open conditions relate to how the condition is established or is not a precondition.

After updating its interpretations, Marshal notifies Conductor of fact routes, open conditions, and threats that it identifies given its knowledge of the domain model. With respect to a plan π and its knowledge about the domain model, Marshal provides the following forms of feedback to Conductor:

- $a_i \xrightarrow{p:(a_k,\ldots,a_l)} a_j$ a fact route exists for fact p from action a_i to a_j.
- (a_i, p) an open condition exists for fact p at action a_i.

Marshal generates this feedback by simulating execution of the plan under its domain model interpretations. From each possible execution, Marshal estimates the probability and entropy of each fact route, open condition, and threat. Marshal applies a user defined minimum threshold to determine which it reports. Marshal reports those exceeding the threshold for probability as "known" and those for entropy as "possible". Conductor displays both forms of knowledge, as described in the previous section.

EVA 22 Scenario

We also developed a procedure in Conductor for an NASA Extravehicular Activity (EVA) procedure. The

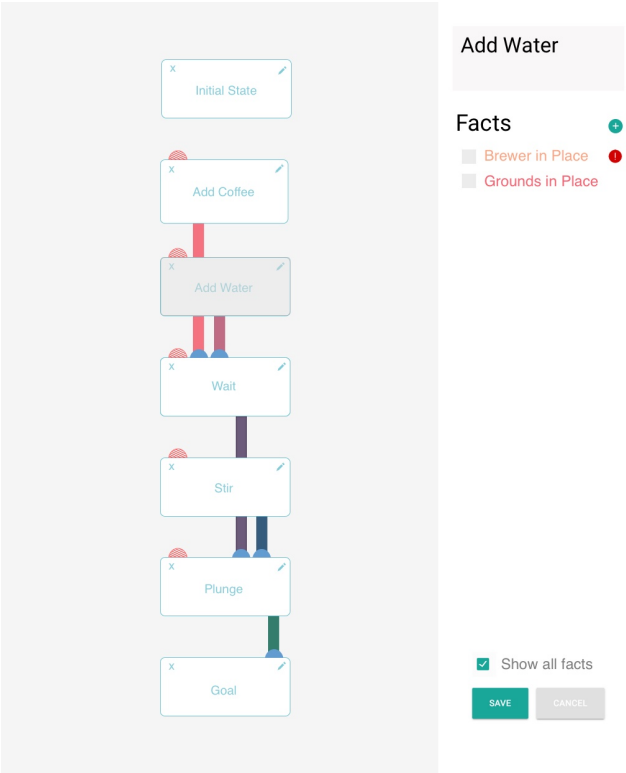

Figure 8: Users can address open conditions identified by Marshal in two ways. The user either establishes the conditions by adding fact routes, or clicks the red "!" button next to the fact to acknowledge or dismiss the open condition.

procedure, called EVA 22, involves two astronaut roles EV1 and EV2, and we illustrate a portion of the procedure for EV1 in Conductor. Figure 10 illustrates a portion of the procedure in PRIDE View with fully detailed instructions for each step. Figure 11 illustrates the annotated steps for EV1 in Conductor. The figure shows the first and last halves of the procedure side-by-side. There are a number of fact routes in the procedure. For example, the fact routes in Table 1 appear in Figure 11.

The procedure involves replacing a failed space to ground transmitter/receiver controller (SGTRC). The fact spare-SGTRC-installed-at-worksite is true between steps SGTRC R&R / MISSE 8 Retrieval and Cleanup, and is required as a precondition of the Goal step (i.e., it is a goal of the procedure). The fact ev1-has-PGT is true throughout the whole procedure, and is a precondition of the Setup and SGTRC R&R / MISSE 8 Retrieval steps. Noting such invariants is useful in developing libraries of procedures because the requirements that must be satisfied to run the procedure are explicit.

The preliminary user feedback we received from NASA EVA procedure authors was that Conductor and the concept of a fact route are intuitive. They believed that Conductor would be useful for developing libraries of annotated procedure elements that can later be integrated semi-automatically into a larger procedure. At their suggestion, we are investigating enhancements to accelerate procedure annotation by using existing domain ontologies for facts and steps.

Related Work

itSimple (Vaquero et al. 2013) is a knowledge engineering tool for planning that allows both domain model creation and plan authoring. itSimple focusses primarily on complete and correct domain modeling so that it can then task an automated planner to generate a plan. Conductor and Marshal focus more on semi-automated plan authoring with semi-automated domain authoring. Conductor aims at a more novice user audience, whereas itSimple at improving the productivity of experts.

NASA has a long history of developing plan authoring tools, which includes tools such as Mapgen (Ai-Chang et al. 2004) and, more recently, OpenSPIFe (Aghevli, Bencomo, and McCurdy). Both Mapgen and OpenSPIFe support automated planning and constraint checking, but require a complete domain model. They allow a restricted form of integrated authoring and domain modeling wherein users may relax constraints. Conductor also allows users to relax the domain model (e.g., remove open conditions), but goes further by helping them add to the model. While Conductor deals with a much simpler class of domain models, it can, in principle, support richer domain models (e.g., temporal and resource constraints).

ReACT! (Dogmus, Erdem, and Patoglu 2015) is similar to Conductor, in that it helps users encode the semantics of operators. It differs in that it focusses on complete specification of preconditions and effects, where Conductor allows some ambiguity in favor of simplicity. ReACT! also handles more expressive hybrid models, where Conductor and Marshal focus upon STRIPS.

Conductor and Marshal address a problem similar to that of KEWI (Wickler, Chrpa, and McCluskey 2014). User-friendly environments for encoding model knowledge by domain experts can help make planning accessible. KEWI differs from our work in that it requires users to be more formal in the knowledge that they encode, structuring it around an ontology. We see this as a trade-off in user skill and knowledge engineering tool support. Conductor and Marshal require comparatively little structure.

The Procedure Integrated Development Environment (PRIDE) (Kortenkamp et al. 2008) permits users to develop procedures in a palette-based drag-and-drop interface. While PRIDE allows much more detailed procedures than the types of plans developed in Conductor, it does not provide the same type of user support. PRIDE automates aspects of the procedure development by using PDDL models (Bonasso and Boddy

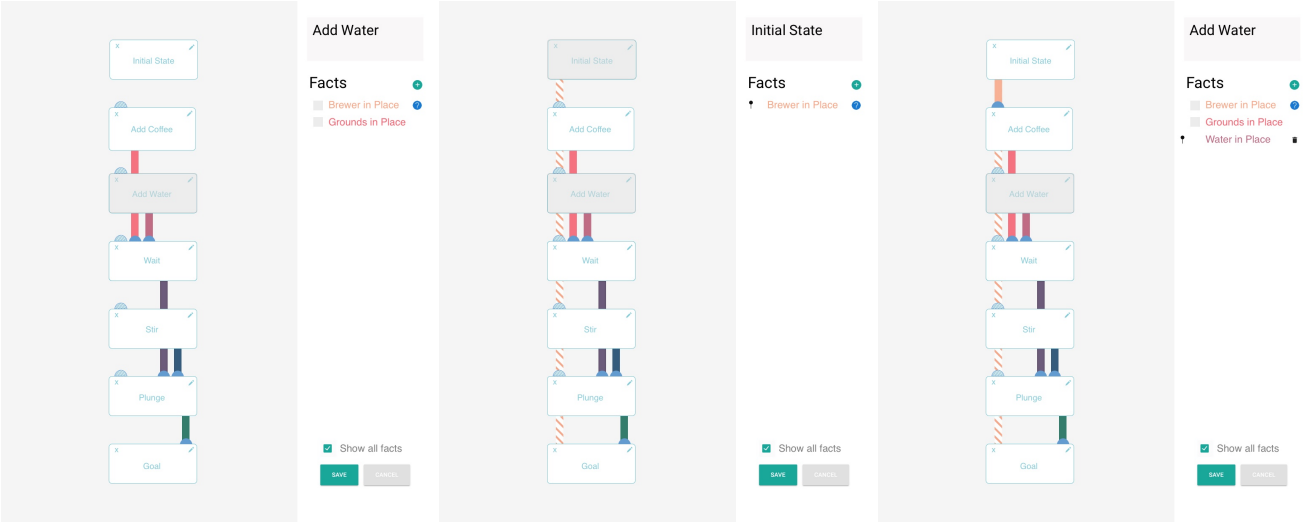

Figure 9: Users can address possible domain model features identified by Marshal similar to addressing open conditions. Users can modify fact routes to match Marshal's suggested updates, or dismiss them.

Initial State	failed-SGTRC-installed-at-worksite:(SGTRC R&R / MISSE 8 Retrieval,Remove failed SGTRC) →	Remove failed SGTRC
SGTRC R&R / MISSE 8 Retrieval	spare-SGTRC-installed-at-worksite:(Goal) →	Goal
Initial State	ev1-has-PGT:(Setup,SGTRC R&R / MISSE 8 Retrieval) →	Goal

Table 1: Fact routes for EV1 procedure in Figure 11.

2010), but like many of the aforementioned tools it separates domain modeling and procedure authoring. We are developing Conductor as tool within the PRIDE suite that can help users design consistent procedures at a high-level, and then use PRIDE to fill in the details necessary for execution.

Conclusion & Future Work

We present a new plan authoring tool called Conductor. Conductor enables novice users to author plans and annotate them with a new form of knowledge called a fact route. Fact routes are easy to specify and are very informative, yet incomplete. Conductor helps overcome the incompleteness by interacting with the Marshal model maintenance system to develop possible interpretations of the model. Using these interpretations, Conductor is able to elicit refinements to the model that could impact the plan. By seamlessly integrating plan authoring and domain modeling, Conductor and Marshal allow novice users to quickly begin authoring plans without a steep learning curve.

While it is possible to extend Marshal to more expressive planning formalisms, such as temporal or hybrid planning, it is not immediately obvious how to extend Conductor. The metro map metaphor should accomodate temporal actions, and will more closely resemble a Gantt chart. Fact routes may extend to hybrid models if they are reinterpreted as resource envelopes, but we may lose some of the clarity inherent to boolean variables.

References

Aghevli, A.; Bencomo, A.; and McCurdy, M. Scheduling and planning interface for exploration (spife). *ICAPS 2011* 54.

Ai-Chang, M.; Bresina, J. L.; Charest, L.; Chase, A.; Hsu, J. C.; Jónsson, A. K.; Kanefsky, B.; Morris, P. H.; Rajan, K.; Yglesias, J.; Chafin, B. G.; Dias, W. C.; and Maldague, P. F. 2004. MAPGEN: mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems* 19(1):8–12.

Bonasso, P., and Boddy, M. 2010. Eliciting planning information from subject matter experts. *KEPS 2010* 5.

Bryce, D.; Benton, J.; and Boldt, M. W. 2016. Maintaining evolving domain models. In Kambhampati, S., ed., *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 3053–3059. IJCAI/AAAI Press.

Dogmus, Z.; Erdem, E.; and Patoglu, V. 2015. React!: An interactive educational tool for AI planning for robotics. *IEEE Trans. Education* 58(1):15–24.

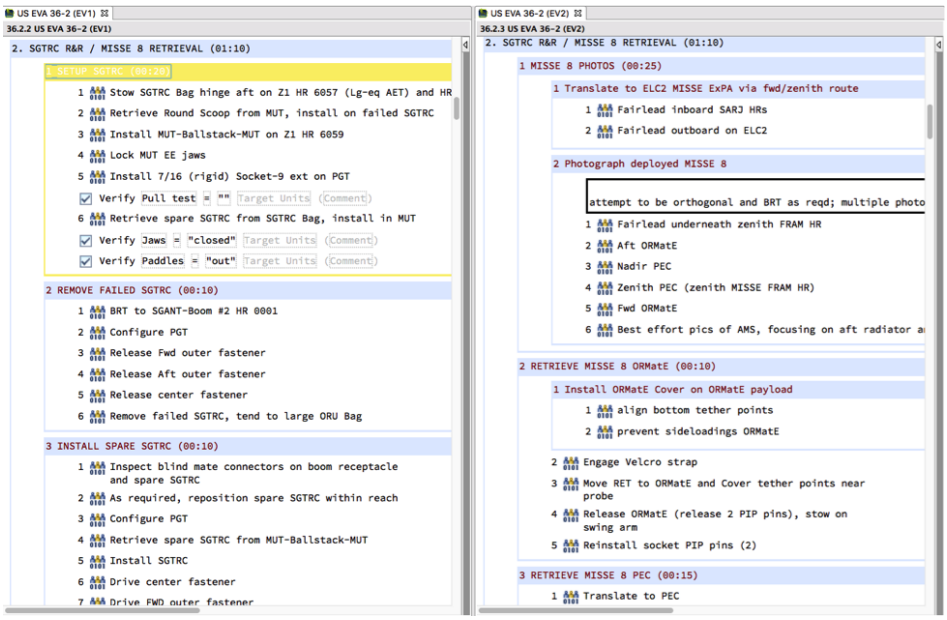

Figure 10: EVA 22 procedure for EV1 represented in PRIDE.

Kortenkamp, D.; Bonasso, R. P.; Schreckenghost, D.; Dalal, K.; Verma, V.; and Wang, L. 2008. A procedure representation language for human spaceflight operations. In *Proceedings of the 9th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS-08)*.

Vaquero, T. S.; Silva, J. R.; Tonidandel, F.; and Beck, J. C. 2013. itsimple: towards an integrated design system for real planning applications. *Knowledge Eng. Review* 28(2):215–230.

Wickler, G.; Chrpa, L.; and McCluskey, T. L. 2014. KEWI - A knowledge engineering tool for modelling AI planning tasks. In Filipe, J.; Dietz, J. L. G.; and Aveiro, D., eds., *KEOD 2014 - Proceedings of the International Conference on Knowledge Engineering and Ontology Development, Rome, Italy, 21-24 October, 2014*, 36–47. SciTePress.

Figure 11: Portion of EV1 steps for EVA 22 represented in Conductor.