27th International Conference on
Automated Planning and Scheduling
June 19-23, 2017, Pittsburgh, USA

UISP 2017

Proceedings of the Workshop on
User Interfaces and
Scheduling and Planning

Edited by:

Richard G. Freedman and Jeremy D. Frank

Organization Committee

Jeremy D. Frank, NASA Ames Research Center

Richard G. Freedman, University of Massachusetts Amherst

Amedeo Cesta, Institute of Cognitive Sciences and Technologies

Subbarao Kambhampati, Arizona State University

David Kortenkamp, TRACLabs

Ronald P. A. Petrick, Heriot-Watt University

Kartik Talamadupula, IBM

Shlomo Zilberstein, University of Massachusetts Amherst

Foreword

Automated planning and scheduling technologies are employed in applications ranging from robotics to factory organization to travel design. The utility of automated planning and scheduling systems is often constrained by the design of the user interfaces. While many of these applications have been designed by members of the ICAPS community, as a whole we have noted that the real world is overlooking automated planning and scheduling technologies in domains where it should be used. A lack of good user interfaces may be one reason for this, prompting this workshop as a way to investigate better ways to apply our tools.

In parallel with this thread is the desire to explore how automated planning and scheduling can help design user interfaces. Workflows for many different user interface tools can be constructed using planning systems as well as other automated reasoning technologies. Historically, there have been a small number of investigations of this type; this workshop presents a new set of challenges to, as well as revives interest in past research initiatives of, the ICAPS community to help design better user interfaces.

The time is also right for the ICAPS community to investigate novel user interface modalities such as natural language processing and augmented reality as ways to facilitate human-planner interaction. While natural language processing systems have been developed over at least the past 20 years, the advent of commodity spoken language systems (e.g. Siri) and natural language processing systems on a chip provides exciting opportunities for integration with automated planning and scheduling. Augmented reality is a 'rising' technology; when coupled with computer vision systems, augmented reality provides new, potentially disruptive methods for supporting plan execution, if not planning, and augmented reality systems may benefit from automated planning and scheduling technology as a form of user interfacing.

The goals of the UISP workshop are:
1) To emphasize how automated planning and scheduling and user interfaces can support each other;
2) To explore how user interfaces can assist various companies and everyday users in better understanding automated planning and scheduling for their own applications; and
3) To discuss how automated planning and scheduling can be used to improve user interfaces for everyday interaction.

The papers included in this proceedings illustrate a diversity of research, approaches, applications, and challenges related to these goals. We hope they will inspire others to further investigate UISP-related work and contribute to bridging the gap between user interfaces and planning and scheduling. As the future of artificial intelligence becomes more focused on the average user than the experienced scientist, such technologies will become more necessary and ubiquitous in daily lives.

Jeremy D. Frank, Richard G. Freedman, Amedeo Cesta, Subbarao Kambhampati, David Kortenkamp,
Ronald P. A. Petrick, Kartik Talamadupula, and Shlomo Zilberstein
UISP 2017 Organizers
June 2017

Contents

CHAP-E: A Plan Execution Assistant for Pilots

J. Benton and **David Smith** and **John Kaneshige** and **Leslie Keely**

NASA Ames Research Center
Moffet Field, California 94035-1000
{j.benton,david.smith,john.t.kaneshige,leslie.keely}@nasa.gov

Abstract

Pilots have benefited from ever-increasing and evolving automation techniques for many decades. This automation has allowed pilots to handle increasingly complex aircraft with greater safety, precision, and reduced workload. Unfortunately, it can also lead to misunderstandings and loss of situational awareness. In the face of malfunctions or unexpected events, pilots sometimes have an unclear picture of the situation and what to do next, or must find and follow written procedures that do not take into account all the details of the particular situation. Pilots may also incorrectly assume the mode or state of an automated system and fail to perform certain necessary actions that they assumed an automated system would handle. To help alleviate these issues, we introduce the Cockpit Hierarchical Activity Planning and Execution (CHAP-E) system. CHAP-E provides pilots with guidance toward executing procedures based on the aircraft and automation system's state and assists pilots through both nominal and off-nominal flight situations.

Introduction

Piloting aircraft requires handling input from a variety of systems, including instruments that inform a pilot of the aircraft's state (e.g., airspeed, vertical speed, altitude, attitude, and heading). While automation has a long history of assisting pilots with handling this information, when malfunctions occur, sometimes multiple messages come from distinct systems, confusing a pilot and making it difficult to understand the next best course of action. A sad example of this occurred during the Air France 447 flight, which crashed, killing all passengers and crew. Coming from Rio de Janeiro, Brazil and going to Paris, France, the flight entered a large area of thunderstorm activity that resulted in both turbulence and ice crystals forming in the pitot tubes, which measure airspeed. Though the anti-ice system came on and a warning sounded, the pitot tubes iced over and no longer provided correct airspeed, causing the autopilot and auto-thrust systems to disengage. The aircraft began to roll from the turbulence, and the pilot overcompensated because the aircraft was now in a control mode that was more sensitive to roll input. Through a series of often disparate warnings and incorrect assumptions that followed (including stall warnings and presumed assumptions that the aircraft's autopilot would not allow a high angle of attack)[1], the aircraft stalled at 38000 feet, plunging into the ocean three and a half minutes later.

This tragic incident serves as an illustrative example of how messages from disparate systems, unclear procedures, and lack of basic data regarding the aircraft's automation state can cause serious issues for pilots and their flights. Numerous other examples exist, including American Airlines 268 and Turkish Airways 1951. Indeed, 55% of all major incidents are due to system malfunctions, and a primary reason those contributed to bad outcomes related to pilots' inability to accurately assess the nature of the failure (FAA 2013). Our objective is to help flight crews by providing a global picture of expected procedures given the aircraft state. Toward these ends, we seek to provide pilots with procedural guidance during flight, keeping track of the aircraft state and providing suggested procedures for pilots to follow.

More specifically, we are interested in the problem of real-time monitoring of all phases of airliner flight, and providing feedback to the pilots when actions are overlooked or are inappropriate, or when the conditions of flight are no longer in accordance with the objectives or clearance.[2] Traditionally, pilots have made use of written or electronic checklists to verify that appropriate actions have been performed and that the aircraft reached the proper state for each particular phase of flight. While these have served to standardize procedures and ensure that critical items have not been overlooked, checklists are both static and passive. For example, the pre-landing checklist confirms that the flaps are at the landing setting, the landing gear is down, the airspeed is in an appropriate range for the landing weight, the approach is stabilized, and the autobrakes are armed. It does not tell the pilots when to lower flaps, when to lower landing gear, what modes and settings to select for the autopilot, or whether the selected landing speed and flap settings are even appropriate for the runway length, wind conditions, and current runway braking action. In other words, the checklist helps confirm the state of the aircraft, but provides no guidance about when or how to achieve that state. This information is

[1] The angle of attack is the angle between the wing and airflow.

[2] This includes things like speed, altitude, descent/climb rate, autopilot mode and settings, route compliance, flap settings, fuel state, etc.

all buried in the pilot's training and expertise, and in procedures in the Pilot's Operating Handbook (POH). However, details can get overlooked when the crew is fatigued, the crew is overworked (e.g., due to weather conditions), or in the event of system failures.

To tackle this problem, we introduce the Cockpit Hierarchical Activity Planning and Execution (CHAP-E) system, a decision support and procedure display system. CHAP-E can display pre-defined plans (procedures) or interface with a situation-aware automated planner to generate and display appropriate procedures. In this way, it can be viewed as a planning, monitoring and execution assistant for the aircraft flight.

This paper focuses on the CHAP-E display and its use during flight. We designed the CHAP-E display to account for the constant movement that occurs during flight. Unlike some domains, during flight the state of the world changes continuously but predictably over time, depending upon action execution. This enables CHAP-E to determine how to safely execute a plan. The state information includes events such as reaching waypoints, instrument data (e.g., altitude and air speed) and aircraft configuration (e.g., flaps and landing gear positions). To predict these, CHAP-E uses an external simulator called the Trajectory Prediction System (TPS) (Kaneshige et al. 2014). It can use these predictions to determine the earliest time when a pilot may begin actions in the plan and the latest time an action can be executed for the plan to remain successful.

We begin by discussing work related to automation during flight and plan execution displays. We then follow with an overview of the CHAP-E system. Finally, we end with a discussion on future work.

Related Work

Automation systems have a long history in aviation. As (Billings 1996) points out, making flight more resistant and tolerant to error stands as the primary purpose of automation assistance in aviation. Despite this, little work has been done in assisting pilots by displaying procedures to them, ensuring their applicability during flight, and monitoring the execution of those procedures. Perhaps the closest match to CHAP-E is MITRE's Digital Copilot, which is designed for smaller single-pilot aircraft and informs the pilot of common mistakes and constraint violations during flight (MITRE 2016).

Other work on procedure displays has been implemented within the context of space missions. The NASA Autonomous Mission Operations (AMO) project used a user interface to track a spacecraft's life support system activity (Frank et al. 2015). It offered recommended activities based on the state of the spacecraft and current operating constraints. Personnel on the spacecraft forwarded the recommendation to flight controllers, and if approved by the flight controllers, the activity would be scheduled and displayed. It also used a system that helped the crew track the progress of plan execution. That system, called WebPD, was integrated with spacecraft systems to provide information about the spacecraft's state. The system then displayed serial procedures along with important relevant state information related to each step in the procedure (Stetson et al. 2015; Frank et al. 2013). A similar system, called the Procedure Integrated Development Environment (PRIDE), was implemented to assist in the development of procedures. The procedures are stored using the Procedure Representation Language (PRL). The procedure view component, PRIDE View, allows a user to follow a procedure step-by-step (Kortenkamp et al. 2008).

Another comparable system is RADAR (Vadlamudi et al. 2016), which assists in producing plans by generating landmarks and offering action suggestions based upon them. Unlike the other systems listed, RADAR uses PDDL (Fox and Long 2003).

Plan Execution Assistant

The design of CHAP-E centers around reducing human error and its potentially negative effects by providing decision support to human pilots. We define human error as action or lack of action taken by a human with unintended effects. Without knowing the intentions of a human actor, we cannot determine whether an error has taken place. A human must share the intention of their actions to identify an error. This makes detecting human error a difficult, complex problem. In our current version of CHAP-E we do not expect to identify all human errors. Instead, we assume a human pilot never intends to cause goal failure or violate important safety and operational constraints, which ties human intent to pilots maintaining safe flight. Fortunately, we can focus on negative effects assuming that a human will want to avoid making errors that would cause potential hazards or cause failure to achieve a given goal.

In this section, we present an overview of the CHAP-E system currently in development, with a particular focus on the approach and landing phases of flight. First, we discuss the CHAP-E plan representation, then how we can determine whether a plan may succeed without violating constraints. Finally, we discuss the CHAP-E display and its operation.

Plan Representation

CHAP-E uses hierarchical plans with causal links. The primitive actions in the plan can be simulated to ensure they do not violate important constraints. As seen in Figure 1, in the plan hierarchy, the highest level task is a flight from the departure airport to the destination airport, flight(from, to). This expands into a serial (via causal links) set of subtasks: ⟨FileFlightPlan(from, to), ObtainClearance(from, to), Taxi(rnwy) Fly(from, to), Taxi(gate), Shutdown⟩.[3] We can further break down the Fly(from, to) action into the phases of flight: ⟨Takeoff(from, rnwy), Climb, Cruise, Descend, Approach, Land(rnwy)⟩. Expanding the Approach phase, we have a set of primitive actions taken by the pilot. These individual actions have enabling safety conditions associated with them, such as a particular segment on the approach, an airspeed range, or an altitude range.

[3]We simplify the example by removing some parameters and tasks.

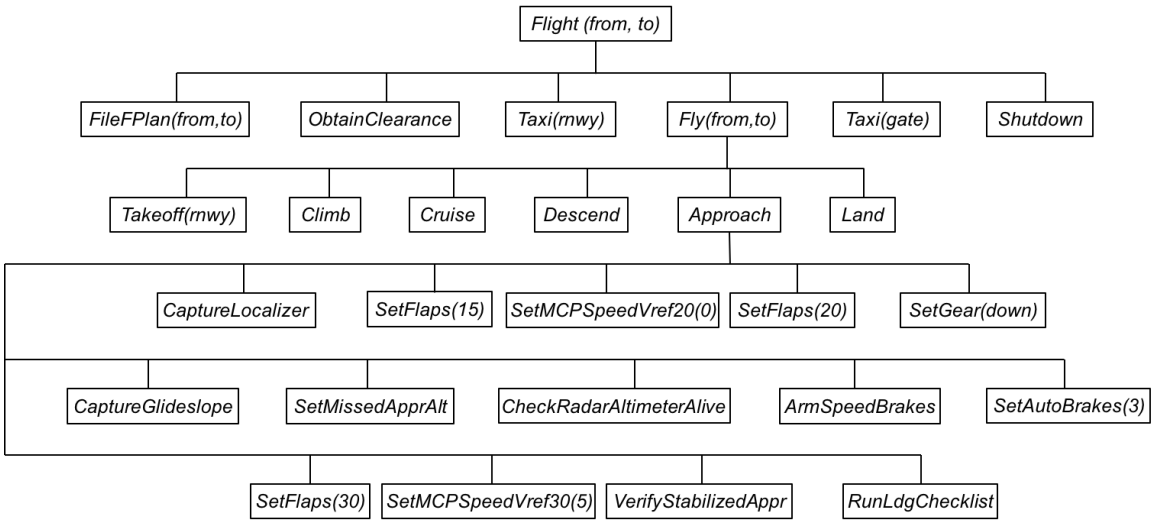

Figure 1: CHAP-E hierarchical plan with the approach phase expanded down to primitive actions

The hierarchical structure provides several advantages. First, much of the expansion cannot take place initially, because some of the parameters and constraints are not yet known. For example, we cannot always initially expand Taxi(rnwy), because the taxi route and the runway may have not been assigned yet. Before this expansion can take place, the initial part of the plan must be executed – we must file the flight plan, and obtain the taxi clearance. Similarly, the Departure, Cruise and Descent activities of the Fly subtask cannot be expanded until the aircraft obtains a route clearance. The Approach and Landing activities usually cannot be expanded until later in the flight when the approach and runway are assigned by Air Traffic Control (ATC) and accepted by the pilots. This will often depend on the traffic and weather conditions at the time of arrival. For example, the current wind conditions usually dictate which runways are viable, and ceiling and visibility constrain the approaches that are possible. This necessitates interleaving the hierarchical expansion of the plan and the plan's execution.

One of the challenges of representing these plans is that many of the actions are keyed off of particular events, rather than times. For example, a standard practice is to lower flaps to 20 degrees and lower the landing gear just before intercepting the glideslope (the vertical guidance for the aircraft) on an approach. Typically, this happens just outside of the Final Approach Fix, a designated waypoint about 5nm (nautical miles) from the end of the runway. The trouble is, there is some uncertainty about the exact time at which this event will occur, since it depends on the aircraft's exact speed and altitude, and on the wind conditions; if the aircraft is a bit faster than expected or a bit high, this event will occur sooner, if the headwinds are higher than expected, this event will occur a bit later. As a result, many of the actions in a CHAP-E plan are triggered off of events, rather than times or the completion of preceding actions. Frequently, these events involve reaching a particular waypoint or distance from a waypoint, reaching a particular altitude, or reaching a particular airspeed.

Figure 2 shows the plan and profile views for the ILS 28R approach into San Francisco.[4] The plan view gives a map-like picture of the approach as seen from above, with a transition to the approach starting at the waypoint ARCHI, intercepting the final approach course at the waypoint DUMBA, and continuing through the Final Approach Fix, the waypoint AXMUL, to the runway. The profile view shows a vertical slice of the altitude profile for the final segments of the approach.

Figure 3 shows a small fragment of a detailed plan for intercepting and flying this approach, for an aircraft beginning just east of the ARCHI waypoint. The plan contains three types of statements: *Events* that are expected to occur, *Actions* that the pilots must perform, and *Monitors*, which indicate conditions that must be maintained throughout some interval. Each event is characterized by a label, followed by the event. For example, the first event is that of crossing the waypoint ARCHI on the transition to the approach. The events ZILED, GIRRR, DUMBA, CEPIN, AXMUL, and RW28R also refer to the crossing of waypoints. The next five events are prefaced by `before!` conditions, which indicate a hard constraint that the event must occur before another event (otherwise the plan becomes invalid). The first of these is that we must have the clearance for the approach from ATC before crossing the ARCHI waypoint. The next three refer to the airspeed dropping below the maximum allowed value for a particular flap setting. The final two refer to the autopilot capturing the localizer and glideslope – the lateral and vertical guidance for the approach. Finally, A1500 and A1000 refer to the events when the altitude becomes less than 1500 ft and 1000 ft above the runway.

Actions are much like events, but these are things the pilots must do. These usually contain both hard and soft con-

[4]Note that waypoints follow a standard naming convention of five all-capitalized letters.

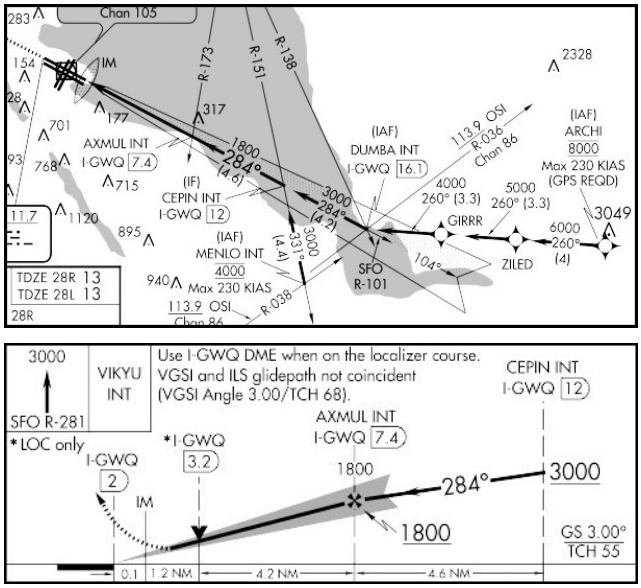

Figure 2: Plan and profile views for the ILS 28R approach into San Francisco

straints (preferences). For example, the first action says that after the clearance event, and before the GIRRR waypoint the pilots must arm the localizer in the autopilot, which allows the autopilot to follow the lateral guidance. This is a hard constraint, as indicated by the exclamation point (!) an the end of `between!`. There is also a soft constraint (preference) that this happen between ARCHI and ZILED (no exclamation point). The second action is similar, with a hard constraint window, and a soft constraint that the action happen before CEPIN. The third action is also prefaced by both hard and soft constraints and specifies a sequence of two events: setting the flaps to 20, and setting the autopilot speed window to the value Vref20. Like events, actions can have names, and the first of this sequence is named F20, which the subsequent action is conditioned on. The fourth action, lowering the landing gear, has a hard constraint that it must be performed before altitude 1500 and a soft constraint to do it after setting the flaps to 20, and before AXMUL. The final action sequence has a hard constraint and two soft constraints. This is needed in this case because we do not know, a priori, which of the events, Gear or AXMUL, will occur first, and we prefer that the action be done after both events have occurred.

Monitors are conditions that must hold over some period of time. For example, the first monitor states that the airspeed must always remain between the reference speed and the maximum speed for the particular flap setting being used at the time. The second and third monitors state that the localizer and glideslope must remain captured, and the final monitor states that the flaps must remain in the landing configuration. If any of these conditions are violated, the plan becomes invalid and must be revised.

There are a couple things worth noting about this plan:

- Most actions are conditioned on events, rather than on other actions. In many cases actions indirectly control when these events will occur, but there is some uncertainty, and it is the events that serve as constraints on when to perform the actions.
- Traditionally, "flexible" plans have been used to help deal with duration and time uncertainty. In a flexible plan, actions are partially ordered, and may be restricted to designated time windows. That is true here also, but the flexibility is expressed in terms of events that will manifest in terms of time windows. So, for example, the first action specifies that the localizer should be armed after the event where the clearance is obtained, and between the events of crossing ARCHI and GIRRR. These events will ultimately prescribe a time window for the action at execution time.
- Monitors are like overall conditions or durative goal conditions. They specify conditions that must hold between particular events. However, these monitors often span several low level actions. As a result, they are associated with (and come from) higher level tasks in the hierarchy. In the example above, they are associated with the Approach task instantiated with the San Francisco ILS 28R.
- The inclusion of both hard and soft constraint windows (discussed later) on actions is particularly important for

```
Events {
  ARCHI: cross(ARCHI) ;
  ZILED: cross(ZILED) ;
  GIRRR: cross(GIRRR) ;
  DUMBA: cross(DUMBA) ;
  CEPIN: cross(CEPIN) ;
  AXMUL: cross(AXMUL) ;
  RW28R: cross(RW28R) ;
  before![ARCHI] {CLR: start(Clearance = ILS28R.ARCHI)} ;
  before![ARCHI] { F5max: start(IAS <= Vmax5)} ;
  before![CEPIN] {F20max: start(IAS <= Vmax20)} ;
  before![AXMUL] {F30max: start(IAS <= Vmax30)} ;
  before![DUMBA] {LocCap: start(FMA-Lateral = LOC)} ;
  before![AXMUL] {GSCap: start(FMA-Vertical = GS)} ;
  A1500: start[Alt <= 1500AGL) ;
  A1000: start[Alt <= 1000AGL) ;
  ... }
  Actions {
  between![CLR,GRRR] & between[ARCHI,ZILED] <<ArmLocalizer>> ;
  between![LocCap,AXMUL] & before[CEPIN] <<ArmGlideslope>> ;
  between![F20max,AXMUL] & between[CEPIN,GSCap]
            <<F20: SetFlaps(20),SetMCPSpeed(Vref20)>> ;
  before![A1500] & between[F20,AXMUL] <<Gear: SetGear(Down)>> ;
  between![F30max,A1000] & after[Gear] & between[AXMUL,A1000]
            <<SetFlaps(30), SetMCPSpeed(Vref30+5)>> ;
  ... }
Monitors {
  throughout[CEDES, RW28R] {IAS in [Vref,Vmax]} ;
  throughout[LocCap, RW28R] {FMA-Lateral = LOC} ;
  throughout[GSCap, RW28R] {FMA-Vertical = GS} ;
  throughout[F30, RW28R] {Flaps = 30} ;
  ... }
```

Figure 3: Fragment of a detailed CHAP-E plan for the ILS 28R approach into San Francisco

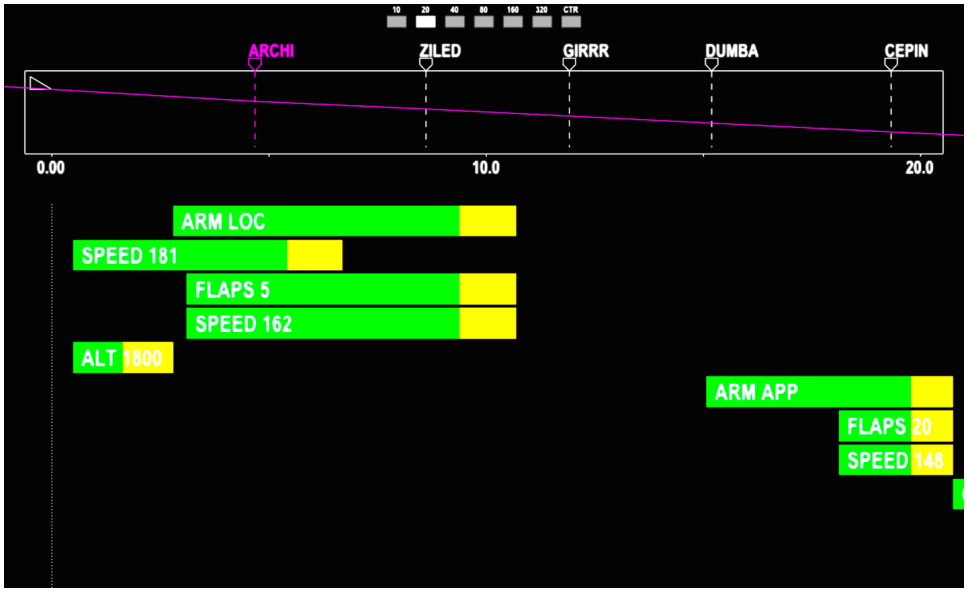

Figure 4: The CHAP-E display

monitoring pilots. Sometimes actions like lowering the landing gear could be performed much earlier, but it would be inefficient and noisy to do so. Standard procedure is to do it just before the final approach fix. The preferences therefore provide a more restrictive window where the actions should be performed, and allow us to warn the pilots when this does not happen by the end of the preference window. This allows for reasonable alerting, without becoming annoying for the pilots.

The plan fragment in Figure 3 contains approximately $1/4$ of the events, actions and monitors necessary for this particular example approach. The PLEXIL executive has been able to use the complete version of this plan to successfully approach and land a 777 at San Francisco in simulation, as well as to monitor pilot actions and warn when actions are not taken within the preference windows.

CHAP-E Display

The purpose of the CHAP-E display is to provide suggested flight procedures to a pilot for maintaining safe flight (see Figure 4). It consists primarily of a vertical profile and waypoint display, showing the expected vertical profile of the aircraft. Below that are the actions to be executed. Each action has an associated time window, representing when the pilot should perform it. The display moves horizontally as time passes.

Vertical Profile and Waypoints The CHAP-E vertical profile display includes waypoint information from the flight plan to provide a reference for the pilots. The profile shows the reference altitude of the aircraft given the current route. The aircraft is depicted as a small triangle at the upper left of this profile. Labels above the vertical profile show waypoint locations that the aircraft will reach when following the current flight plan. These provide a reference for the pilots; the labels will match instructions for airport-specific flight procedures, and may be referenced by air traffic control requests. They also help give scale for when a pilot should execute each action.

Actions and Time Windows Generally, a pilot should be allowed flexibility on when to execute actions in the plan. This means we depend on the pilots' training and habits so they may determine risk and action priority. To accommodate this aim, we display actions to the pilot in a gantt chart-like style indicating time windows for when the pilot should execute them. An example of an execution window is shown in Figure 5. Each window consists of five time points: an earliest start time (EST), preferred earliest start time (PEST), preferred start time (PST), preferred latest start time (PLST), and latest start time (LST). The display shows the preferred earliest start time, preferred latest start time and latest start time. The two end points, the earliest start time and latest start time, represent the interval in which the action may execute and the plan will still be valid. If executed outside of these times, the plan will likely fail. The preferred earliest and latest start times show when we prefer the pilot perform the action. The preferred start time is when we would ideally perform the action in a fully automated system. The execution times are given in terms of time relative to reaching waypoints during the flight plan and displayed in a gantt chart style.

To obtain the earliest and latest start times, CHAP-E can simulate the execution of the plan across varying start times for each action using an advanced simulation capable of capturing the physics and expected autopilot modes of the aircraft (see Figure 6 for a visual depiction of simulation) (Kaneshige et al. 2014). The simulation takes a series of commands and the time at which we expect the pilot to execute the commands, where each command cor-

responds to a pilot action in the plan. From this, it returns a per-second discretized profile of the state of the aircraft over the course of the displayed period. Using operational constraints, CHAP-E can determine whether the execution schedule would result in a safe flight and achievement of the goal (i.e., landing safely on a specified runway).

As indicated above, the display only shows the preferred earliest start time, the preferred latest start time and the latest start time. As discussed earlier, many actions, such as lowering the landing gear, can be performed very early without causing a plan to become invalid. Displaying the earliest start time may clutter the display with many long overlapping action boxes. To avoid that, we instead rely on preferred start times inferred from standard operating procedures, which usually suggest particular times for action execution. We use these to infer the preferred time points, and display the preferred earliest start time. The latest start time is more critical, so we display it. The preferred latest start time represents when we warn the pilot if an action has not yet been performed. For finding the preferred time points, we rely on domain knowledge for now, though we may explore other options to determine them automatically.

To show these time points, we use standard coloring often seen on flight displays. CHAP-E draws a green window between the preferred earliest start time and the preferred latest start time and an amber window between the preferred latest start time and the latest start time. If the preferred latest start time passes, a warning is spoken by CHAP-E with the action name (e.g., "gear down") and a status message is displayed at the bottom of the CHAP-E display. If the latest start time point passes, then a new plan will be displayed to recover.[5] When an action is executed by the pilot, it will be removed from the display, and the displayed action time windows below it will adjust their positions by moving up.

For the earliest and latest start times, we currently are using hard-coded time windows, but are implementing an initial technique for using the simulation to find them automatically. Our initial approach is a hill-climbing method, where we will first begin with a working schedule. Then, for each action, CHAP-E will simulate executing that action an arbitrary ϵ amount later. If the simulated trajectory continues to remain within specified operational limits and reach the goal, then the process repeats with another ϵ increase. Once a non-compliant simulated execution is found, the process begins with that action again simulating execution ϵ sooner than the original time point, and again repeating this process until a non-compliant simulated execution is found. The process then moves to the next action until all actions in the displayed plan can be simulated. In the case that not all actions are known due to lack of plan expansion, we will simulate as much into the future as possible. Note that this process is deterministic in nature and does not take into account potential exogenous events. It also treats the actions as being independent for purposes of computing the earliest and latest start times. In general this assumption is not valid. Performing an action like lowering flaps increases drag and causes the aircraft to slow down, which may change the time windows for subsequent actions and events triggered off of aircraft speed. As a result, when such an action is performed, subsequent windows may expand, shrink, or shift.

[5]Note that at the time of this writing, we are developing the planning mechanism for this step.

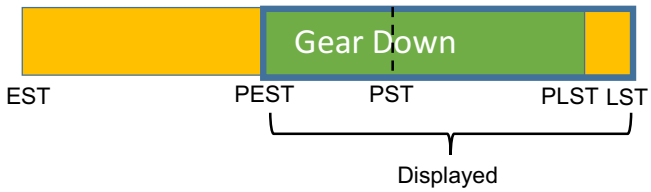

Figure 5: Time windows of an example "gear down" action, showing the earliest start time (EST), preferred earliest start time (PEST), preferred start time (PST), preferred latest start time (PLST), and latest start time (LST)

When an action is performed, CHAP-E must recognize this and remove the action from the display. This is more complex than it might first appear. For example, if the recommended action is to set the MCP-Speed to 163 knots, but the crew instead sets the speed to 165 knots, CHAP-E must determine whether this "unexpected" action still satisfies the necessary conditions for future actions that were to be achieved by the "recommended" action. If so, CHAP-E can remove the recommended action from the display. If not, CHAP-E will leave the recommended action(s), but must determine whether the unexpected action interferes with any conditions that need to be preserved in order for the plan to remain valid.

Challenges exist in continuously determining execution windows. The state of the flight continuously changes as the aircraft progresses. This means the time windows can change as unpredicted adjustments or pilot action (or inaction) occur. Though the simulation is relatively fast (approximately 60 milliseconds for a 5 minute projection), it may be impractical to rediscover time windows continuously. We're exploring several possibilities to mitigate this issue, including using courser-grained hill-climbing until important events occur (e.g., unexpected or missed pilot actions).

Discussion and Future Work

On commercial airliners, two pilots work in tandem during flight. One pilot, the *pilot flying* (PF), performs most of the direct piloting operations, including interacting with the aircraft controls. The other pilot, the *pilot monitoring* (PM), handles communication, runs checklists, and monitors the aircraft state, occasionally taking requests from the PF to perform certain actions on the aircraft. When the PF misses performing an action or fails to notice a significant change to the aircraft state, the PM notifies the PF.

Though CHAP-E initially has focused on assisting human pilots with plan execution via a display, we also have been exploring adding stronger automation characteristics to the system so it can handle both PF and PM tasks given a current plan. We plan to explore methods for the system to switch between PM and PF tasks. Currently, we can auto-

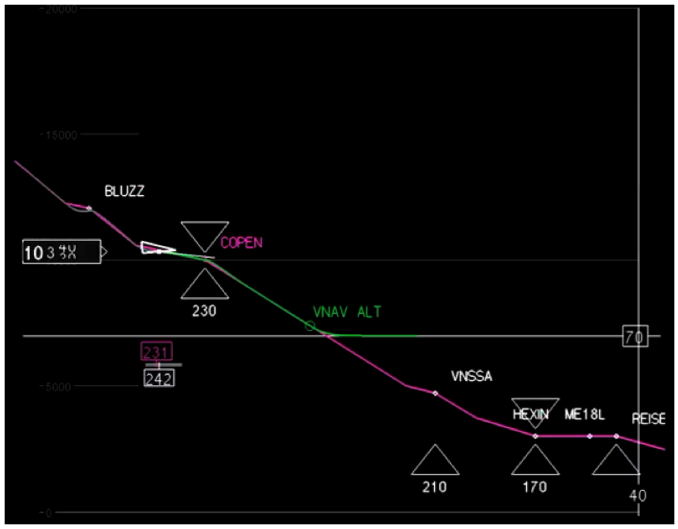

Figure 6: Visual depiction of plan execution simulation

mate flight completely by using the preferred start time as the time to execute actions. However, we require a mechanism for transferring flight control to CHAP-E for particular actions. This may happen automatically (for example, if the pilots appear to have missed performing a critical action) or upon request (for example, if the pilots become occupied and wish CHAP-E to perform specific actions). Our goal is to make the process intuitive to a pilot. We have begun considering using a touch-screen interface on CHAP-E to tap on actions that the pilots hope CHAP-E will perform, as well as methods of automatically determining safety-critical actions that may require CHAP-E's intervention when a pilot fails to perform them.

Acknowledgements

This work was supported by NASA Aeronautics's SASO/SECAT program.

References

Billings, C. E. 1996. Human-centered aviation automation: Principles and guidelines. Technical Report 110381, NASA Ames Research Center.

FAA. 2013. Operational use of flight path management systems.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Frank, J. D.; Spirkovska, L.; McCann, R.; Wang, L.; Pohlkamp, K.; and Morin, L. 2013. Autonomous mission operations. In *IEEE Aerospace Conference*.

Frank, J. D.; Iverson, D.; Knight, C.; Narasimhan, S.; Swanson, K.; Scott, M. S.; Pohlkamp, K. M.; Mauldin, J. M.; McGuire, K.; and Moses, H. 2015. Demonstrating autonomous mission operations onboard the international space station. In *AIAA SPACE 2015 Conference and Exposition*.

Kaneshige, J.; Benavides, J. V.; Sharma, S.; Panda, R.; and Steglinski, M. 2014. Implementation of a trajectory prediction function for trajectory based operations. In *AIAA Atmospheric Flight Mechanics Conference*.

Kortenkamp, D.; Bonasso, R. P.; Schreckenghost, D.; Dalal, K. M.; Verma, V.; and Wang, L. 2008. A procedure representation language for human space flight operations. In *9th International Symposium on Artifical Intelligence, Robotics and Automation for Space i-SAIRAS*.

MITRE. 2016. The solo pilot gets a second set of eyes. https://www.mitre.org/publications/project-stories/the-solo-pilot-gets-a-second-set-of-eyes.

Stetson, H. K.; Frank, J. D.; Haddock, A.; Cornelius, R.; Wang, L.; and Garner, L. 2015. AMO EXPRESS: A command and control experiment for crew autonomy. In *AIAA SPACE 2015 Conference and Exposition*.

Vadlamudi, S.; Chakraborti, T.; Zhang, Y.; and Kambhampati, S. 2016. Proactive decision support using automated planning. Technical report, Arizona State University. https://arxiv.org/abs/1606.07841.

Alternate Realities for Mission Operations Plan Execution

Pete Bonasso*, Dave Kortenkamp*, Blair MacIntyre†, Bryn Wolfe*

*TRACLabs, Inc., bonasso@traclabs.com, †Georgia Tech, blair@cc.gatech.edu

Abstract

Procedures are a mechanism by which NASA crewmembers execute plans. Alternate reality systems can help replace some of the guidance that ground controllers offer to crewmembers during procedure execution. As space exploration missions take crews further away from Earth, new forms of procedure assistance will be necessary. This paper describes an early development of an alternate reality (AR) system called PRIDE-AVR. PRIDE-AVR is an integration of the PRIDE electronic procedure development and execution system with augmented, virtual and hybrid reality technologies. We describe the system architecture and three proofs of concept demonstrations that use these AR technologies.

Motivation

Standard operating procedures are the mechanism by which plans are executed during typical spacecraft operations. Execution of procedures on the International Space Station (ISS) is currently heavily dependent upon ground controllers assisting crewmembers in performing planned operations and maintenance as well as with responses to off-nominal situations. This close collaboration becomes more difficult in exploration missions that take human crews beyond the easy reach of Mission Control, so crewmembers will need to have more autonomy from ground controllers. Alternate realities – augmented, virtual or hybrid – can help replace some of the guidance that ground controllers offer to crewmembers during procedure execution (Tang et al., 2003).

The context of the current work is authoring and executing NASA procedures that are then used for plan execution. The on-board short-term plans (OSTPs) for the International Space Station (ISS) are carried out by executing pre-written procedures. We have developed a procedure authoring and executing system called PRIDE (Izygon et al., 2008) that is currently used by NASA to produce machine-readable procedures.

In support of NASA, TRACLabs and Georgia Tech's Augmented Environmental Lab are working to integrate our PRIDE procedure development system (Izygon et al., 2008) with augmented, virtual and hybrid reality technologies in a system called PRIDE Augmented and Virtual Reality (PRIDE-AVR).

PRIDE-AVR

Augmented reality is a live direct or indirect view of a physical, real-world environment whose elements are supplemented by computer-generated sensory input such as sound, video, graphics or geospatial data. Virtual Reality is a realistic and immersive simulation of a three-dimensional environment, created using interactive software and hardware, and experienced or controlled by movement of the body[1]. Hybrid reality, sometimes known as mixed reality (de Souza e Silva, 2009), is the merging of real and virtual

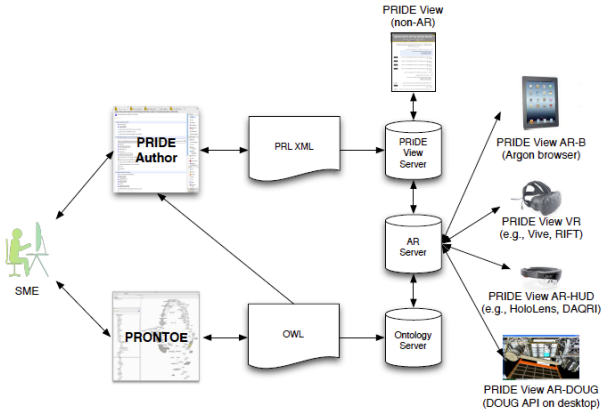

Figure 1 *The architecture of the PRIDE-AVR system.*

worlds to produce new environments and visualizations where physical and digital objects co-exist and interact in real time. With PRIDE-AVR we are investigating all three alternate realities in support of NASA procedure execution.

[1] http://www.dictionary.com/browse/virtual--reality

Our PRIDE-AVR design is shown in **Figure 1**. It builds on several existing components, including the PRIDE electronic procedure platform and a system ontology. The former includes a procedure authoring tool (PRIDE Author) and a server (PRIDE View Server) that shows procedures as web pages and supports crew member execution of procedures. The latter includes an ontology editor called PRONTOE (Bell et al., 2013) and an ontology server. Recently developed components are an alternate reality (AR) Server and new PRIDE View clients for the different alternate reality systems shown on the right of the figure.

The PRIDE viewer maintains the state of the executing procedure, that is, the current instruction and the success or failure of an instruction and/or of the procedure as a whole. As the user executes a procedure, the AR server accesses that information, tracks the progress and looks in the current instruction's PRL for references to objects in the domain ontology. The AR server queries the ontology server for any alternate reality attributes of those objects then sends the information to the AR system involved in the procedure execution.

NASA procedures can have conditional branching, instructions that are coordinated across multiple procedures, and instructions that invoke other procedures. For this preliminary work, we have used only linear procedures.

This paper describes the development of three proof-of-concept demonstrations using PRIDE-AVR in support of NASA missions.

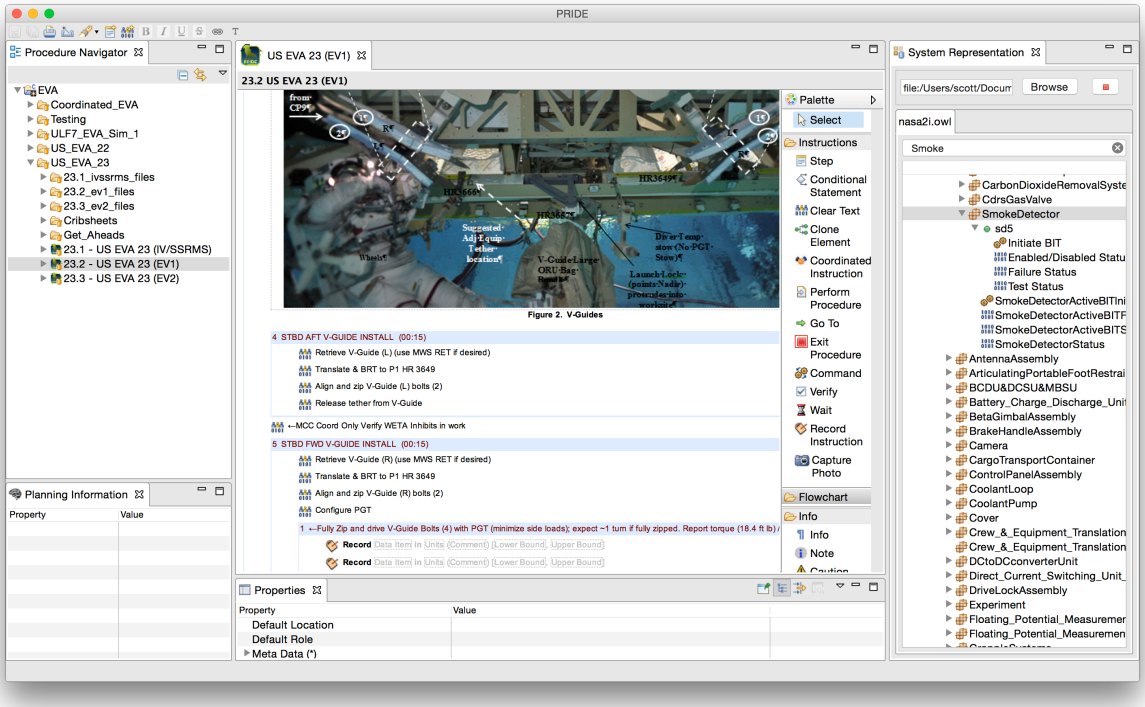

Figure 2 *A screenshot of an EVA procedure being developed in the PRIDE Author application.*

Preparing the Procedures for AR Support

Figure 2 shows the PRIDE Author interface. Users drag instruction types from the palette and drops them onto the center canvas, where they can edit the details. Users can also drag and drop entities from a domain ontology (Bonasso et al., 2013)(the System Representation pane in the figure) into an instruction where their URIs are embedded in the resulting Procedure Representation Language (PRL) XML file (Kortenkamp et al., 2008). Any of these

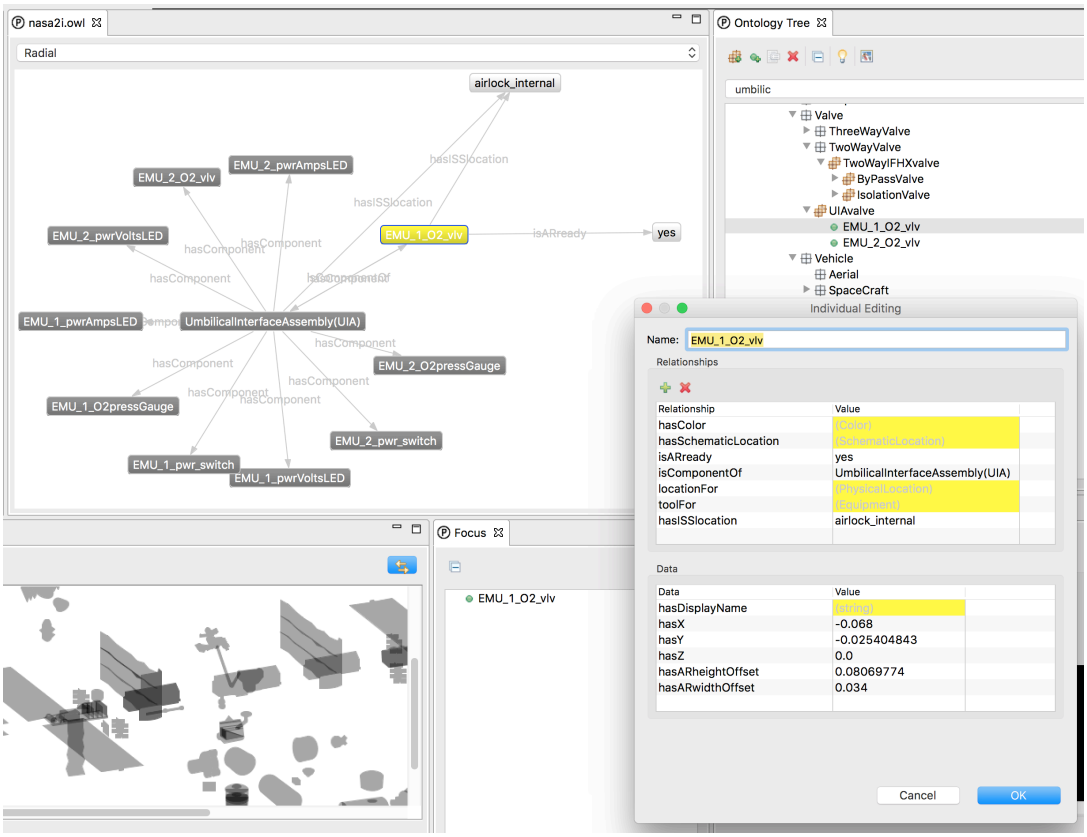

Figure 3 *A screenshot of PRONTOE showing a valve with its AR display offsets.*

entities can be AR objects, that is, they have properties germane to viewing in an AR system (e.g., see **Figure 3**).

Integrating Procedure Execution with Alternate Realities

As a procedure is executed, the AR server monitors its progress via a REpresentational State Transform (REST) interface (Fielding, 2000) to the PRIDE View server. If it detects the URI of any entity in the ontology, it queries the ontology server for that entity's properties, and if there are AR properties, it communicates them along with the instruction text to the AR system being used. For example, **Figure 3** shows coordinate offsets for a valve switch on a control panel. **Figure 4** shows that offset being used to draw a rectangle around the appropriate switch on the control panel.

Three Demonstrations[2]

The critical function of the AR server is to decide on an appropriate AR cue to send to the AR viewers. We developed three demonstrations that make use of the following AR cues:

Figure 4 *An iPad view of a control panel with the oxygen valve connected to EMU (spacesuit) 1.*

[2] Go to https://traclabs.com/projects/alternate-realities/ to see videos of these demonstrations.

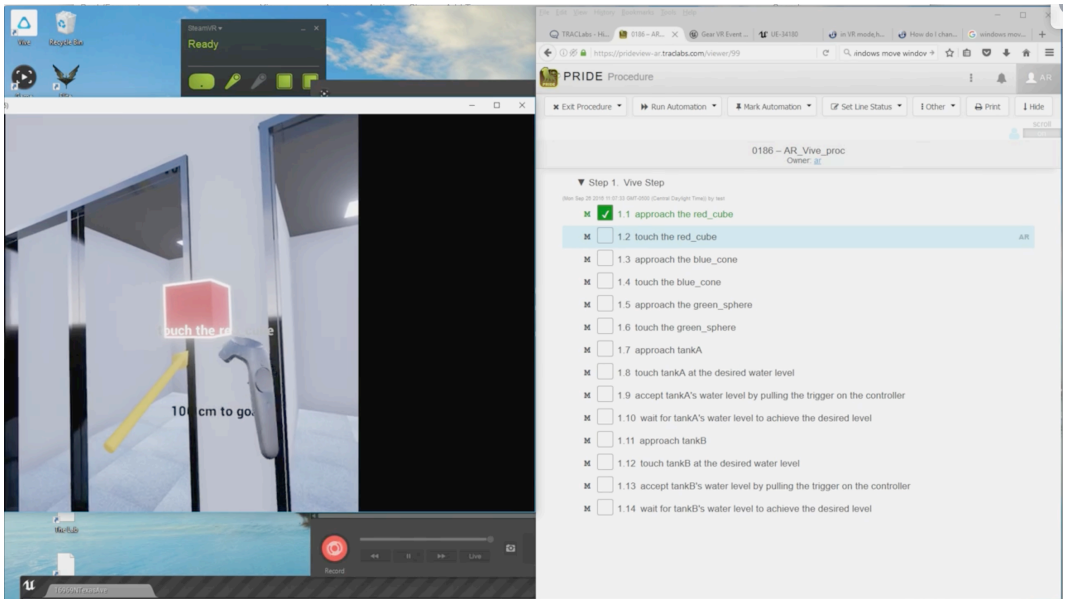

Figure 5 *View from HTC Vive headset alongside an external view of the executing procedure.*

• Show: highlight an object (outline, glow, flash, etc.), or provide additional details (e.g., wiring diagrams) as insets or overlays
• Instruct: Place a text instruction in a specific place in the field of view or when the user looks at a specific location
• Locate/Find: Show an object's location, for example, by displaying an arrow pointing to the object if it is in the field of view or pointing in the object's direction if it is not
• Data: show (live) data related to an object near that object or near an instruction referencing that object, for example, when a piece of telemetry needs to be verified in a procedure.

Each of these demonstrations uses the same PRIDE-AVR architecture as shown in **Figure 1**. The only changes are to the individual procedures, the ontology, and the hardware output device. It is important to remember that the procedure author does not need to do anything special to create augmented and virtual reality procedures. They simply drag objects from the ontology into the procedure and the AR server automatically turns them into AR cues. Moreover, PRONTOE ensures that the ontology can also be maintained by subject matter experts who need no programming experience. Thus, PRIDE-AVR allows flight controllers and other experts to create and change alternate reality systems with no coding or knowledge of those systems.

Augmented Reality Browser

We used PRIDE Author to create a portion of the Extravehicular Mobility Unit (EMU), or spacesuit, checkout procedure. This procedure has a significant number of instructions that refer to different components of the EMU and of the Umbilical Interface Assembly (UIA) to which it is connected. These components were modeled in the ontology (**Figure 3**). We printed a 2x3 foot image of the UIA and used it to create a Vuforia1 Image Target[3]. We then used the Argon browser (MacIntyre et al., 2011) and JavaScript framework developed at Georgia Tech's Augmented Environments Lab to create a web server that combined the

Figure 6 *View of the Real-world Fluid Transfer System (FTS)*

Vuforia image tracking technology with the data coming from the AR server to provide an augmented reality view of the EMU checkout procedure in a browser running on an iPad (**Figure 4**).

[3] www.vuforia.com

PTC's Vuforia image tracking software (http://www.ptc.com/en/about/history/vuforia), embedded in the Argon4 browser, has a 2D coordinate system for the photograph. The position and size of the instruments on the panel, in the coordinate system of the image used for tracking, were stored in our ontology and passed by the AR server to the Argon4 web application as parameters to the augmentation http command. These values are entirely dependent on the system used for tracking the position and orientation of the display device relative to the control panel. The next version of our implementation will be to express these locations relative to the parent component (e.g., Umbilical Interface Assembly (UIA)), so that when we are tracking the location of the component relative to the display, we can render the augmentations appropriately in 3D, just as we are doing in the VR system.

If the UIA (or a portion of it) is in the camera field of view as reported by Vuforia, then the current procedure instruction is displayed at the bottom of the live camera image and any UIA component that is referenced in that instruction is outlined. A Done button is also displayed. When the Done button is displayed and clicked on by the user, a step-completed message is passed to the AR server, which instructs the PRIDE View server to automatically advance to the next instruction and the process repeats.

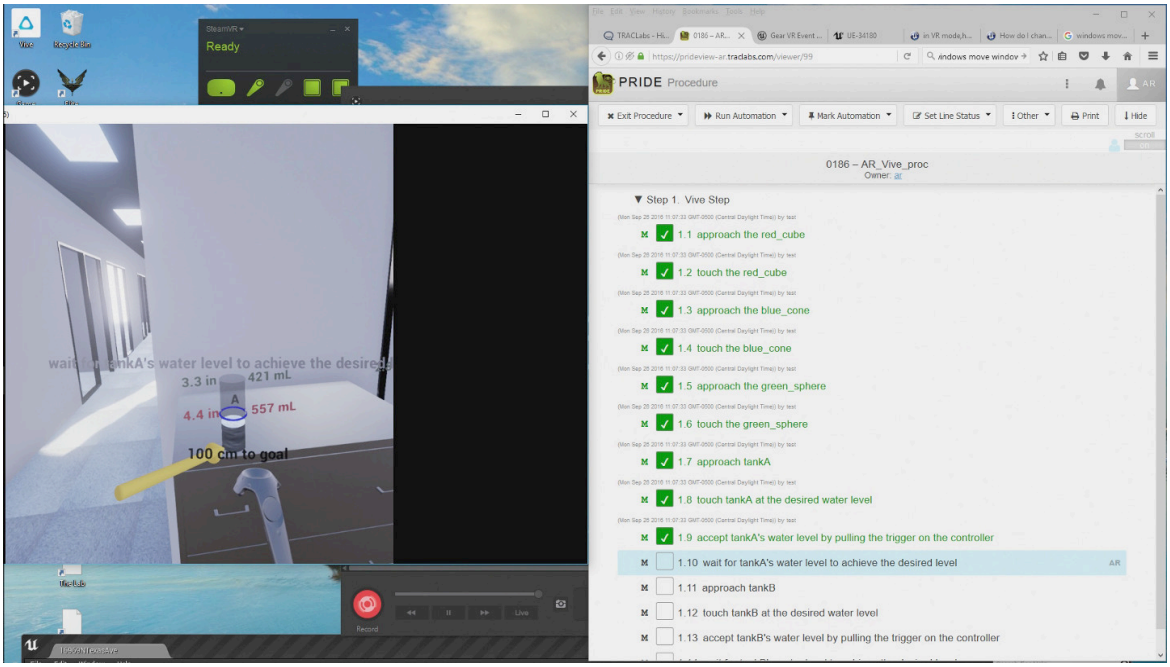

Figure 7 *View of the Virtual World Fluid Transfer System and the attendant procedure.*

Virtual/Hybrid Reality Demonstration

Our VR demo test bed consists of an HTC Vive connected to a computer in an open area of our facility. The Vive is a stereoscopic immersion platform that provides both visual and auditory information to the user, with hand controllers for interacting with the virtual environment. Multiple illuminators positioned around the open area paint patterned light on the user's headset and hand controllers, which then interpret those light patterns to extract location and orientation. That information updates the virtual environment, allowing free exploration of the virtual world within the confines of the Vive system's real-world arena.

Our VR proof of concept demonstration illustrates many of the concepts necessary to integrate electronic procedures into a VR display system. As before, the AR server tracks the executing procedure and queries the ontology server for AR properties of objects contained in the instruction. This information is presented to the VR system through a JSON RESTful interface, supplying current instruction information and accepting "instruction complete" commands for advancing to the next instruction. The VR rendering system parses current instruction information and renders guidance cues to the user in the form of textual instructions, visual guidance indicators, and spatial audio cues as directed by the AR server.

We implemented a procedure that walks the user through a set of manual tasks such as approaching and

touching various 3D virtual objects in the virtual environment (see **Figure 5**).

The TRACLabs Vive test bed also includes real-world devices with which the user can interact, thus implementing a form of hybrid reality. The second half of the demo has the user interacting with a real-world fluid transfer system (FTS) consisting of two cylindrical glass tanks with fluid level sensors, two fluid pumps, and a controller (see **Figure 6**). The FTS monitors fluid levels and accepts commands to move water from one tank to the other. The FTS provides a JSON RESTful interface for querying and commanding tank levels for both tanks. In the virtual environment, the tanks are modeled to mimic their physical characteristics in both appearance and location (see **Figure 7**). Fluid levels are reproduced in the virtual environment and the user can command new fluid levels by interacting with the virtual model.

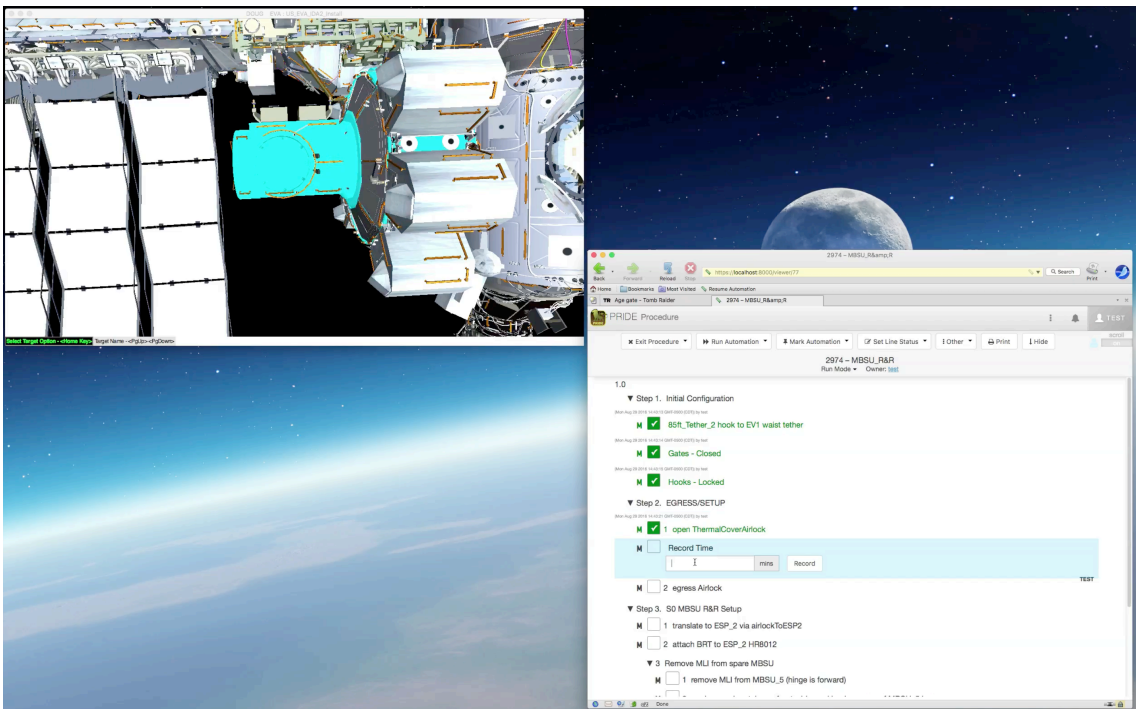

Figure 8 *View of an executing EVA procedure and the resulting DOUG display with the referenced ISS object highlighted in blue.*

AR-DOUG

In the third demonstration, the PRIDE-AVR system was used to drive NASA's Dynamic On-board Ubiquitous Graphics (DOUG) system[4], which is used to train astronauts for Extravehicular Activities (EVAs). Again, the AR server monitors a procedure executing in PRIDE View and parses the current instruction for references to objects in the ontology. For any references found, it queries the ontology to obtain DOUG info, such as the ISS location and the DOUG name. If it finds such information it commands DOUG to "fly to" the referenced object, and flashes the object (see **Figure 8**). We also added crew translation paths to the ontology, so, if the instruction being executed describes a translation action and references a translation path, the AR server will instruct DOUG to highlight the handrails and other hand holds as DOUG flies the view camera along the path.

Next Steps

Our work thus far shows that our PRIDE_AVR system architecture is feasible enough to support all three kinds of alternate reality viewers. Our next steps are to extend and enhance various components of the system to be able to address a larger set of requirements. For example, we will extend the AR server to reason over additional cues, such as audio cues, the use of countdown timers, and using additional media such as short movies or animations.

Our current system had a viewer plug-in for the HTC Vive using the Unreal engine and development environment. We will continue to expand that viewer plug-in with a goal of deploying it in the NASA Hybrid Reality Labora-

[4] https://vrlab.jsc.nasa.gov/

tory (HRL) HTC Vive environment. Used for inexpensive training of astronauts, the HRL has 3D models of the ISS (both interior and exterior) and allows users to move through the ISS and interact with objects in it. We are developing a tutorial procedure that guides the user through the various types of interactions in the HRL.

We will also extend our plug-in suite to include the Microsoft HoloLens augmented reality system.

References

Bell, S., Bonasso, R. P., Boddy, M., and Kortenkamp, D. 2013. PRONTOE: A case study for developing ontologies for operations. In *5th International Conference on Knowledge Engineering and Ontology Development (KEOD 13)*. Algarve, Portugal.

Bonasso, R. P., Kortenkamp, D., Boddy, M., and Bell, S. 2013. Ontological Models To Support Planning Operations. In *Proceedings of Internation Workshop on Planning and Scheduling in Space (IWPSS 13)*. Mountain View, CA.

de Souza e Silva, A. S., Daniel M. 2009. *Digital Cityscapes: merging digital and urban playspaces*. New York: Peter Lang Publishing, Inc.

Fielding, R. T. 2000. Architectural Styles and the Design of Network-based Software Architectures (chapter 5: Representational State Transfers). Computer Science, UC Ervine.

Izygon, M., Kortenkamp, D., and Molin, A. 2008. A procedure integrated development environment for future spacecraft and habitats. In *Space Technology and Applications International Forum (STAIF)*, vol. 969. Albuquerque, NM: American Instutute of Physics.

Kortenkamp, D., Bonasso, R. P., and Schreckenghost, D. 2008. A Procedure Representation Language for Human Spaceflight Operations. In *The 9th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS-08)*. Los Angeles, CA.

MacIntyre, B., Hill, A., Rouzati, H., Gandy, M., and Davidson, B. 2011. The argon ar web browser and standards-based ar application environment. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. Basel Basel, Switzerland: IEEE.

Tang, A., Owen, C., Biocca, F., and Mou, W. 2003. Comparative effectiveness of augmented
reality in object assembly. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. Fort Lauderdale, FL: ACM, 73-80.

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.

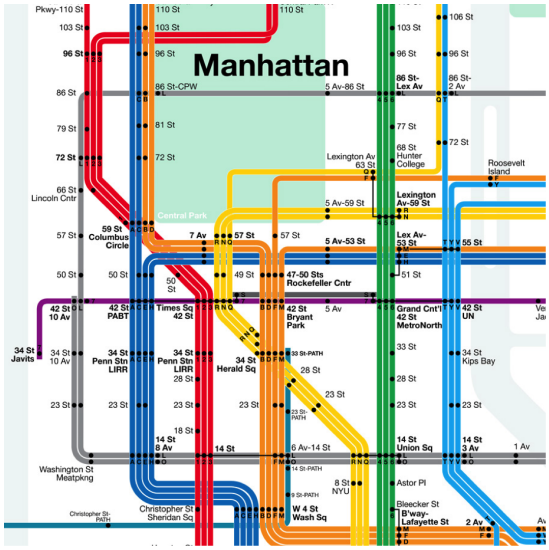

Figure 1: Manhattan Metro Map

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.

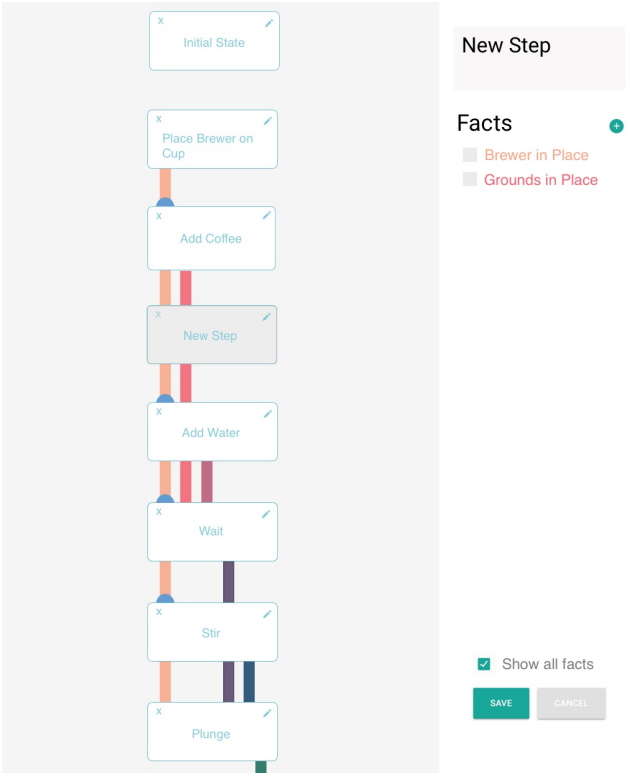

Figure 5: Users add steps to Conductor and can edit their details. The details panel allows users to change the step name, and the fact routes impacting or impacted by the step.

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-

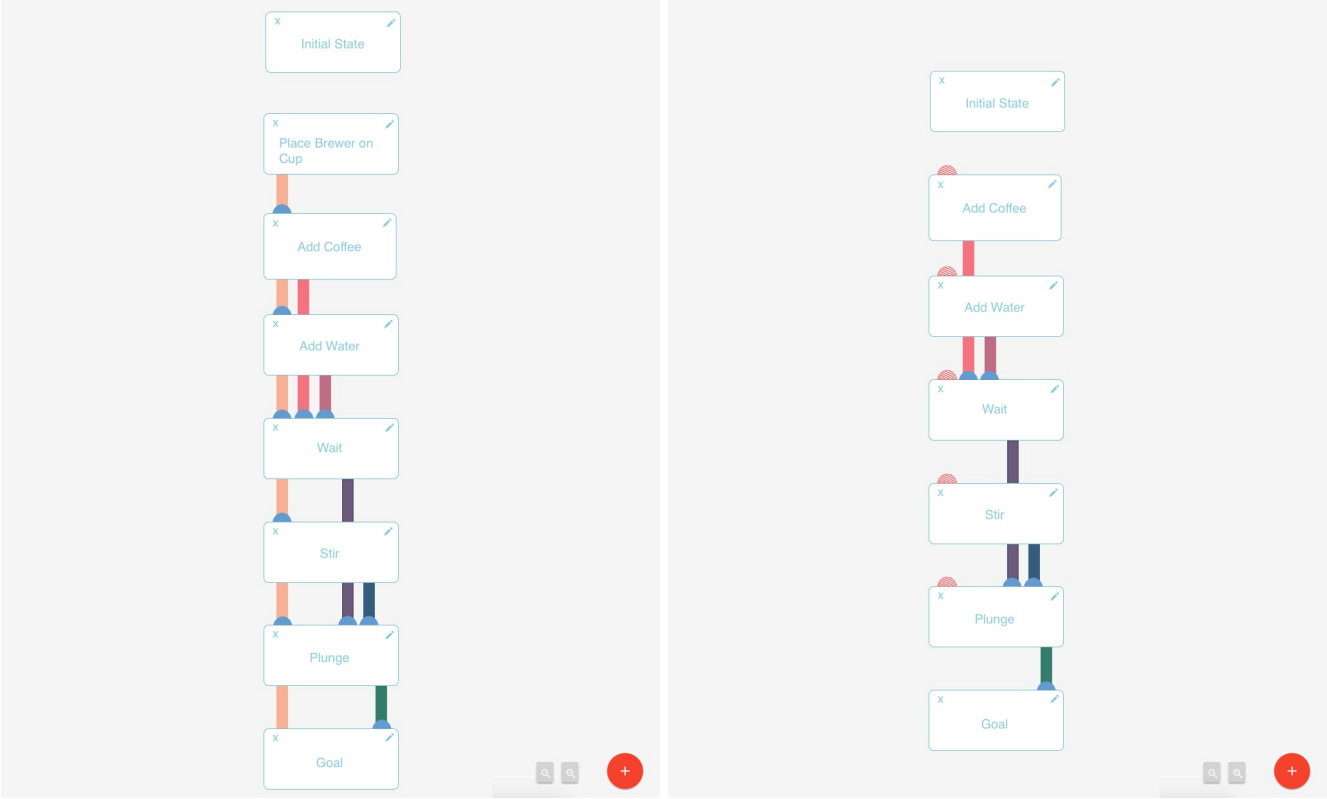

Figure 6: Users can remove steps in Conductor, which can disrupt the fact routes. The left illustrates the procedure before removing the "Place Brewer on Cup" step and the right illustrates after. Marshal computes the impact on the fact routes and marks any open conditions in red.

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

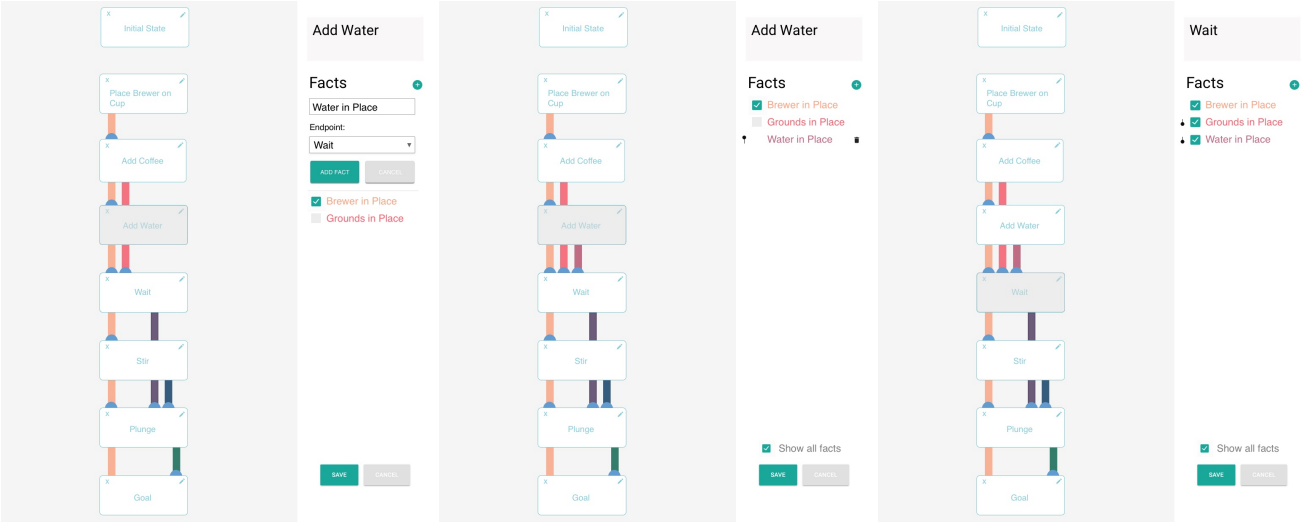

Figure 7: Users can add fact routes of the form $a_i \xrightarrow{p:(a_k,\ldots,a_l)} a_j$ in Conductor by adding a fact to a step (left, before, and center, after), and then terminating the route and adding stations (right).

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

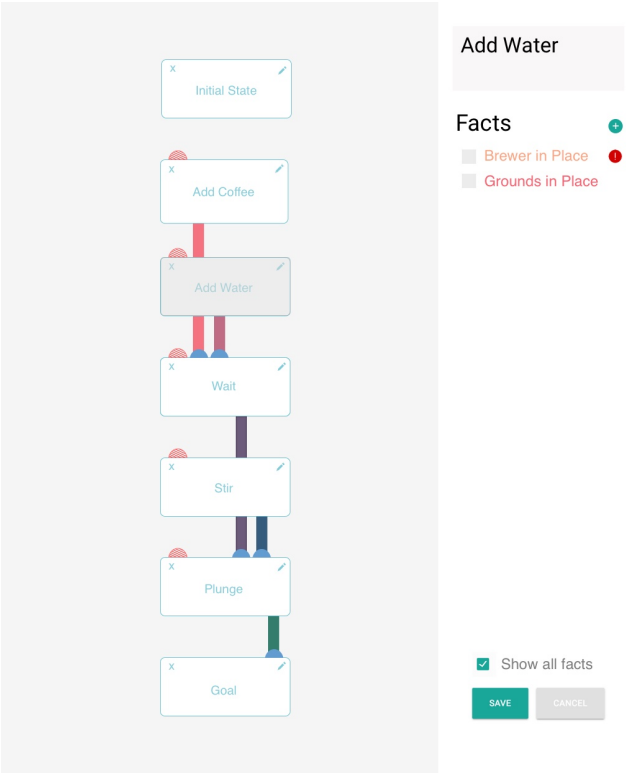

Figure 8: Users can address open conditions identified by Marshal in two ways. The user either establishes the conditions by adding fact routes, or clicks the red "!" button next to the fact to acknowledge or dismiss the open condition.

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

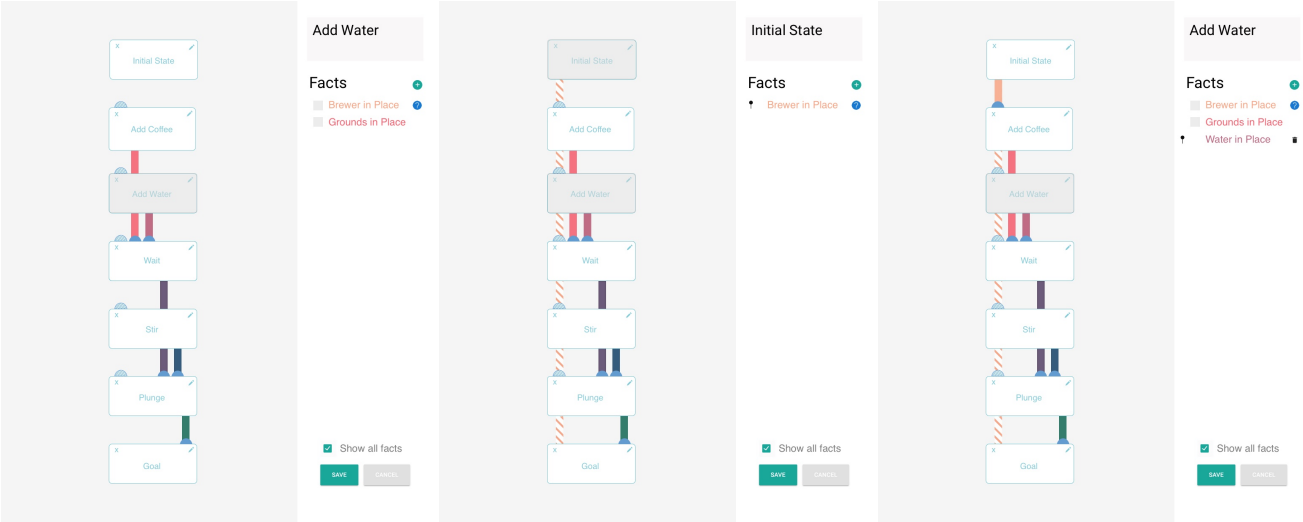

Figure 9: Users can address possible domain model features identified by Marshal similar to addressing open conditions. Users can modify fact routes to match Marshal's suggested updates, or dismiss them.

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.

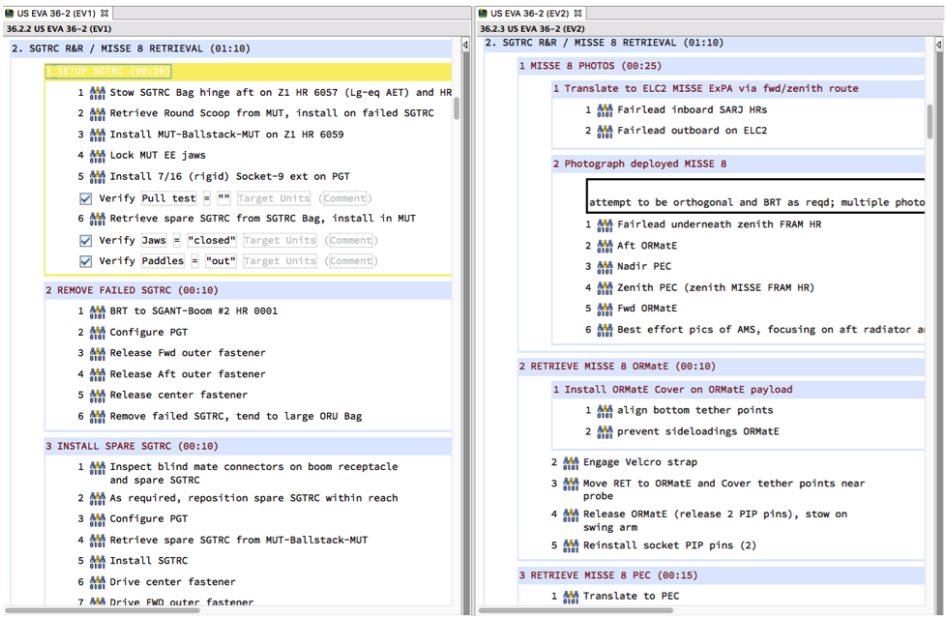

Figure 10: EVA 22 procedure for EV1 represented in PRIDE.

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.

Augmented Workspace for Human-in-the-Loop Plan Execution*

Tathagata Chakraborti and **Sarath Sreedharan** and **Anagha Kulkarni** and **Subbarao Kambhampati**

School of Computing, Informatics, and Decision Systems Engineering
Arizona State University, Tempe AZ 85281 USA

{ tchakra2, ssreedh3, akulka16, rao } @ asu.edu

Abstract

Ambiguity and noise in natural language instructions create a significant barrier towards adopting autonomous systems into safety critical workflows involving humans and machines. In this paper, we build on recent advances in electrophysiological monitoring methods and augmented reality technologies, to develop alternative modes of communication between humans and robots involved in large-scale proximal collaborative tasks. We will first introduce augmented reality techniques for projecting a robot's intentions to its human teammate, who can interact with these cues to engage in real-time collaborative plan execution with the robot. We will then look at how electroencephalographic (EEG) feedback can be used to monitor human response to both discrete events, as well as longer term affective states while execution of a plan. These signals can be used by a learning agent, a.k.a an affective robot, to modify its policy. We will present an end-to-end system capable of demonstrating these modalities of interaction. We hope that the proposed system will inspire research in augmenting human-robot interactions with alternative forms of communications in the interests of safety, productivity, and fluency of teaming, particularly in engineered settings such as the factory floor or the assembly line in the manufacturing industry where use of such wearables can be enforced.

Introduction

Effective planning for human robot teams not only involve the capacity to be "human-aware" during the plan generation process, but also require the ability to interact with the human during the plan execution phase, as well as collect data during it so as to inform the decision making process, either immediately or over time, of a learning agent. Prior work has underlined this need (Karpas et al. 2015) as well as explored ways to exchange (Tellex et al. 2014) information in natural language during interaction with the human in the loop. However, the state of the art in natural language considerably limits the scope of such interactions, especially where precise instructions are required. Moreover, prior work in learning appropriate models for human awareness largely rely on labeling phases (Zhang et al. 2017) that are quite unrealistic for large scale data collection. In this paper, we will show how natural language and labeling techniques can be replaced, especially in settings such as the manufacturing industry, with the help of wearable technologies (such as the HoloLens and the Emotiv Epoc+ EEG headset) for effective interaction during human-in-the-loop operation of robots.

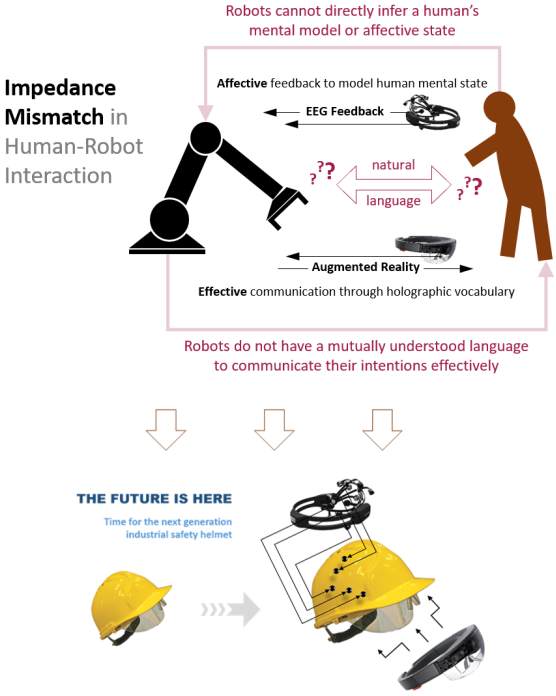

Figure 1: Alternative forms of communication to combat impedance mismatch in human robot interactions in settings where wearables can be integrated for closed loop feedback from EEG signals and augmented reality.

Indeed, the last decade has seen a massive increase in robots deployed on the factory floor (Robotenomics 2017). This has led to fears of massive loss of jobs for humans in the manufacturing industry, as well concerns of safety for the jobs that do remain. The latter is not an emerging concern, though. Automation of the manufacturing industry has gone hand in hand with incidents of misaligned intentions between the robots and their humans co-workers, leading to

*The work is part of *Project Cloudy with a Chance of Synergy*, from team ÆRobotics which appeared in the US Finals of the Microsoft Imagine Cup 2017. http://www.ae-robots.com/

at least four instances of fatality (Weiss 2015). This dates back to as early as 1979 when a robot arm crushed a worker to death while gathering supplies in the Michigan Ford Motor Factory, to as recent as 2015 in a very similar and much publicized accident in the Volkswagen factory in Baunatal, Germany. With 1.3 million new robots predicted to enter the workspace by next year (PRNewswire 2016), such concerns are only expected to escalate.

A closer look at the dynamics of employment in the manufacturing industry also reveals that the introduction of automation has in fact increased productivity (Muro and Andes 2015) as well as, surprisingly, contributed to a steady increase in the number of jobs for human workers (Look 2016) in Germany (which so far dominates in terms of deployed robots in the industry). We posit then either a semi-autonomous workspace in future with increased hazards due to misaligned interests of robots in the shared environment, or a future where the interests of the human workers will be compromised in favor of automation. In light of this, it is essential that the next-generation factory floor is able to cope with the needs of these new technologies.

At the core of this problem is the impedance mismatch between humans and robots in how they represent and communicate information, as illustrated in Figure 1. Despite the progress made in natural language processing, natural language understanding is still a largely unsolved problem, and as such robots find it difficult to **(1)** express their own goals and intentions effectively; as well as **(2)** understand human expressions and emotions. Thus there exists a significant communication barrier to be overcome from either side, and robots are essentially still "autistic" (Kaminka 2013) in many aspects. While this may not always be a serious concern for deploying completely autonomous agents in isolated environments such as for space or underwater exploration, the priorities change considerably when humans and robots are involved in collaborative tasks, especially for concerns of safety, if not to just improve the effectiveness of collaboration. This is emphasized in the *Roadmap for U.S. Robotics* (Christensen et al. 2009) which outlines that *"humans must be able to read and recognize robot activities in order to interpret the robot's understanding"*. Recent work has focused on generation of legible motion plans (Dragan et al. 2015) and explicable task plans (Zhang et al. 2017), and verbalization of intentions in natural language (Tellex et al. 2014; Perera et al. 2016).

The Manufacturing Environment Our primary focus here is on structured settings like the manufacturing environment where wearables can be a viable solution for improving the workspace. Indeed, a reboot of the safety helmet and goggles as illustrated in Figure 1 only requires retro-fitting existing wearables with sensors that can enable these new technologies. Imagine, then, a human and robot engaged in an assembly task, where they are constructing a structure collaboratively. Further suppose that the human now needs a tool from the shared workspace. At this time, neither agent is sure what tools and objects the other is going to access in the immediate future - this calls for seamless transfer of relevant information without loss of workflow. Existing (general purpose) solutions will suggest intention recognition (Hayes and Scassellati 2016) or natural language (Tellex et al. 2014) communication as a means to respond to this situation. With regards to naturalistic modes of interaction among agents, while natural language and intent or gesture recognition techniques remain the ideal choice in most cases, and perhaps the only choice in some (such as robots that would interact with people in their daily lives), we note that these are inherently noisy and ambiguous, and not necessary in controlled environments such as on the factory floor or by the assembly line where the workspace can be engineered to enforce protocols in the interests of safety and productivity, in the form of safety helmets integrated with wearable technology (Ruffaldi et al. 2016).

Instead, in our system, the robot projects its intentions as *holograms* thus making it directly accessible to the human in the loop, e.g. by projecting a pickup symbol on a tool it might use in future. Further, unlike in traditional mixed reality projection systems, the human can directly interact with these holograms to make his own intentions known to the robot, e.g. by gazing at and selecting the desired tool thus forcing the robot to replan. To this end, we develop, with the power of the HoloLens[1], an alternative communication paradigm that is based on the projection of explicit visual cues pertaining to the plan under execution via holograms such that they can be intuitively understood and directly read by the human partner. The "real" shared human-robot workspace is now thus augmented with the virtual space where the physical environment is used as a medium to convey information about the intended actions of the robot, the safety of the work space, or task-related instructions. We call this the *Augmented Workspace*. Recent development of augmented reality techniques (Sean O'Kane 2015) has opened up endless possibilities in such modes of communication.

This, by itself, however, provides little indication of the mental state of the human, i.e. how he is actually responding to the interactions - something that human teammates naturally keep track of during a collaborative exercise. In our system, we propose to use real-time EEG feedback using the Emotiv EPOC+ headset[2]this purpose. This has several advantages - specific signals in the brain are understood to have known semantics (more on this later), and are detected immediately and with high accuracy, thus short circuiting the need for the relatively highly inaccurate and slower signal processing stage in rivaling techniques such as emotion and gesture recognition. Going back to our previous use case, if the robot now makes an attempt to pick up the same tool again, the error can fire an event related EEG response - which may readily be used as in a closed loop feedback to control or stop the robot. Further, if the robot is making the same mistake again and again, causing the human to be stressed and/or irritated, it can listen to the human's affective states to learn better, and more human-aware, policies over time. We demonstrate these capabilities as part of the *Consciousness Cloud* which provides the robots real-time shared access to the mental state of all the humans in the workspace. The agents are thus able to query the cloud about particulars (e.g. stress levels) of the current mental state, or receive specific alerts related to the human's response to events (e.g.

oddball incidents like safety hazards and corresponding ERP spikes) in the environment.

Finally, instead of the single human and robot collaborating over an assembly task, imagine now an entire workspace shared by many such agents, as is the case of most manufacturing environments. Traditional notions of communication become intractable in such settings. With this in mind, we make the entire system cloud based - all the agents log their respective states on to a central serve, and can also access the state of their co-workers from it. As opposed to peer-to-peer information sharing, this approach provides a distinct advantage towards making the system scalable to multiple agents, both humans and robots, sharing and collaborating in the same workspace, as envisioned in Figure 3.

Contributions Thus, in this paper, we propose approaches to tear down the communication barrier between human and robot team members **(1)** by means of holograms/projections as part of a shared alternative vocabulary for communication in the **Augmented Workspace**, and **(2)** by using direct feedback from physiological signals to model the human mental state in the shared **Consciousness Cloud**. The former allows for real-time interactive plan monitoring and execution of the robot with a human-in-the-loop, while the latter, in addition to passive plan monitoring, also allows a planning agent to learn preferences of its human co-worker and update its policies accordingly. We will demonstrate how this can be achieved on an end-to-end cloud-based platform built specifically to scale up to the demands of the next-generation semi-autonomous workspace envisioned in Figure 3.

Related Work

Intention Projection and Mixed Reality The concept of intention projection for autonomous systems have been explored before. An early attempt was made by (Sato and Sakane 2000) in their prototype Interactive Hand Pointer (IHP) to control a robot in the human's workspace. Similar systems have since been developed to visualize trajectories of mobile wheelchairs and robots (Watanabe et al. 2015; Chadalavada et al. 2015), which suggest that humans prefer to interact with a robot when it presents its intentions directly as visual cues. The last few years have seen active research (Omidshafiei et al. 2015; 2016; Shen, Jin, and Gans 2013; Ishii et al. 2009; Mistry et al. 2010; Leutert, Herrmann, and Schilling 2013; Turk and Fragoso 2015; Maurtua et al. 2016) in this area, but most of these systems were passive, non-interactive and quite limited in their scope, and did not consider the state of the objects or the context of the plan pertaining to the action while projecting information. As such, the scope of intention projection has remained largely limited. Instead, in this paper, we demonstrate a system that is able to provide much richer information to the human-in-the-loop during collaborative plan execution, in terms of the current state information, action being performed as well as future parts of the plan under execution. We also demonstrate how recent advances in the field of augmented reality make this form of online interactive plan execution particularly compelling. In Table 1 we provide the relative merits of augmented reality with the state-of-the-art in mixed reality projections.

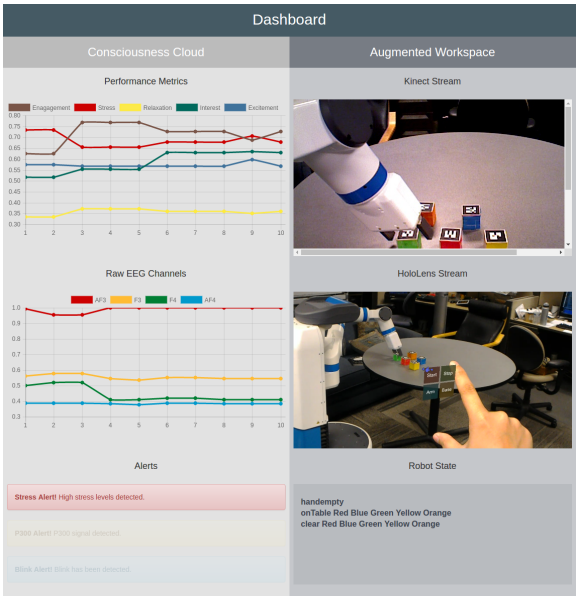

Figure 2: The Dashboard - displaying elements of the Consciousness Cloud and the Augmented Workspace - monitors the state of the shared workspace.

EEG Feedback and Robotics Electroencephalography (EEG) is an electrophysiological monitoring method to measure voltage fluctuations resulting from ionic currents within the brain. The use of EEG signals in the design of BCI has been of considerable interest in recent times. The aim of our project is to integrate EEG-based feedback in human-robot interaction or HRI. Of particular interest to us are Event Related Potentials or ERPs which are measured due the response to specific sensory, cognitive, or motor events, and may be especially useful in gauging the human reaction to specific actions during the execution of a robot's plan (Hoffmann et al. 2008; Acqualagna et al. 2010; Guger et al. 2012; Ramli et al. 2015). Recently, researchers have tried to improve performance in robotics tasks by applying error-related potentials or ErrPs (Rao 2013; Ferrez and Millán 2008) to a reinforcement learning process (Iturrate, Montesano, and Minguez 2010; Iturrate et al. 2012). These are error signals produced due to undesired or unexpected effects after performing an action. The existence of ErrPs and the possibility of classifying them in online settings has been studied in driving tasks (Zhang et al. 2015), as well as to change the robots immediate behavior (Salazar-Gomez et al. 2017). However, almost all of the focus has remained on the control of robots rather than as a means of learning behavior (Bi, Fan, and Liu 2013), and very little has been made of the effect of such signals on the task level interactions between agents. This remains the primary focus of our system.

[1]https://www.microsoft.com/microsoft-hololens/en-us
[2]https://www.emotiv.com/epoc/

System Overview

There are two major components of the system (refer to Figure 3) - (1) the **Augmented Workspace** which allows the robots to communicate with their human co-workers in the virtual space; and (2) the **Consciousness Cloud** which provides the robots real-time shared access to the mental state of all the humans in the workspace. This is visible in the centralized **Dashboard** that provides a real-time snapshot of the entire workspace, as seen in Figure 2. The Augmented Workspace Panel shows real-time stream from the robot's point of view, the augmented reality stream from the human's point of view and information about the current state of plan execution. The Consciousness Cloud Panel displays the real-time affective states (engagement, stress, relaxation, excitement and interest), raw EEG signals from the four channels (AF3, F3, AF4 and F4) used to detect response to discrete events, as well as alerts signifying abnormal conditions (p300, control blink, high stress, etc.). The Dashboard allows the humans to communicate or visualize the collaborative planning process between themselves. It can be especially useful in factory settings to the floor manager who can use it to effectively monitor the shared workspace.

The Augmented Workspace

In the augmented workspace (refer to Figure 4). the HoloLens communicates with the user endpoints through the `REST API server`. The API server is implemented in python using the `Flask` web server framework. All external traffic to the server is handled by an `Apache2` server that communicates with the python application through a `WSGI` middle layer. The `Apache2` server ensures that the server can easily support a large number of concurrent requests.

The `REST` service exposes both `GET` and `POST` endpoints. The `GET` links provides the HoloLens application with a way of accessing information from the robot, while the `POST` link provides the HoloLens application control over the robots operation. Currently, we are using the API to expose information like the robotic planning state, robot joint values and transforms to special markers in the environment. Most API GET calls will first try to fetch the requested information from the memcached layer, and would only try a direct query to the `MySQL` database if the cache entry is older than a specified limit. Each query to the database also causes the corresponding cache entry to be updated. The `MySQL` server is updated by a daemon that runs on `Azure` and keeps consuming messages sent from the robot through various queues implemented using the rabbitMQ service.

Modalities of Interaction

We will now demonstrate different ways augmented reality can improve the human-robot workspace, either by providing a platform for interactive plan execution for online collaboration, or as a means of providing assistive cues to guide the plan execution process. A video demonstrating all these capabilities is available at `https://goo.gl/pWWzJb`.

Interactive Plan Execution Perhaps the biggest use of AR techniques in the context of planning is for human-in-the-loop plan execution. For example, a robot involved in an assembly task can project the objects it is intending to manipulate into the human's point of view, and annotate them with holograms that correspond to intentions to use or pickup. The human can, in turn, access or claim a particular object in the virtual space and force the robot to re-plan, without there ever being any conflict of intentions in the real space. The humans in the loop can thus not only infer the robot's intent immediately from these holographic projections, but can also interact with them to communicate their own intentions directly and thereby modify the robot's behavior online. The robot can also then ask for help from the human, using these holograms. Figure 6 shows, in detail, one such use case in our favorite BlocksWorld domain.

The human can go into finer control of the robot by accessing the Holographic Control Panel, as seen in Figure 7(a). The panel provides the human controls to start and stop execution of the robot's plan, as well as achieve fine grained motion control of both the base and the arm by making it mimic he user's arm motion gestures on the MoveArm and MoveBase holograms attached to the robot.

Assistive Cues The use of AR is, of course, not just restricted to procedural execution of plans. It can also be used to annotate the workspace with artifacts derived from the current plan under execution in order to improve the fluency of collaboration. For example, Figure 7(b-e) shows the robot projecting its area of influence in its workspace either as a 3D sphere around it, or as a 2D circle on the area it is going to interact with. This is rendered dynamically in real-time based on the distance of the end effector to its center, and to the object to be manipulated. This can be very useful in determining safety zones around a robot in operation. As seen in Figure 7(f-i), the robot can also render hidden objects or partially observable state variables relevant to a plan, as well as indicators to improve peripheral vision of the human, to improve his/her situational awareness.

The Consciousness Cloud

The Consciousness Cloud has two components - the affective state monitor and the discrete event monitor (as shown in Figure 5). In the affective state monitoring system, metrics corresponding to affective signals recorded by the Emotiv EPOC+ headset are directly fed into a rabbitMQ queue, as before, called Raw Affective Queue to be used for visualization, and a reward signal (calculated from the metrics) is fed into the Reward Queue. The robot directly consumes the Reward Queue and the signals that appear during an action execution is considered as the action reward or environment feedback for the AI agent (implementing a reinforcement learning agent). For the discrete event monitoring system, the raw EEG signals from the brain are sampled and written to a rabbitMQ queue called EEG queue. This queue is being consumed by our Machine learning or classifier module, which is a python daemon running on a azure server. When this python daemon is spawned it trains an SVM classifier using a set of previously labelled EEG signals. The signals consumed from the queue are first passed through a feature extractor and then the extracted features are used by the SVM to detect specific events (e.g. blinks). For each

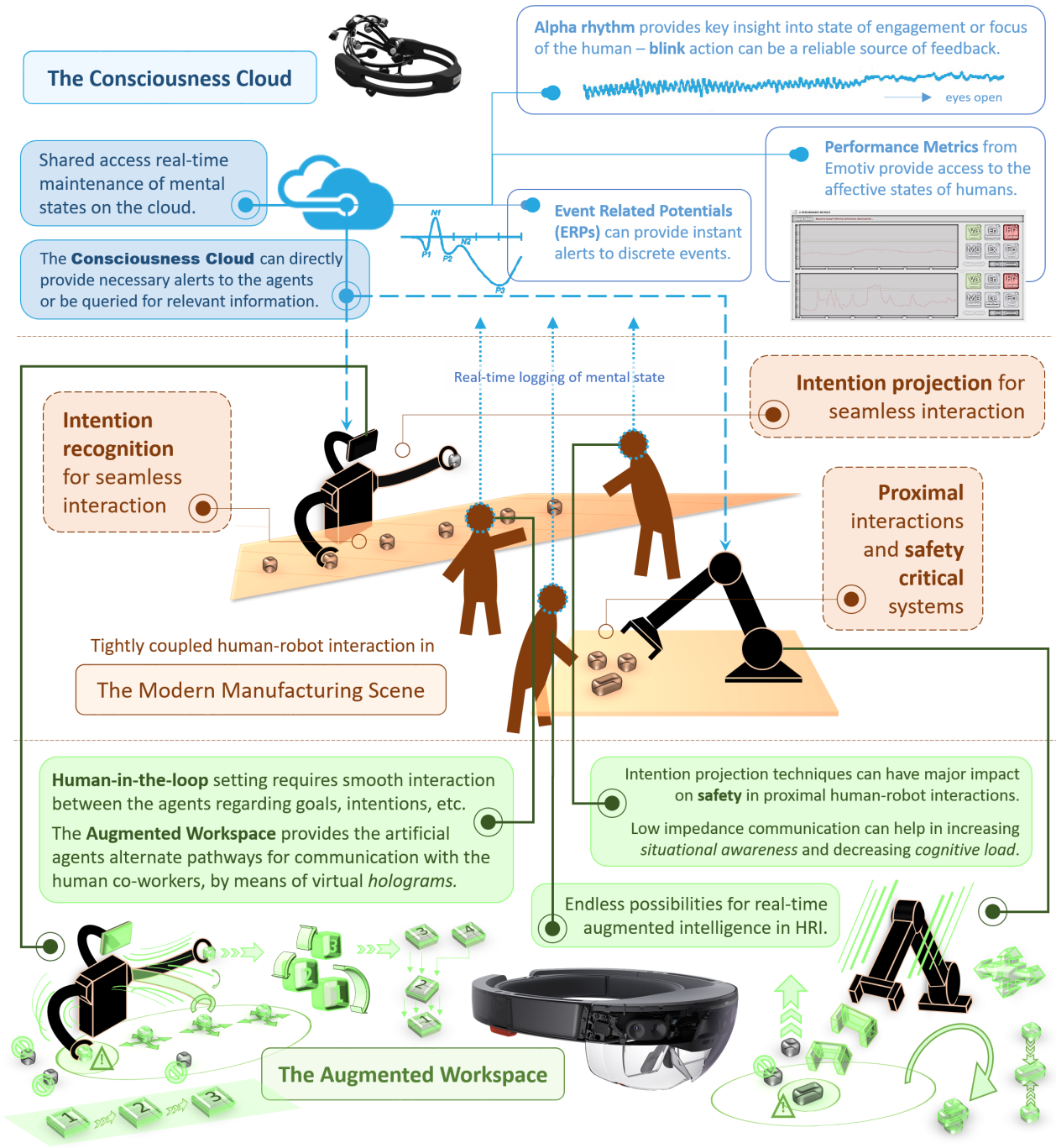

Figure 3: A conceptual impression of the next generation workshop floor involving multiple humans and robots sharing the workspace and collaborating either singly or in groups across different tasks. The humans are wearing safety helmets integrated with electrodes to capture EEG feedback, as well as HoloLens style safety goggles that provide access to augmented reality based communication through a shared-access cloud platform. This means that all the robots now have access to a real-time mental model of all their human co-workers on the cloud which they can use to inform or modulate their own behavior. The robots can also project their goals and intentions, as well as their private regions of interest, into their immediate environment, thereby the improving situational awareness of their human teammates. These two components - called the *Consciousness Cloud* and the *Augmented Workspace* - forms a sophisticated plan execution and plan monitoring system that can adapt while taking real-time feedback from humans-in-the-loop.

Property	AR	MR	Comments
Interaction	✓	✗	One of the key features of AR is that it provides the humans with the ability to interact directly, and effectively, with the holograms. This becomes particularly difficult in MR, especially due to difficulties in accurate gaze and gesture estimation.
Occlusion	?	✗	Unlike MR, AR is not particularly disadvantaged by occlusions due to objects or agents in the workspace. However, it does reduce the field of view significantly (though this is expected to improve with future iterations of the HoloLens).
Ergonomics	✗	✓	At present the size, weight and the occlusion of the peripheral view due to the HoloLens makes it somewhat unsuitable for longer operations, while the MR approach does not require any wearables and leaves the human mostly uninhibited. However, this is again expected to improve in later iterations of the HoloLens, as well as if they are custom made and optimized for a setting such as this.
Scalability	✓	?	MR will find it difficult to scale up to beyond peer-to-peer interactions or a confined space, given the requirement of viable projectors for every interaction. This is hardly an issue for the HoloLens which provides unrestricted mobility and portability of solutions.
Scope	✓	✗	MR is limited by a 2D canvas (environment), whereas AR can not only provide 3D projections that can be interacted with but also can express information that 2D projections cannot - e.g. a 3D volume of safety around the robot rather than just the projected area on the floor.

Table 1: Relative merits of Augmented Reality (AR) and Existing Mixed Reality (MR) approaches for intention projection.

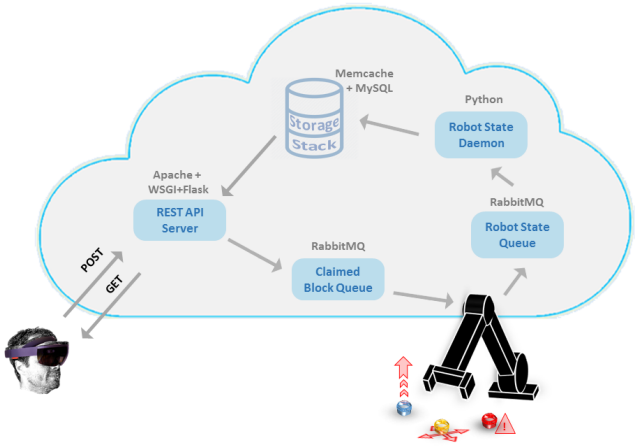

Figure 4: Architecture diagram of Augmented Workspace.

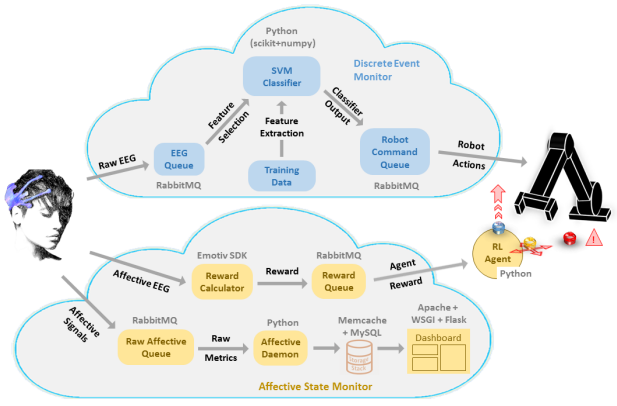

Figure 5: Architecture diagram of Consciousness Cloud.

event a corresponding command is sent to the Robot Command queue, which is consumed by the robot. For example, if a STOP command is sent for the blink event, it would cause the robot to halt its current operation.

Modalities of Interaction

Figure 8 demonstrates different ways in which EEG signals can be used to provide closed loop feedback to control the behavior of robots. This can be useful in two ways - either as a means of plan monitoring, i.e. controlling the plan execution process using immediate feedback, or as a reward signal for shaping and refining the policies of a learning agent. A video demonstrating these capabilities is available at `https://goo.gl/6LhKNZ`.

Discrete Events Discrete events refer to close to instantaneous events, producing certain typical (easy to classify) signals. We identify three modalities of EEG-based feedback - (1) Event Related Potentials or ERPs (e.g. p300) that can provide insight into the human's responses like surprise; (2) Affective States like stress, valence, anger, etc. that can provide longer term feedback on how the human evaluates interactions with the robot; and finally (3) Alpha Rhythm that can relate to factors such as task engagement and focus of the human teammate. This type of feedback is useful in the online monitoring of the plan execution process by providing immediate feedback on errors or mistakes made by the robot. The video demonstration shows a particular example when the human avoids coming into the harm's way by stops the robot's arm by blinking. Figure 8 shows another such use case where the robot is building words (chosen by the human) out of lettered blocks and makes a wrong choice of a letter at some stage - the mistake may be measured as a presence of ERP signal here. The latter has so far gotten mixed results leading us to shift to different EEG helmets (Emotiv Epoc+ lacks electrodes in the central area of the brain where p300s are known to be elicited) for better accuracy.

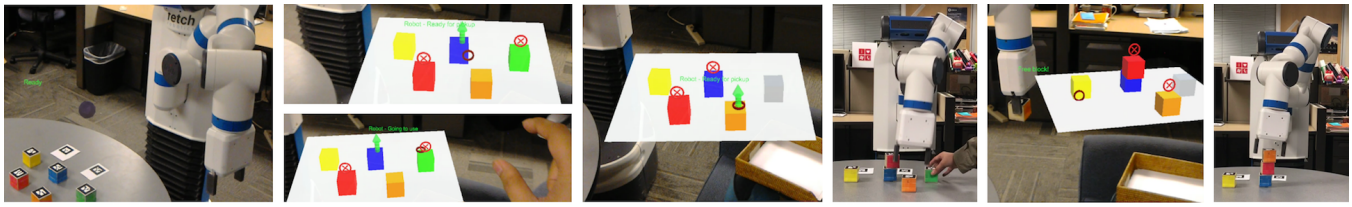

Figure 6: Interactive execution of a plan in the augmented workspace - **(a)** First person view of the workspace. The robot wants to build a tower of height three with blocks blue, red and green. **(b)** Block are annotated with intuitive holograms, e.g. an upward arrow on the block the robot is going to pick up immediately and a red cross mark on the ones it is planning to use later. The human can also gaze on an object for more information (in the rendered text). **(c)** & **(d)** The human pinches on the green block and claims it for himself. The robot now projects a faded out green block and re-plans online to use the orange block instead (as evident by pickup arrow that has shifted on the latter at this time). **(e)** Real-time update and rendering of the current state showing status of the plan and objects in the environment. **(f)** The robot completes its new plan using the orange block.

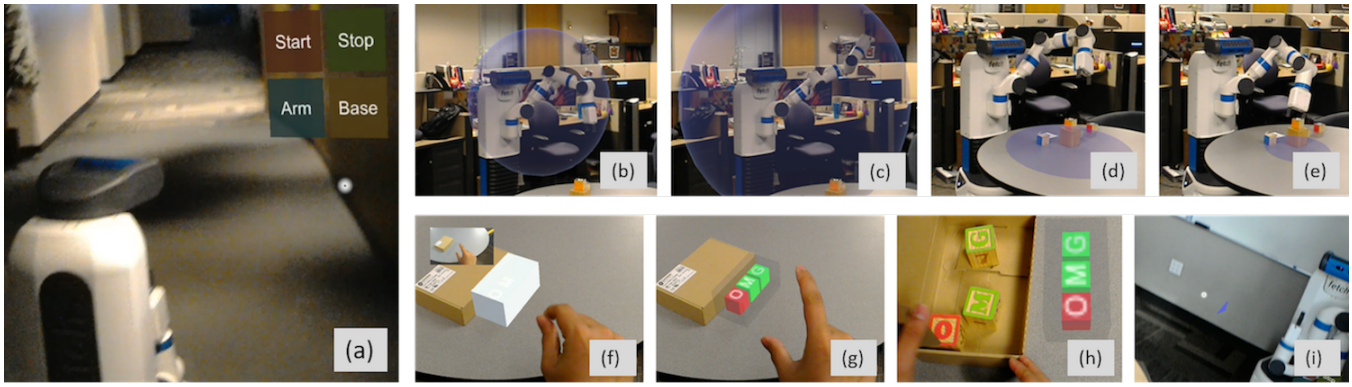

Figure 7: Interactive plan execution using the **(a)** Holographic Control Panel. Safety cues showing dynamic real-time rendering of volume of influence **(b) - (c)** or area of influence **(d) - (e)**, as well as **(i)** indicators for peripheral awareness. Interactive rendering of hidden objects **(f) - (h)** to improve observability and situational awareness in complex workspaces.

Affective States Here, our aim is to train a learning agent to model the preferences of its human teammate by listening to his/her emotions or affective states. We refer to this as *affective robotics* (analogous to the field of affective computing). As we mentioned before, the Emotiv SDK currently provides five performance metrics, namely valence/excitement, stress/frustration, engagement, attention, and meditation. At this time, we have limited ourselves to excitement and stress as our positive (R^{H+}) and negative reward signals (R^{H-}). We use a linear combination of these two metrics to create a feedback signal that captures the humans emotional response to a robots action. It is important to note that these signals do not capture the entire reward signal but only capture soft goals or preferences that the robot should satisfy, which means the total reward for the agent is given by $R = R^T + R^H$, where R^T is the reward for the original task. However, learning this from scratch becomes a hard (as well as somewhat unnecessary if the domain physics is already known) problem given the number of episodes this will require. Keeping this in mind, we adopt a two staged approach where the learning agent is first trained on the task in isolation without the human in the loop (i.e Q-learning with only R^T) so that it can learn a policy that solves the problem (π^T). Then we use this plan as the initial policy for a new Q-learning agent that considers the full rewards (R) with the human in the loop. This *"bootstrapping"* approach should reduce the training time.

The scenario, as seen in Figure 8, involves a workspace that is shared by a robot and a human. The workspace consists of a table with six multicolored blocks. The robot is expected to form a three-block tower from these blocks. As far as the robot is concerned all the blocks are identical and thus the tower can be formed from any of the blocks. The human has a goal of using one of those specific blocks for his/her own purpose. This means whenever the robot uses that specific block it would produce high levels of frustration within the human. The goal of the robot is thus to use this negative reward to update its policy to make sure that it doesnt use one of the blocks that the human requires.

For the first phase of training, we trained the agent using a simulated model of the task. For the state representation, we used a modified form of the IPC BlocksWorld pddl domain. We used a factored representation of the state with 36 predicates and one additional predicate `tower3_formed` to detect task completion. At every step, the agent has access to 50 actions to manipulate the blocks on the table and 80 ad-

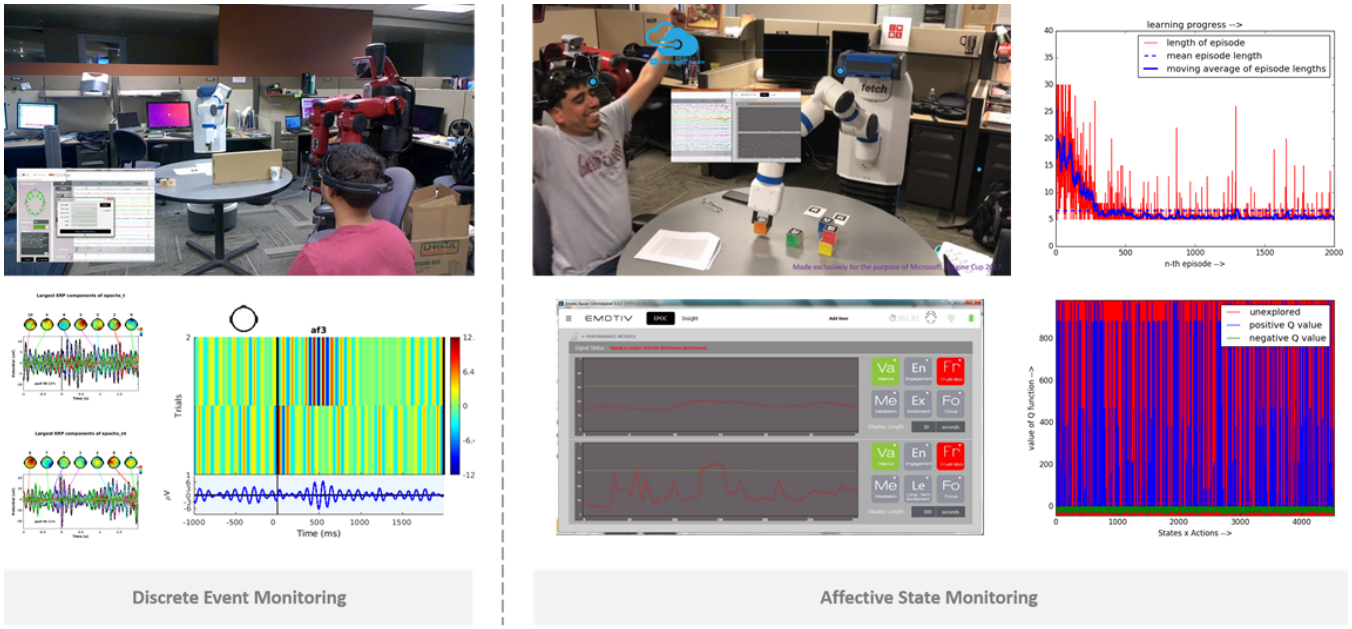

Figure 8: Different modes of EEG feedback - the robot can observe response to discrete events (left - listening for p300s) and listen to longer term affective states of the human (right - a reinforcement learner using stress values as negative feedback), and use this information to refine its policies.

ditional actions `form3tower` to check for the goal. As for the task rewards, each action is associated with a small negative reward and if the agent achieves the goal it receives a large positive reward. We also introduced an additional reward for every time the number of `ontable` predicates reduces (which means the agent is forming larger towers) to improve the convergence rate. We found that the agent converged to the optimal policy (the agent achieves the goal in 5 steps) at around 800 iterations. Figure 8 shows the length of the episodes produced after each iteration and the distribution of Q values across the table. Once the initial bootstrapping process was completed, we used the resultant Q-value table as our input for the second phase of the learning, as seen in the video demonstration. While there are some issues with convergence that are yet to be resolved, initial results showing the robot exploring new policies using the stress signals are quite exciting.

Conclusions & Future Work

In conclusion, we presented two approaches to improve collaboration among humans and robots from the perspective of task planning, either in terms of an interactive plan execution process or in gathering feedback to inform the human-aware decision making process. To this end, we discussed the use of holograms as a shared vocabulary for effective communication in an augmented workspace. We also discussed the use of EEG signals for immediate monitoring, as well as long term feedback on the human response to the robot, which can be used by a learning agent to shape its policies towards increased human-awareness. Such modes of interaction opens up several exciting avenues of research. We mention a few of these below.

Closing the planning-execution loop The ability to project intentions and interact via those projections may be considered in the plan generation process itself - e.g. the robot can prefer a plan that is easier to project to the human for the sake of smoother collaboration. This notion of projection-aware task or motion planning adds a new dimension to the area of human-aware planning.

A holographic vocabulary also calls for the development of representations - PDDL3.x - that can capture complex interaction constraints modeling not just the planning ability of the agent but also its interactions with the human. Further, such representations can be *learned* to generalize to methods that can, given a finite set of symbols or vocabulary, compute domain independent projection policies that decide what and when to project to reduce cognitive overload on the human.

ERP and timed events Perhaps the biggest challenge towards adopting ERP feedback over a wide variety of tasks is the reliance of detecting these signals on the exact time of occurrence of the event. Recent advancements in machine learning techniques can potentially allow windowed approaches to detect such signals from raw data streams.

Evaluations While preliminary studies with fellow graduate student subjects have been promising, we are currently working towards systematic evaluation of our system under controlled conditions, complying with the ISO 9241-11:1998 standards, targeted at professionals who are engaged in similar activities repeatedly over prolonged periods. This is essential in evaluating such systems since the value of information in projections is likely to reduce significantly with expertise and experience.

Acknowledgments. This research is supported in part by the ONR grants N00014161-2892, N00014-13-1-0176, N00014- 13-1-0519, N00014-15-1-2027, and the NASA grant NNX17AD06G. We would also like to thank Professors Yu Zhang and Heni Ben Amor (Arizona State University) for their valuable inputs on the project.

References

Acqualagna, L.; Treder, M. S.; Schreuder, M.; and Blankertz, B. 2010. A novel brain-computer interface based on the rapid serial visual presentation paradigm. In *EMBC*, 2686–2689. IEEE.

Bi, L.; Fan, X. A.; and Liu, Y. 2013. Eeg-based brain-controlled mobile robots: A survey. *IEEE SMC*.

Chadalavada, R. T.; Andreasson, H.; Krug, R.; and Lilienthal, A. J. 2015. That's on my mind! robot to human intention communication through on-board projection on shared floor space. In *ECMR*.

Christensen, H. I.; Batzinger, T.; Bekris, K.; Bohringer, K.; Bordogna, J.; Bradski, G.; Brock, O.; Burnstein, J.; Fuhlbrigge, T.; Eastman, R.; et al. 2009. A roadmap for us robotics: from internet to robotics. *Computing Community Consortium and Computing Research Association*.

Dragan, A.; Bauman, S.; Forlizzi, J.; and Srinivasa, S. 2015. Effects of robot motion on human-robot collaboration. In *HRI*.

Ferrez, P. W., and Millán, J. d. R. 2008. Error-related eeg potentials generated during simulated brain–computer interaction. *EMBC*.

Guger, C.; Allison, B. Z.; Großwindhager, B.; Prückl, R.; Hintermüller, C.; Kapeller, C.; Bruckner, M.; Krausz, G.; and Edlinger, G. 2012. How many people could use an ssvep bci? *Frontiers in neuroscience*.

Hayes, B., and Scassellati, B. 2016. Autonomously constructing hierarchical task networks for planning and human-robot collaboration. In *ICRA*, 5469–5476. IEEE.

Hoffmann, U.; Vesin, J.-M.; Ebrahimi, T.; and Diserens, K. 2008. An efficient p300-based brain–computer interface for disabled subjects. *Journal of Neuroscience methods* 167(1):115–125.

Ishii, K.; Zhao, S.; Inami, M.; Igarashi, T.; and Imai, M. 2009. Designing laser gesture interface for robot control. In *INTERACT*.

Iturrate, I.; Chavarriaga, R.; Montesano, L.; Minguez, J.; and del Millan, J. R. 2012. Latency correction of error potentials between different experiments reduces calibration time for single-trial classification. In *EMBC*, 3288–3291. IEEE.

Iturrate, I.; Montesano, L.; and Minguez, J. 2010. Single trial recognition of error-related potentials during observation of robot operation. In *EMBC*, 4181–4184. IEEE.

Kaminka, G. A. 2013. Curing robot autism: a challenge. In *AAMAS*.

Karpas, E.; Levine, S. J.; Yu, P.; and Williams, B. C. 2015. Robust execution of plans for human-robot teams. In *ICAPS*, 342–346.

Leutert, F.; Herrmann, C.; and Schilling, K. 2013. A spatial augmented reality system for intuitive display of robotic data. In *HRI*.

Look, C. 2016. Robots are coming, but not for your job. Bloomberg.

Maurtua, I.; Pedrocchi, N.; Orlandini, A.; de Gea Fernández, J.; Vogel, C.; Geenen, A.; Althoefer, K.; and Shafti, A. 2016. Fourbythree: Imagine humans and robots working hand in hand. In *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*, 1–8. IEEE.

Mistry, P.; Ishii, K.; Inami, M.; and Igarashi, T. 2010. Blinkbot: look at, blink and move. In *UIST*, 397–398. ACM.

Muro, M., and Andes, S. 2015. Robots seem to be improving productivity, not costing jobs. Harvard Business Review.

Omidshafiei, S.; Agha-Mohammadi, A.-A.; Chen, Y. F.; Ure, N. K.; How, J. P.; Vian, J.; and Surati, R. 2015. Mar-cps: Measurable augmented reality for prototyping cyber-physical systems. In *AIAA ARC*.

Omidshafiei, S.; Agha-Mohammadi, A.-A.; Chen, Y. F.; Ure, N. K.; Liu, S.-Y.; Lopez, B. T.; Surati, R.; How, J. P.; and Vian, J. 2016. Measurable augmented reality for prototyping cyberphysical systems: A robotics platform to aid the hardware prototyping and performance testing of algorithms. *IEEE Control Systems* 36(6):65–87.

Perera, V.; Selvaraj, S. P.; Rosenthal, S.; and Veloso, M. 2016. Dynamic Generation and Refinement of Robot Verbalization. In *RO-MAN*.

PRNewswire. 2016. Global survey: 1.3 million industrial robots to enter service by 2018. The International Federation of Robotics.

Ramli, R.; Arof, H.; Ibrahim, F.; Mokhtar, N.; and Idris, M. Y. I. 2015. Using finite state machine and a hybrid of eeg signal and eog artifacts for an asynchronous wheelchair navigation. *Expert Systems with Applications* 42(5):2451–2463.

Rao, R. P. 2013. *Brain-computer interfacing: an introduction*. Cambridge University Press.

Robotenomics. 2017. Industrial robot.

Ruffaldi, E.; Brizzi, F.; Tecchia, F.; and Bacinelli, S. 2016. Third point of view augmented reality for robot intentions visualization. In *AVR*, 471–478. Springer.

Salazar-Gomez, A. F.; DelPreto, J.; Gil, S.; Guenther, F. H.; ; and Rus, D. 2017. Correcting robot mistakes in real time using eeg signals. In *ICRA*. IEEE.

Sato, S., and Sakane, S. 2000. A human-robot interface using an interactive hand pointer that projects a mark in the real work space. In *ICRA*, volume 1, 589–595. IEEE.

Sean O'Kane. 2015. Microsoft used this adorable robot to show off new hololens features. The Verge.

Shen, J.; Jin, J.; and Gans, N. 2013. A multi-view camera-projector system for object detection and robot-human feedback. In *ICRA*.

Tellex, S.; Knepper, R.; Li, A.; Rus, D.; and Roy, N. 2014. Asking for help using inverse semantics. In *RSS*.

Turk, M., and Fragoso, V. 2015. Computer vision for mobile augmented reality. In *Mobile Cloud Visual Media Computing*. Springer. 3–42.

Watanabe, A.; Ikeda, T.; Morales, Y.; Shinozawa, K.; Miyashita, T.; and Hagita, N. 2015. Communicating robotic navigational intentions. In *IROS*, 5763–5769. IEEE.

Weiss, S. 2015. Could your robot hurt you? perhaps, but not intentionally. Brown Human-Robot Interaction (HRI) Lab.

Zhang, H.; Chavarriaga, R.; Khaliliardali, Z.; Gheorghe, L.; Iturrate, I.; and d R Millán, J. 2015. Eeg-based decoding of error-related brain activity in a real-world driving task. *Journal of neural engineering*.

Zhang, Y.; Sreedharan, S.; Kulkarni, A.; Chakraborti, T.; Zhuo, H. H.; and Kambhampati, S. 2017. Plan explicability and predictability for robot task planning. In *ICRA*. IEEE.

WEB PLANNER: A Tool to Develop Classical Planning Domains and Visualize Heuristic State-Space Search

Maurício C. Magnaguagno, Ramon Fraga Pereira, Martin D. Móre and **Felipe Meneguzzi**
School of Computer Science (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre - RS, Brazil
{mauricio.magnaguagno, ramon.pereira, martin.more}@acad.pucrs.br
felipe.meneguzzi@pucrs.br

Abstract

Automated planning tools are complex pieces of software that take declarative domain descriptions and generate plans for complex domains. New users often find it challenging to understand the plan generation process, while experienced users often find it difficult to track semantic errors and efficiency issues. To simplify this process, in this paper, we develop a cloud-based planning tool with code editing and state-space visualization capabilities. The code editor focuses on visualizing the domain, problem, and resulting sample plan, helping the user to see how such descriptions are connected without changing context. The visualization tool explores two alternative visualizations aimed at illustrating the operation of the planning process and how the domain dynamics evolve during plan execution.

1 Introduction

Classical planning algorithms typically require a declarative domain specification (often in PDDL (McDermott et al. 1998; Gerevini and Long 2005)) describing action schemata, which, in turn, define the dynamics of the underlying domain. Given the declarative nature of the formalism, planning algorithm implementations are often opaque regarding the intermediate steps between reading the formalism and generating a plan. Thus, writing such specifications may be a challenging task for new users even for simple domains, while detecting semantic mistakes in complex domains is always non-trivial. Practical applications of classical planners require not only a formalization of the domain in PDDL that is correct, but also takes advantage of the search mechanisms employed by the underlying planners to find solutions efficiently.

Most modern classical planning solvers (Hoffmann and Nebel 2001; Helmert 2006; Richter and Westphal 2010; Hoffmann 2011) use heuristic functions to estimate which states are likely to be closer to the goal state and save time and memory during the planning process. Different planning domains may require different heuristic functions to focus the search on promising branches and be solved within a reasonable time with little memory footprint. Thus, key to understanding the efficiency of a domain formalization is its impact on the heuristic function used by the underlying planner.

Even when the user successfully compiles and executes a planning instance with the chosen heuristic function the planner may fail to find a correct plan for the intended domain. In these cases, virtually no planning algorithm offers extra information, and the user only knows that the domain or problem are described in a way that makes it impossible to find a valid plan. Finally, since most planners are academic projects made to execute under very specific environments they lack a clear documentation to guide new users in the compilation process, while a web-based planner offers planning algorithms with no setup time.

This paper describes a tool to address the challenges of helping a domain expert to tune a formalization to any planning heuristics and spotting semantic errors in planning domains. Our tool, which we describe in Section 3, includes a PDDL code editor with syntax highlight and auto-complete aimed at helping users to efficiently develop PDDL domains in a similar workflow to many popular integrated development environments (IDEs). Importantly we integrate the editor to two visualization tools, described in Section 2, developed to help users cope with the declarative nature of PDDL and explore the effects of changes to the domain in solving concrete problems. First, we use a visual metaphor from the literature to see how a plan execution achieves (or does not) a goal state from an initial state (Magnaguagno, Pereira, and Meneguzzi 2016). Second, we develop a new state-space search visualization that uses tree drawing (in both cartesian and radial layouts) in conjunction with heatmaps to represent how the distance (*e.g.*, how colder or warmer) to the goal state changes during search. We use a case study in Section 4 to illustrate how our approach works and validate our approach from user tests, which we describe in Section 5 showing the results we obtained from employing the tool in a planning course. In Section 6, we survey related work on planning tools and data visualization, and conclude the paper in Section 7 discussing our conclusions and future work.

2 Background

2.1 Planning

Planning is the problem of finding a sequence of actions (*i.e.*, plan) that achieves a particular goal from an initial state (Ghallab, Nau, and Traverso 2004). A state is a finite set of facts that represent logical values according to some

interpretation. Facts are divided into two types: positive and negated facts. Predicates are denoted by an n-ary predicate symbol applied to a sequence of zero or more terms. An operator is represented by: a name that represents the description or signature of an action; a set of preconditions, *i.e.*, a set of facts or predicates that must be true in the current state to be executed; a set of effects, which has an add-list of positive facts or predicates, and a delete-list of negative facts or predicates. An action is an instantiated operator over free variables. A planning instance is represented by: a domain definition, which consists of a finite set of facts and a finite set of actions; and a problem definition, which consists of an initial state and a goal state. The solution of a planning problem is a plan, which is a sequence of actions that modifies the initial state into one in which the goal state holds by the successive execution of actions in a plan. To formalize planning instances, we use the STRIPS (Fikes and Nilsson 1971) fragment of PDDL (McDermott et al. 1998), which contains domain and problem definition in different files.

Heuristic functions are used to estimate the cost of achieving a particular goal (Ghallab, Nau, and Traverso 2004). In classical planning, this estimate is often the number of actions to achieve the goal state from a particular state by exploring only promising states. Estimating the number of actions is a NP-hard problem (Bylander 1994). In automated planning, heuristics can be domain-dependent or domain-independent, and a well-tuned heuristic can result in a substantial reduction in search time by pruning a vast part of the state-space.

2.2 Data Visualization

Visualization techniques aim to convey some kind of information using graphical representation (Ward, Grinstein, and Keim 2015). The use of data visualization techniques is often associated to a set of data with the aim of communicating a particular information clearly and efficiently via graphical representation.

Data visualization techniques are concerned with what is the best way to display a dataset, for instance, how to display relation information. Relation information can be displayed efficiently by using hierarchies that convey relation information. Edges in a hierarchical tree represent a relation between nodes. A Cartesian tree visualization is a way to display hierarchical trees as a coordinate system. A radial tree visualization is a way to display a hierarchical tree structure in which such tree expands outwards and radially. In Subsection 3.2 we explore such tree visualizations. Besides hierarchical visualization, we highlight other visualization methods that are closely related to the ones we develop in this work, such as *Gantt charts* (Wilson 2003), which are used to show how tasks are correlated and how much time is expected to complete them, *Waveforms* (Ha 2010, Chapter 1 – page 2) are used to express the behavior of analog or digital data through time, and Heatmap visualization (Ward, Grinstein, and Keim 2015), which uses a color scheme to illustrate values in a graphic in which each color in the scheme represents one limit value and the many values in the interval are represented by the mix of such colors.

3 WEB PLANNER Architecture

We designed our tool envisioning a development process centered around two tasks by the domain developer. In the first task, the user aims to describe both domain and problem correctly. In the second task, the user tries to identify details of the description (in terms of predicate use) that impact performance and how these predicates appear during the planning process. The domain designer is free to move between these tasks and repeat until satisfied with the results. Once a planning instance is described it is possible to visualize the explored state-space, even when the planning process fails. When the planning process returns a plan the user is able to visualize how predicates were added or deleted by each action in the plan. Such interface could also help planning system developers to explore how planners in development behave.

To avoid the considerable setup time of some planner implementations and maintain a consistent interface across platforms, we use a web interface. The planner is executed in the server, while the editor, output and visualizations are displayed and executed in the browser. The communication between the two sides uses JSON[1]. Figure 1 shows the architecture of the WEB PLANNER.

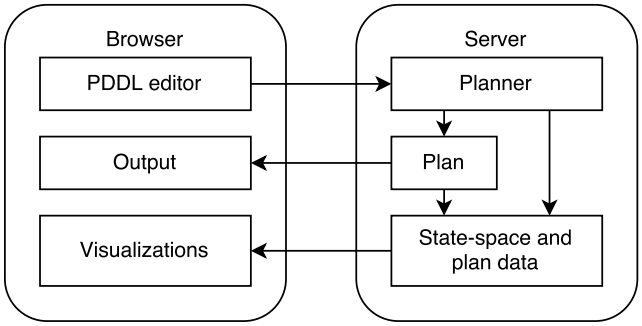

Figure 1: Overview of the WEB PLANNER Architecture.

3.1 Domain Development Interface

To better describe planning domains and problems, we identified some requirements to improve the process of editing such descriptions, as follows:

- PDDL syntax highlight and auto-complete to alleviate user learning curve. For example, to define a new action, our PDDL editor provides an action-template (an auto-complete function of our editor, pressing *CTRL+Space* after typing the word *action*) that shows how an action is defined in PDDL, as shown in Figure 2. Our editor also provides templates for domain and problem description, just pressing *CTRL+Space* after typing the word *domain* or *problem*, respectively;
- See and edit both domain and problem simultaneously, avoid going back and forth between descriptions gives a

[1]JSON (JavaScript Object Notation) is an open-standard format for structuring data.

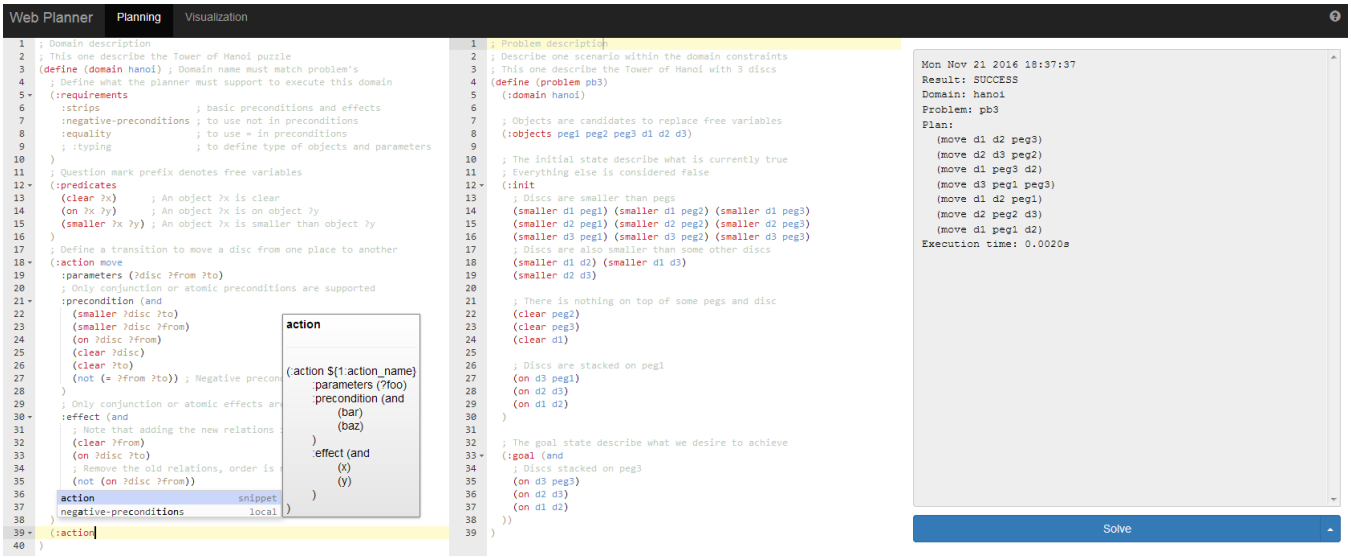

Figure 2: WEB PLANNER editor interface with domain editor (left), problem editor (center) and plan output (right). Action template is provided by autocomplete.

better idea of them being used together while minimizing the user effort; and

- Execute the current planning instance without a context change;

To meet such requirements, we split the editor interface horizontally in 3 parts: domain, problem, and planner output. The ability to see input alongside output is very important for both advanced users, that are modifying or extending legacy PDDL, and new users, such as students, that are not used with the domain and problem distinction. Instead of starting with a blank planning instance we opted for a simple but complete Towers of Hanoi example to be loaded by default.

The solve button sends the planning instance to the server to obtain an output based on the domain and problem descriptions contained in the editor. Our editor uses brace, a variant of the ace editor, and it is able to highlight most PDDL elements, some of which are currently not supported by the back-end planner. The output provided by the planner contains the plan and execution time when successful, error messages when the parser fails, or a failure message when no plan is found. Due to screen space limitations and demand, the visualizations were left to a secondary interface, as users can only visualize after the initial description step.

3.2 Visualization Interface

We currently support two visualizations, one focusing on the explored state-space and the other on the execution of the first plan found.The impact of heuristics in the state-space is often introduced in AI lectures using images, such as the ones from Figure 3, to show how the contour of the explored states grows in all directions on blind search and towards the goal state in informed search (using heuristics) (Russell and Norvig 2009, Chapter 3 – page 97). Such images target an audience new to the concept of using a computed auxiliary function to speed-up search. More interesting examples are possible with animations on a grid, showing the step-by-step process of search. Since not all domains can be mapped to a grid, the visualization process is often limited to path-finding domains. To generate such contours we opted for a tree-based visualization.

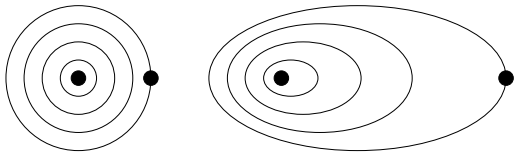

Figure 3: Search contours are defined by search mechanism and heuristic function, either equally exploring in all directions (left) or giving priority towards the goal state (right).

Heuristic Visualization: The heuristic visualization we developed takes advantage of interactive elements to avoid information overload while providing alternative layouts, cartesian and radial tree visualizations. The radial layout matches the abstraction used by heuristic examples while the cartesian layout generates a visualization more compact. In practice, we use the Reingold-Tilford algorithm[2] to display both tree layouts. Using tree visualizations we aim to show how planning heuristics explore the state-space to achieve a particular goal.

To compare and explore the state-space of a planning instance, we implemented two planning methods. The first method is based on breadth-first search, and thus uses no

[2]Reingold-Tilford is an algorithm for an efficient tidy arrangement of layered nodes. We use an implementation based on a D3 example available at: `http://bl.ocks.org/mbostock/4063550`.

heuristic, exploring the state space in the order of distance from the initial state. The second method implements greedy best-first search using Hamming distance (Hamming 1950) as a heuristic. Our visualization tool supports other search mechanisms and heuristic functions as long as such mechanisms search through the state-space, the selected ones are used only as examples.

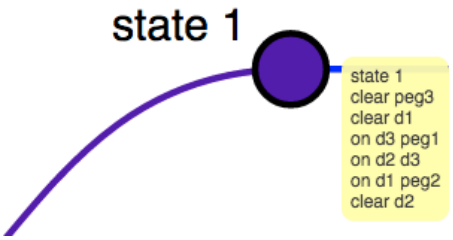

Figure 4: Tooltip that displays the set of instantiated predicates in a state. This figure illustrates state 1 and its predicates for a planning instance of the Hanoi domain.

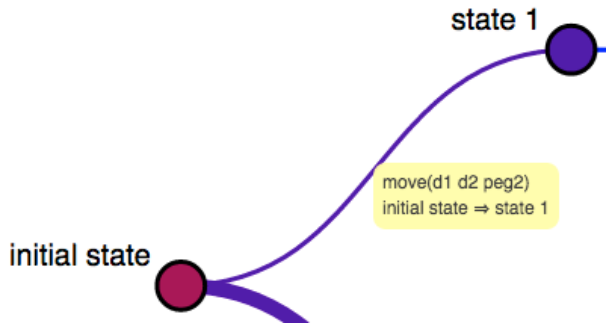

Figure 5: Tooltip that displays the instantiated action applied between two states. This figure illustrates state 1 and its predicates for a planning instance of the Hanoi domain.

With the explored state-space and heuristic information about each state we use an hierarchical tree to represent the data obtained from the planning process. In this tree, each node represents a state (*i.e.*, a set of instantiated predicates), and an edge represents a state-transition (*i.e.*, the execution of an action). The root node represents the initial state. Our visualization displays the state-space of a planning heuristic by coloring the estimated distance between states using a heatmap. Information such as the content of the state or the applied action is hidden until the user hovers that position with the cursor to display such data in a tooltip, as show in Figures 4 and 5. Nodes and edges are colored according to the estimated distance to the goal state. Red nodes represent states closer to the goal state, *i.e.*, warmer, while distant nodes are represent by blue, *i.e.* colder. The heuristic gradient is defined in Figure 6. It is important to highlight that the estimated distance can be different according to the used heuristic. Therefore, the state-space search can reach different states in a different order according to the heuristic used, for example, Figures 10 and 11. As more states are explored the more visible the contours are, as seen in Figure 7. If planning is successful the edges from initial to goal node are emphasized. The trees are also generated to failed planning instances, which can be used for debug purposes.

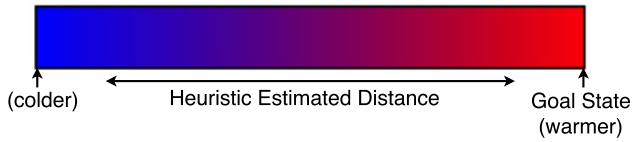

Figure 6: Color scheme that our visualization tool uses to represent the estimated distance.

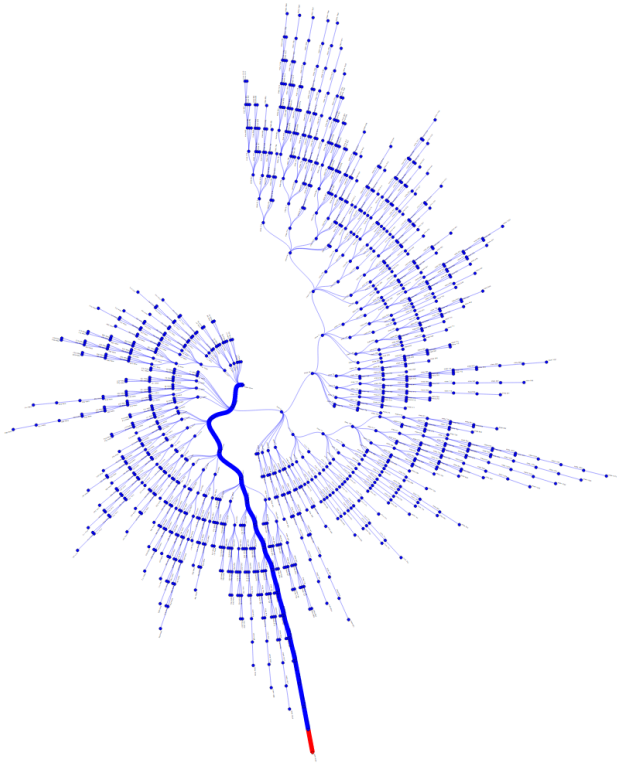

Figure 7: Contours become visible as more states are explored. This planning instance obtain all goal predicates at the same time, which makes the heatmap mostly blue (colder), while the goal state is located at the bottom in red (warmer).

Dovetail Metaphor Visualization: The second visualization we implemented is a visual metaphor called Dovetail (Magnaguagno, Pereira, and Meneguzzi 2016), which is useful to see how predicates change along the plan execution. Each ground predicate that appear in an action effect is represented as one line while both initial state, goal state and actions are represented as columns. Our interface allows a user to move and zoom to parts of this visualization (illustrated in Figure 8), with tooltips providing extra information as shown in Figure 9 for the domain of the case study of Section 4. The use of this visual abstraction (Dovetail) aims

to improve the learning curve for defining and debugging planning domains and problems.

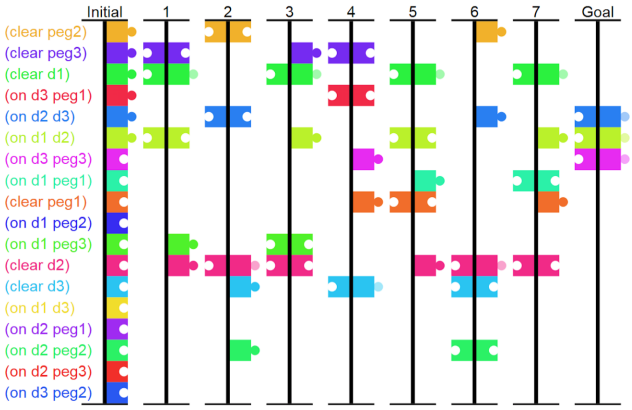

Figure 8: Dovetail plan visualization of Hanoi domain with 3 discs and a plan of size 7.

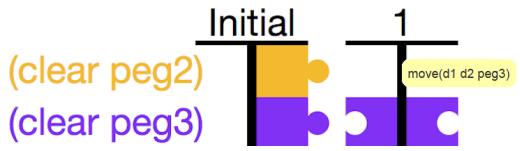

Figure 9: Tooltip that displays the instantiated action in a plan on Dovetail.

4 Case Study

To validate our tool, we developed a case study of a planning instance using different planning heuristics displaying the state-space.We selected the Hanoi towers domain to illustrate what can be expected from the visualizations. The Hanoi domain describes the towers of Hanoi problem, where one must move a stack of discs from one peg to another without stacking a larger disc onto a smaller one, three pegs are available in total. Problem instances of this domain show that the goal cannot be accomplished in an incremental way, requiring the plan to build and destroy partial towers several times to obtain the complete tower in the final peg. Domains that present such behavior are not pruned as much as others by the Hamming distance as a heuristic function and have a visible color fluctuation between the gradient limits instead of a clear movement towards red, as seen in the Cartesian tree of Figure 10. The Cartesian tree is better to represent and compare such smaller graphs, while the radial tree highlights the side to which the heuristic gave priority during search, as seen in Figure 11, where the top-left branch was not explored. Other domains may suddenly reach a goal state from a mostly blue colored graph, in which all states are far away from the goal, as seen in Figure 7, or incrementally reaching the goal clearly going from one extreme of the gradient to the other, as in the Logistics domain.

To better understand how the predicates are affected by the plan we use the Dovetail metaphor. This particular Hanoi

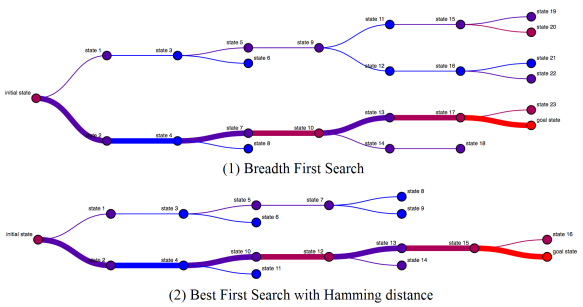

Figure 10: Cartesian tree visualizations of the state-space of Hanoi with 3 discs.

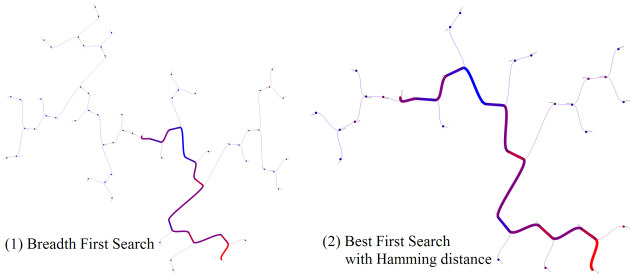

Figure 11: Radial tree visualizations of the state-space of Hanoi with 3 discs.

planning instance is solved by a 7-step plan, represented by the pieces labeled with numbers at the top, Figure 8. Each piece has preconditions represented on the left side and effects represented on the right side. In this case we can see the first action, *move(d1 d2 peg3)*, moving a clear disc *d1* that starts on disc *d2* to a clear peg *peg3*, leaving *d2* clear and *peg3* not clear. We can see the predicate *clear d1* being tested by each odd-index action, revealing the pattern of movements related with the disc *d1*.

5 Survey Results

To evaluate WEB PLANNER, a group of four users from our automated planning course[3] were asked to fill a survey after using the tool to describe the *RPG* domain from the International Competition on Knowledge Engineering for Planning and Scheduling [4]. The survey contained the following questions and answers:

- How familiar are you with automated planning languages and algorithms?
  - Only 2 users have used PDDL before.
- Did Web-planner visualizations help you to find any bugs/errors/interesting points during the course of your task?
  - One user found missing preconditions.
- Mark other planners/tools you used in your experiments:

[3] http://github.com/pucrs-automated-planning/syllabus
[4] http://ickeps2016.wordpress.com

- Fast-Downard (1), JavaFF (1), JavaGP (3), Planning.domains (3), STRIPS-Fiddle (1)

- Which features you missed the most?
  - Support more requirements (2), Auto-complete (1), Option to clear console (1), Find (common) errors in PDDL (1).

Results of system reaction show evidence of the utility of our tool, albeit with many suggested improvements, in Figure 12 with minimum, maximum and average represented. The current planning output must be improved in order to provide more meaningful messages about errors while taking advantage of the integrated editor to draw attention to specific lines where parsing errors were detected. Other improvements are more related to the editor itself, making it more flexible to attend different user needs, such as theme, font size and the ability to re-size each part of the editor. Users also asked for more planners/requirements to be supported.

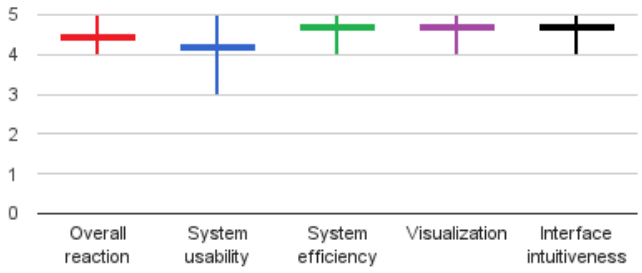

Figure 12: Survey results, users were asked to evaluate the system between frustrating (0) and satisfying (5).

6 Related work

We now discuss related work and tools that are used to formalize planning domains, visualize changes on a large amount of hierarchical data, and visualize state-space search algorithms.

`Planning.Domains`[5] is a collection of web tools for automated planning. These web tools provide a web PDDL editor, an API that contains a wide collection of PDDL benchmark domain and problem files (most of them used on the International Planning Competition), and a planner in the cloud that allows using not only a planning solver, but also VAL (Howey, Long, and Fox 2004), a plan validation tool. Similar to our approach, `Planning.Domains` provides a PDDL editor, however, our approach provides not only a web editor with syntax highlighting, but also a set of tools to develop and visualize planning domains using metaphors and alternative data visualization methods.

To edit PDDL domains and problems, we highlight two approaches, `myPDDL` and PDDL Studio. `myPDDL`[6] (Strobel and Kirsch 2014) is an editor extension for Sublime Text, which provides PDDL syntax highlighting, snippets, and domain visualization (*e.g.*, diagram types). PDDL Studio (Plch et al. 2012) is an IDE to edit PDDL domains and problems. This IDE provides syntax highlighting, code completion, and context hints specifically designed for PDDL.

Graphical Interface for Planning with Objects (GIPO) (Simpson, Kitchin, and McCluskey 2007) is a tool for planning domain knowledge engineering that allows the specification of domains in PDDL and *Hierarchical Task Network* (HTN). Besides domain knowledge engineering, GIPO provides an animator tool to graphically inspect the plans produced by the internal planner, given a domain and problem specification. Unlike our approach, GIPO checks a set of plans to validate a domain and problem specification, indicating whether the domain and problem specification do support the given plans. Similar to Dovetail metaphor we implemented in WEB PLANNER, GIPO also provides an animator tool to visualize how a sequence of actions (*i.e.*, a plan) connects to form a plan that achieves a goal state from an initial state. *VisPlan* (Glinskỳ and Barták 2011) is an interactive tool to visualize and verify plans' correctness. This tool is closely related to Dovetail metaphor in the sense of helping planning users to better understand how a sequence of actions achieve a goal from an initial state. *VisPlan* identifies possible flaws (*i.e.*, incorrect actions) in a plan, allowing users to manually modify this plan by repairing these identified flawed actions.

PDVer (Raimondi, Pecheur, and Brat 2009) is a methodology and tool that verifies if a PDDL domain satisfies a set of requirements (*i.e.*, planning goals). This tool allows an automatic generation of these requirements from a *Linear Temporal Logic* (LTL) specification into a PDDL description. This tool is concerned with how the corresponding PDDL action constraints are translated from an LTL specification. Whereas *PDVer* provides a summary of test cases (positive and negative) indicating why a PDDL domain specification does not satisfy a set of requirements to achieve a goal.

itSimple (Vaquero et al. 2012) is concerned with domain modeling, using steps to guide the user from informal requirements (UML) to an objective representation (Petri Nets). The itSimple features provide a visualization and simulation tool to help understanding planning domains through diagrams. *itSimple* uses UML diagrams to model planning instances and Petri Nets for validating planning instances.

Magnaguagno *et al.* (2016) developed a visual metaphor to visualize and learn how the planning process works. We have applied this visual metaphor in our web tool by using colors for different instantiated predicates in a state along a plan execution. Dovetail results suggest that this visual metaphor can be useful to define and debug the planning process.

We found two approaches to data visualization suitable for heuristics. In (Kuwata and Cohen 1993), Kuwata and Cohen develop visualization methods to understand and analyze the search-space and behavior of heuristic functions, by exploring the usefulness of these methods on shaping state-space search. The heuristic functions they explore are A* and IDA*. Tu and Shen (2007) propose a set of strategies to visualize and compare changes in hierarchical data using treemaps.

[5] `http://planning.domains`

[6] `http://github.com/Pold87/myPDDL`

7 Conclusions

In this paper, we report on a cloud-based planning tool we developed, which consists of a PDDL editor to formalize planning domains and problems, and visualizations to help understand the effect of planning heuristics in the domains. This work aims to simplify the setup process required to execute planners while providing visualizations to better understand how domain differences and heuristics can impact the performance of the planner. A small-scale survey indicates promising results while asking for improvements that are already in development.

As future work, we intend to support user-defined heuristics in our planner along with alternative options to the user, such as selectable color schemes for the visualization and a side-by-side state-space view for comparison. We believe that such tool can help new heuristics to be developed and tested, giving the user a better grasp of the impact of heuristics to the state-space exploration, which is usually an invisible entity. Instead of outputting only a plan and the time to compute it in the editor interface, we expect to add not only better parsing error messages but also detection of bad constructions, such as unnecessary requirements or effects that are equal to preconditions. Our WEB PLANNER tool is available at `http://web-planner.herokuapp.com`.

Acknowledgments

We acknowledge the support given by CAPES/Pro-Alertas (88887.115590/2015-01) and CNPQ within process number 305969/2016-1 under the PQ fellowship.

References

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Journal of Artificial Intelligence Research (JAIR)* 69:165–204.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Journal of Artificial Intelligence Research (JAIR)* 2(3):189–208.

Gerevini, A., and Long, D. 2005. Plan Constraints and Preferences in PDDL3. *The Language of the Fifth International Planning Competition. Technical Report, Department of Electronics for Automation, University of Brescia, Italy*.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning - Theory and Practice.* Elsevier.

Glinskỳ, R., and Barták, R. 2011. Visplan–interactive visualisation and verification of plans. *Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)* 134–138.

Ha, T. T. 2010. *Theory and design of digital communication systems*. Cambridge University Press.

Hamming, R. W. 1950. Error detecting and error correcting codes. *Bell System Technical Journal* 29(2):147–160.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research (JAIR)* 14(1):253–302.

Hoffmann, J. 2011. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *Computing Research Repository (CoRR)* abs/1106.5271.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), 15-17 November 2004, Boca Raton, FL, USA*, 294–301.

Kuwata, Y., and Cohen, P. R. 1993. Visualization Tools for Real-Time Search Algorithms. *Computer Science Technical Report*.

Magnaguagno, M. C.; Pereira, R. F.; and Meneguzzi, F. 2016. DOVETAIL - An Abstraction for Classical Planning Using a Visual Metaphor. In *Proceedings of FLAIRS, 2016.*

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL − The Planning Domain Definition Language. *Technical Report – Yale Center for Computational Vision and Control*.

Plch, T.; Chomut, M.; Brom, C.; and Barták, R. 2012. Inspect, edit and debug PDDL documents: Simply and efficiently with PDDL Studio. In *Proceedings of ICAPS'09*, 15–18.

Raimondi, F.; Pecheur, C.; and Brat, G. 2009. PDVer, a Tool to Verify PDDL Planning Domains. In *Proceedings of ICAPS'09 Workshop on Verification and Validation of Planning and Scheduling Systems, Thessaloniki, Greece*.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research (JAIR)* 39(1):127–177.

Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd edition.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning domain definition using GIPO. *Knowledge Eng. Review* 22(2):117–134.

Strobel, V., and Kirsch, A. 2014. Planning in the Wild: Modeling Tools for PDDL. In *Joint German/Austrian Conference on Artificial Intelligence*, 273–284. Springer.

Tu, Y., and Shen, H. W. 2007. Visualizing Changes of Hierarchical Data using Treemaps. *IEEE Transactions on Visualization and Computer Graphics* 13(6):1286–1293.

Vaquero, T.; Tonaco, R.; Costa, G.; Tonidandel, F.; Silva, J. R.; and Beck, J. C. 2012. itSIMPLE 4.0: Enhancing the modeling experience of planning problems. In *Proceedings of ICAPS'12*, 11–14.

Ward, M. O.; Grinstein, G.; and Keim, D. 2015. *Interactive Data Visualization: Foundations, Techniques, and Applications, Second Edition - 360 Degree Business*. Natick, MA, USA: A. K. Peters, Ltd., 2nd edition.

Wilson, J. M. 2003. Gantt charts: A centenary appreciation. *European Journal of Operational Research* 149(2):430 – 437.

Session Analysis using Plan Recognition

Reuth Mirsky and Ya'akov (Kobi) Gal
Department of Software and Information Systems Engineering
Ben-Gurion University of the Negev, Israel
{dekelr,kobig}@bgu.ac.il

David Tolpin
PayPal, Israel
dtolpin@paypal.com

Abstract

This paper presents preliminary results of our work with a major financial company, where we try to use methods of plan recognition in order to investigate the interactions of a costumer with the company's online interface. In this paper, we present the first steps of integrating a plan recognition algorithm in a real-world application for detecting and analyzing the interactions of a costumer. It uses a novel approach for plan recognition from bare-bone UI data, which reasons about the plan library at the lowest recognition level in order to define the relevancy of actions in our domain, and then uses it to perform plan recognition.

We present preliminary results of inference on three different use-cases modeled by domain experts from the company, and show that this approach manages to decrease the overload of information required from an analyst to evaluate a costumer's session — whether this is a malicious or benign session, whether the intended tasks were completed, and if not — what actions are expected next.

1 Introduction

Online websites are often open-ended and flexible interfaces, supporting interaction styles by users that include exogenous actions and high noise rate. Such interfaces provide a rich framework for users and are becoming increasingly prevalent in many web and mobile applications, but challenge conventional plan recognition algorithms for inferring users' activities with the software.

The open-ended nature of these settings afford a rich spectrum of interaction for clients: they can perform the same task in many different ways (such as logging in through a website or by a mobile application), engage in exploratory activities involving trial-and-error, they can repeat activities indefinitely (browsing), and they can interleave between activities.

This paper focuses on inferring customers' activities for an Internet financial company, in which customers widely engage in exploratory behavior, and present initial results for plan recognition in such settings that can be the basis of future work in this direction.

Our data set consists of customer sessions of a large financial company[1]. The customer sessions are recorded as low-level activities of 'click stream' — web page visits and user interface actions. This data is a rich source of information about customer's behavior, but cannot be used as-is to describe the intentions or plans of the customer: buy product, sell product, manage account, etc. The underlying model used for the recognition process is a set of hierarchical task networks (HTNs) called a "plan library", that allows hierarchical decomposition of a task, which can be used both as an explanatory accessory to an overseer and for prediction of future actions. Our main goal is to detect frauds, validate transactions and predict future actions — both in terms of the user's intents and in terms of the execution of these intents.

The basic stream of information can be thought of as a recording of customer's *conversation* with the system. Understanding the language of conversation would bring benefits in various areas. For example:

- Predictive verification — by following customer's actions in real time, we can have our systems guess next actions and alert when an agent biases from their predictive plan.
- Proactive validation — if we can understand the purpose of customer's visit based on the beginning of their activity, we can let our systems start validation of the session even before it takes place, facilitating faster response and better utilization of resources.
- Computer security — certain types of fraud are performed by malicious software controlling customer computers. We may be able to distinguish between malicious and benign actors on the other side of the wire.

However, as a flexible interface, the system allows the user to engage in more than one task at a time (making multiple purchases), make mistakes (pressing a button twice instead of once), repeat actions (browse between different parts of the interface) and more. All of these are features that define an exploratory environment [5].

While these traits make the interface useful and convenient to the user, it challenges conventional plan

[1] All interactions and examples were altered to preserve privacy

recognition algorithms for inferring users' activities with the software, as they need to reason about all these factors [1, 17].

2 Related Work

The tasks described in the introduction are traditionally addressed by activity recognition algorithms, which are used to tokenize the stream of data into detectable actions [2, 11, 4]. However, these algorithms usually cannot perform higher levels of inference to describe the agent's behavior or predict the future actions of the agent [8, 18]. Different approaches also used an HMM representation or data-driven learning to elicit tasks from low-level activities [14, 12].

Some notable exceptions are works by [15, 3, 16] which try to combine low level activities and higher level domain knowledge, or [16] which propose a multi-agent model for robot collaboration based on plan recognition. However, the works in these lines of research tend to use a domain-theory based recognition or other probabilistic representations, which do not capture the hierarchical nature of task decomposition like Hierarchical task networks (HTNs) and plan libraries.

In order to address the above tasks, we need to use hierarchical plan recognition, a field of research exploring algorithms that recognize the plans of the agent based on a partial sequence of actions, and predict future actions [10]. Few works [7, 6] did use a plan library as the underlying domain knowledge for the task, but they did not provide predictions or used this information to formalize the output. Other works [9] do output a complete hierarchy, but the plan recognition algorithm used does not work well in exploratory environments.

In this work, we present a criteria for constructing the plan library that will then be used by the agent to perform the plan recognition task. Thus, we do not need to use an activity recognizer. We integrate instead the low-level activities directly into the plan recognition task.

3 Definitions

Plan recognition takes as input a plan library and a partial sequence of observations, and outputs a set of possible explanations.

3.1 Plan Library

The plan library can be thought of as the language of actions, as well as goals we want to be able to recognize. The library must specify *terminals*, *non-terminals*, *goals*, and *derivation rules*.

Definition 1 *(Explanation) A plan library is a tuple* $PL = \langle T, NT, G, R \rangle$*, where* T *is a set of are low-level, observable action,*NT *is a set of complex level actions, composed from either* T*s or other* NT*s,* G *is a subset of non-terminals corresponding to the highest level of actions, representing the goals the agent can achieve and* R *is a set of derivation rules which specify how each complex action can be decomposed into other actions of the form* $\langle nt \rangle \rightarrow \langle sequence \rangle \mid \langle order \rangle$*, where:*

- $\langle nt \rangle$ *is a non-terminal, a complex action.*
- $\langle sequence \rangle$ *is a sequence of actions which compose the complex action.*
- $\langle order \rangle$ *is a list of tuples defining the the ordering of actions — actions must appear in the same order as in the tuple, but ordering between different tuples is unspecified. For example, '(login, addName)' states that the user must log-in before adding a card.*

Consider the following very simple plan library for our domain.

- $T = \{login, addName, addCredit, signup, submit, home, payment, success, transfer, confirm\}$
- $NT = \{AddAccount, Buy\}$
- $G = \{AddAccount, Buy\}$
- $R =$
  $AddAccount \rightarrow login, addName, addCredit \mid [(login, addName)(addName, addCredit)]$
  $AddAccount \rightarrow signup, addName, submit \mid [(signup, addName)(addName, submit)]$
  $Buy \rightarrow home, payment, success \mid [(home, payment)(payment, success)]$
  $Buy \rightarrow home, transfer, confirm \mid [(home, transfer)(transfer, confirm)]$

3.2 Plans and Explanations

Based on the definitions above, we can apply an algorithm to a sequence of actions to obtain explanations about ongoing behaviors and anticipated future actions.

A plan is a labeled tree $p = (V, E, \mathcal{L})$, where V and E are the nodes and edges of the tree, respectively, and $\mathcal{L}$ is a labeling function $\mathcal{L} : V \rightarrow NT \cup T$ mapping every node in the tree to either a basic or a complex action in the plan library. Each inner node is labeled with a complex action such that its children nodes are a decomposition of its complex action into constituent actions according to one of the refinement methods. The set of all leaves of a plan p is denoted by $leaves(p)$, and a plan is said to be *complete* iff all its leaf nodes are labeled basic actions, i.e., $\forall v \in leaves(p), \mathcal{L}(v) \in T$.

An *observation sequence* is an ordered set of basic actions that represents actions carried out by the observed agent. A plan p *describes* an observation sequence O iff every observation is mapped to a leaf in the tree. Formally, there exists an injective function $f : O \rightarrow leaves(p) \cap T$ such that $f(o) = v$. The observed agent is assumed to plan by choosing a subset of complex actions as intended goals and then carrying out a separate plan for completing each of these goals.

An agent may pursue several goals at the same time. Therefore, an explanation can include a set of plans, as described in the following definition:

Definition 2 *(Explanation) An explanation for an observation sequence is a set of plans such that each plan describes a mutually exclusive subset of the observation*

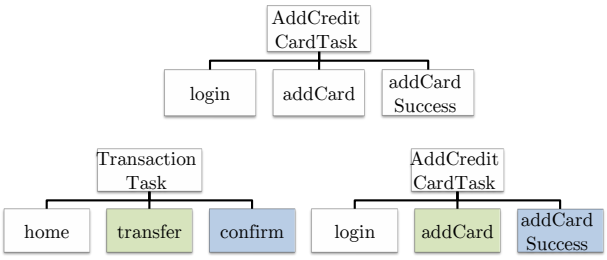

Figure 1: An Explanation with Three Plans.

sequence and taken together the plans describe all of the observations. We then say that the explanation describes *the observation sequence.*

For the rest of this paper we will use a running example based on the following sequence of processed observations: $[home, login, addName, login, addCredit]$ When we submitted the sequence and the plan library to a plan recognition algorithm, we obtained two explanations. The first one is presented in Figure 1.

This explanation is a set of trees. In each tree the leaves are either the actions we observed, or actions predicted to perform in the future. A special tag $frontier$ specifies the location in the tree from where the expansion may continue, based on future actions. Leaves with this tags are colored in green in the figure. Future actions that still has preceding actions that have not been executed are colored with blue. If a tree has no 'frontier' leaf (top tree in our example), then the goal is fully described by actions.

The output above represents the following explanation of the input sequence: (1) The agent performed one buy transaction and two account additions; (2) The agent performed the first action in the buy task, but then stopped – we expect that the next action to complete this task is $transfer$; (3) Regarding the account additions, one was completed and the other was not.

For our very basic example, one can think about a brute-force algorithm for explaining the activity. However in general, a sophisticated algorithm is required to provide short and meaningful explanations quickly, in a manner suitable for online processing. The remaining of this paper talks about the very beginning of work in collaboration with analysts and developers from the financial company.

4 Recognition Components

We now detail the components we use to perform the plan recognition task from the raw click stream of costumer sessions.

4.1 Preprocessing

A session is the basic unit of interaction between a customer and the system. Efficient analysis and validation of sessions in real time is crucial for technical realization of the company's business. The raw input in our domain is a stream of costumer sessions. Each session is constructed from a list of entries, where each entry has a specific timestamp, user information and a special label describing the page on the company's site the user is visiting. The average length of a session was 80.49 entries.

As a first step, we needed to reduce this number as many entries are irrelevant to the costumer's main tasks in the system and integrating them into terminals in the plan library would be both inefficient and less informative. We defined the following criteria for a session entry to be considered relevant to us:

Definition 3 *(Landmark in PL) A raw entry e in a complete session S is considered a landmark in relation to a plan library PL if $S \setminus \{e\}$ cannot describe a plan from PL.*

In collaboration with our partners inside the company, we elicited about 20 types of entries, defined by their page labels, which are landmarks for executing the relevant tasks. We classified each of the pages into a suitable basic action as they appear in our plan library. Counting only these pages as relevant actions we wish to observe, we received an average of 9.68 observations per session (with stdev of 7.26).

However, as an exploratory environment, many entries in a session are redundant even if they can be considered landmarks, as the user can make mistakes or perform a landmark action more times than necessary. To perform an elimination of these entries and to remain only with the most relevant observations, we performed another sifting. We discarded all observations that were a repetition of the earlier observation. After this elimination, we reduced the average number of relevant observations per session to 3.68 (stdev=2.62).

4.2 Recognition

There are many possible strategies that a costumer can use to perform interactions, and variations within each due to exploratory activities and mistakes carried out by the costumer. Even after the preprocessing we enforced on the set of actions performed by the costumer, there are still two types of exploratory behaviors which can hinder the ability of the recognizer the infer the costumer's actions correctly: (1) Exogenous actions: even after our preprocessing effort, many actions cannot be combined together when looking at the complete sequence, thus the plan recognizer cannot output any explanation the described all of the actions in the session. (2) One explanation may relate a given action to a relevant task, while another may relate this action to a failed attempt or a mistake. The space of possible explanations can become very large, even for a small number of observations.

The CRADLE algorithm [13] proposes solutions for both of these problems using three components:

Inference it receives as input a plan library, an observation sequence, and outputs a set of explanations, each of which is an explanation of the observation sequence in the sense of Definition 2.

Filters it filters redundant explanations according to a set of domain-independent conditions. A filter is a function taking a candidate explanation e, and returning *true* or *false* depending on whether the candidate explanation does or does not pass a certain condition. For our presented domain, we tried several values and filter types and finally set 3 filters: (1) The number of plans in the explanation is less than or equal to the average; (2) The number of frontier nodes in the explanation is less than or equal to the average; (3) The number of different plans in the explanation is less than 4. We discard all explanations which do not pass these thresholds

Exogenous Actions CRADLE can handle exogenous actions and mistakes and can omit them from the set of explanations as needed.

5 Empirical

We used a click stream of selected sessions, labeled by the company's analysts as sessions containing the relevant tasks. In total, we tested 3 types of sessions, with 50 instances of each session type: Buy, Add account for existing user (AAExist) and Add account for new user (AANew). For each of these sessions, we performed the process described in Section 4.

Figure 2 presents the average number of different types of explanations outputted by CRADLE per task: Total is the total number of explanations; Full Plan is the number of explanations in which at least on plan was completed; No Open means the number of explanations in which all plans were completed and had no open frontier. Notice that even in sessions that were labeled by analysts as relevant to some task, the plan recognition process shows that only a small portion of the sessions contain a completed task. These results are similar in the PHATT runs, even though it does not discard explanations, thus might output more explanations with a completed task. We intend to validate these results with other data to evaluate if these sessions were indeed sessions of incomplete tasks.

In order to evaluate the performance of CRADLE, we compared the runtime and number of outputted explanation in comparison to the PHATT algorithm, augmented with the exogenous actions handling of CRADLE (without this augmentation, PHATT would not be able to output any explanation at all that describes the complete sequence of observations). Table 5 summarizes these runs.

The first and most important thing to notice in the table, is the difference between the original length of a session entry, and the outputted set of explanations by the plan recognizers. An average decrease of 83% in the number of entities representing the session. Such a decrease allows a faster analysis of the session, since

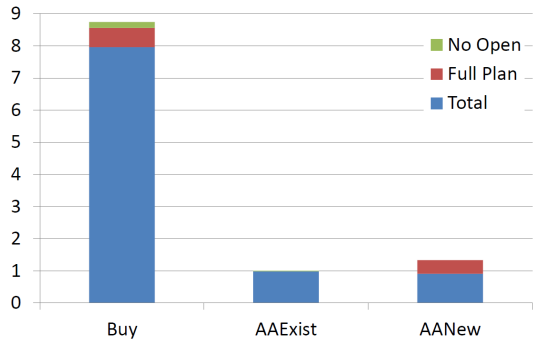

Figure 2: Average Number of Explanations by Type.

the number of possible explanations is exponential with the number of observations [13]. For example, even with the pruning of CRADLE, the average number of explanations it outputted for the AANew sessions is 146.06 (compared to 1.33 with the preprocessing). Moreover, the outputted explanations have a structure imposed by the plan library that the original stream lacked.

The second point to notice it that the values of the AAExist case are similar in both algorithms. We attribute this to the fact that performing this task is always performed in the same fashion, without any options to bias in the interface of the company's website.

6 Future Work

This a preliminary work for using plan recognition for exploratory environments on real-world click streams. It uses a novel approach for plan recognition from barebone UI data, which reasons about the plan library in the lowest recognition level in order to define the relevancy of actions in our domain, and then uses it to perform plan recognition.

While we manage to process low level data using mostly tools intended for higher levels of inference, there is still much to be done from both ends of the process: First, we expect that we can use most intelligent tools for the preprocessing stage, either from the world of activity recognition or natural language processing. As we started to show here, we wish to use the domain knowledge (e.g. the plan library) in our low-level action detection. Second, we wish to visualize this information in a coherent manner that will allow

	Buy	AAExist	AANew
Session Entries	147.02	14.13	80.33
Observations	5.64	2.06	3.33
CRADLE Explanations	8.74	1.00	1.33
PHATT Explanations	18.32	1.00	2.00
CRADLE Time (seconds)	0.06	0.01	0.01
PHATT Time (seconds)	0.07	0.01	0.03

Table 1: Runtime and Explanation Set Size for CRADLE and PHATT

analysts evaluate sessions in real time for fast verification and validation, or to be able to counter adversarial behavior in time.

References

[1] O. Amir and Y. Gal. Plan recognition and visualization in exploratory learning environments. *ACM Transactions on Interactive Intelligent Systems*, 3(3):16:1–23, 2013.

[2] L. Bao and S. S. Intille. Activity recognition from user-annotated acceleration data. In *International Conference on Pervasive Computing*, pages 1–17. Springer, 2004.

[3] T. V. Duong, H. H. Bui, D. Q. Phung, and S. Venkatesh. Activity recognition and abnormality detection with the switching hidden semi-markov model. In *Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 838–845. IEEE, 2005.

[4] R. G. Freedman and S. Zilberstein. Integration of planning with recognition for responsive interaction using classical planners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[5] Y. Gal, S. Reddy, S. Shieber, A. Rubin, and B. Grosz. Plan recognition in exploratory domains. *Artificial Intelligence*, 176(1):2270 – 2290, 2012.

[6] C. W. Geib and R. P. Goldman. Plan recognition in intrusion detection systems. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*, volume 1, pages 46–55. IEEE, 2001.

[7] R. Granada, R. F. Pereira, J. Monteiro, D. D. Ruiz, R. C. Barros, and F. Meneguzzi. Hybrid activity and plan recognition for video streams. In *Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR)*, 2017.

[8] N. Y. Hammerla, S. Halloran, and T. Ploetz. Deep, convolutional, and recurrent models for human activity recognition using wearables. *arXiv preprint arXiv:1604.08880*, 2016.

[9] P. A. Jarvis, T. F. Lunt, and K. L. Myers. Identifying terrorist activity with ai plan recognition technology. *AI Magazine*, 26(3):73, 2005.

[10] H. A. Kautz. *A formal theory of plan recognition.* PhD thesis, University of Rochester, 1987.

[11] L. Liao, D. Fox, and H. Kautz. Location-based activity recognition. *Advances in Neural Information Processing Systems*, 18:787, 2006.

[12] O. Madani, H. H. Bui, and E. Yeh. Prediction and discovery of users' desktop behavior. In *AAAI Spring Symposium: Human Behavior Modeling*, pages 49–55, 2009.

[13] R. Mirsky, Y. K. Gal, and S. Shieber. Cradle: An online plan recognition algorithm for exploratory domains. In *ACM Transactions on Intelligent Systems and Technology*, 2017.

[14] S. Natarajan, H. H. Bui, P. Tadepalli, K. Kersting, and W.-K. Wong. Logical hierarchical hidden markov models for modeling user activities. In *International Conference on Inductive Logic Programming*, pages 192–209. Springer, 2008.

[15] X. Qin and W. Lee. Attack plan recognition and prediction using causal networks. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 370–379. IEEE, 2004.

[16] K. Talamadupula, G. Briggs, T. Chakraborti, M. Scheutz, and S. Kambhampati. Coordination in human-robot teams using mental modeling and plan recognition. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2957–2962. IEEE, 2014.

[17] O. Uzan, R. Dekel, O. Seri, and Y. Gal. Plan recognition for exploratory learning environments using interleaved temporal search. *AI Magazine*, 36(2):10–21, 2015.

[18] K. Yordanova. Texttohbm: A generalised approach to learning models of human behaviour for activity recognition from textual instructions for activity recognition from textual instructions. In *Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR)*, 2017.

RADAR - A Proactive Decision Support System for Human-in-the-Loop Planning

Sailik Sengupta and **Tathagata Chakraborti** and **Sarath Sreedharan**
Satya Gautam Vadlamudi and **Subbarao Kambhampati**

School of Computing, Informatics, and Decision Systems Engineering
Arizona State University, Tempe AZ 85281 USA

{sailiks,tchakra2,ssreedh3}@asu.edu, gautam.vadlamudi@capillarytech.com, rao@asu.edu

Abstract

Proactive Decision Support (PDS) aims at improving the decision making experience of *human decision makers* by enhancing both the quality of the decisions and the ease of making them. In this paper, we ask the question what role automated decision making technologies can play in the deliberative process of the human decision maker. Specifically, we focus on expert humans in the loop who now share a detailed, if not complete, model of the domain with the assistant, but may still be unable to compute plans due to cognitive overload. To this end, we propose a PDS framework RADAR based on research in the automated planning community that aids the human decision maker in constructing plans. We will situate our discussion on principles of interface design laid out in the literature on the degrees of automation and its effect on the collaborative decision making process. Also, at the heart of our design is the principle of *naturalistic decision making* which has been shown to be a necessary requirement of such systems, thus focusing more on providing suggestions rather than enforcing decisions and executing actions. We will demonstrate the different properties of such a system through examples in a fire-fighting domain, where human commanders are involved in building response strategies to mitigate a fire outbreak. The paper is written to serve both as a position paper by motivating requirements of an effective proactive decision support system, and also an emerging application of these ideas in the context of the role of an automated planner in human decision making, in a platform that can prove to be a valuable test bed for research on the same.

Human-in-the-loop planning or HILP (Kambhampati and Talamadupula 2015) is a necessary requirement today in many complex decision making or planning environments. In this paper, we consider the case of HILP where the human responsible for making the decisions in complex scenarios are supported by an automated planning system. High-level information fusion that characterizes complex long-term situations and support planning of effective responses is considered the greatest need in crisis-response situations (Laskey, Marques, and da Costa 2016). Indeed, automated planning based proactive support was shown to be preferred by humans involved in teaming with robots (Zhang et al. 2015) and the cognitive load of the subjects involved was observed to have been reduced (Narayanan et al. 2015).

We note that the humans are in the driver's seat in generating plans.We investigate the extent to which an automated planner can support the humans in planning, despite not having access to the complete domain and preference models. This is appropriate in many cases, where the human in the loop is ultimately held responsible for the plan under execution and its results. This is in contrast to earlier work on systems such as TRAINS and MAPGEN (Allen 1994; Ai-Chang et al. 2004), where the planner is in the drivers seat, with the humans "advising" the planner. It is also a far cry from the earlier work on mixed-initiative planning where humans enter the land of automated planners and manipulate their internal search data structures. In our framework, the planners have to enter the land of humans.

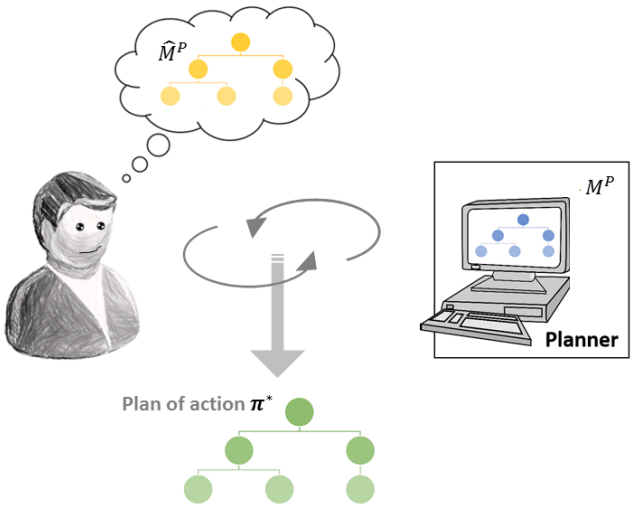

Figure 1: Planning for decision support involves iterative and the need to consider difference of models between the planner and the human in the loop.

An important complication arises due to the fact that the planner and the human can have different (possibly complementary) models of the same domain or knowledge of the problem at hand, as shown in Figure 1. In particular, humans might have additional knowledge about the domain as well as the plan preferences that the automated planner is not privy to. This means that plan suggestions made by the automated planner may not always make sense to the human in the loop, i.e. appear as suboptimal in her domain. This

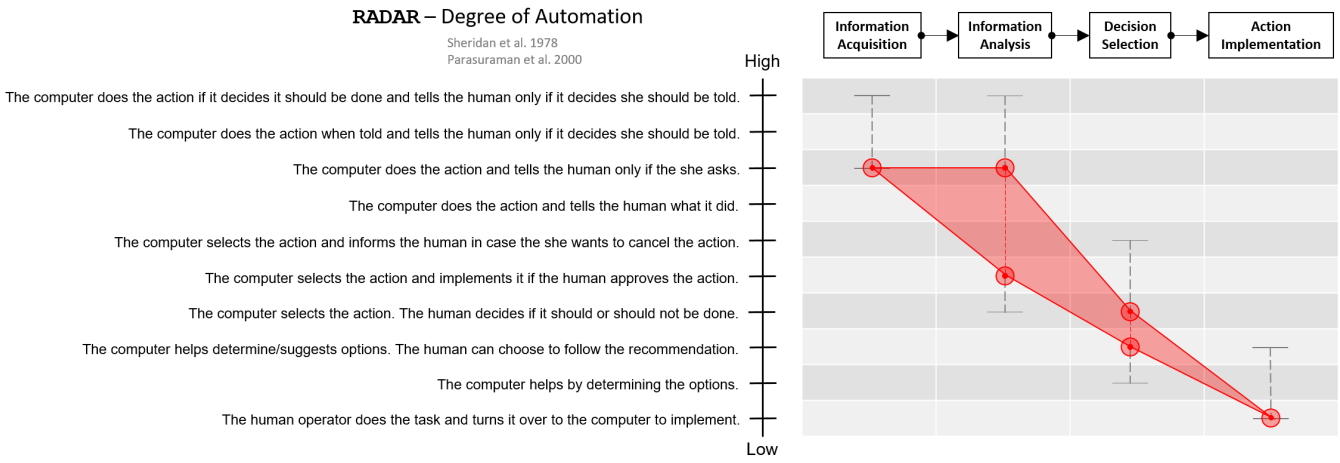

Figure 2: Degrees of automation of the various stages of decision support, and the role of RADAR in it.

can occur either when the human or the planner has a faulty model of the world. This is an ideal opportunity to provide model updates or explanations and reconcile this model difference through iterative feedback from the human. This calls for active participation from the human in the loop rather than simply adopting a system generated plan.

Though having to deal with an incomplete model is the usual case in many mixed initiative settings, i.e. an automated support component, without a full model, cannot actually generate entire plans from scratch but can sometimes complete or critique existing ones - the extent to which a planner can be of help is largely dependent on the nature of the model that is available. Keeping this in mind, in the current paper we focus on scenarios which come with more well-defined protocols or domain models, and illustrate how off-the-shelf planning techniques may be leveraged to provide more sophisticated decision support. Examples where such technologies can be helpful include any complex tasks, especially disaster response or emergency situations, where the mental overload of the human (either due to the complexity of the problem at hand or the sheer volume of data that needs to be considered to make an informed decision) can affect the quality of successful recovery.

To this end, we propose a proactive decision support (PDS) system RADAR following some of the design principles laid out in the literature in the human-computer interface community, to demonstrate possible roles that existing automated planning technologies can play in the deliberative process of the human decision maker in terms of the degree of automation of the planning process it affords.

Naturalistic Decision Making The proposed proactive decision support system supports *naturalistic decision making* (NDM), which is a model that aims at formulating how humans make decisions is complex time-critical scenarios (Zsambok and Klein 2014; Klein 2008). It is acknowledged as a necessary element in PDS systems (Morrison et al. 2013). Systems which do not support NDM have been found to have detrimental impact on work flow causing frustration to decision makers (Feigh et al. 2007). At the heart of this concept is, as we discussed before, the requirement of letting the human be in control. This motivates us to build a proactive decision support system, which focuses on aiding and alerting the human in the loop with his/her decisions rather than generate a static plan that may not work in the dynamic worlds that the plan has to execute in. In cases when the human wants the planner to generate complete plans, he still has the authority to ask RADAR to explain its plan when it finds it to be inexplicable (Chakraborti et al. 2017). We postulate that such a system must be augmentable, context sensitive, controllable and adaptive to the humans decisions. Various elements of human-automation interaction such as, adaptive nature and context sensitivity are presented in (Sheridan and Parasuraman 2005). (Warm, Parasuraman, and Matthews 2008) show that vigilance requires hard mental work and is stressful via converging evidence from behavioral, neural and subjective measures. Our system may be considered as a part of such vigilance support thereby reducing the stress for the human.

Degrees of Automation One of the seminal works by (Sheridan and Verplank 1978), builds a model that enumerates ten levels of automation in software systems depending on the autonomy of the automated component. Later, in the study of mental workload and situational awareness of humans performing alongside automation software, (Parasuraman 2000) separates automation into four parts- Information Acquisition, Information Analysis, Decision Selection and Action Implementation (see Figure 2). We use this system as an objective basis for deciding which functions for our system should be automated and to what extent so as to reduce human's mental overload while supporting Naturalistic Decision making. (Parasuraman and Manzey 2010) shows that human use of automation may result in automation bias leading to omission and commission errors, which underlines the importance of reliability of the automation (Parasuraman and Riley 1997). Indeed, it is well known (Wickens et al. 2010), that climbing the automation ladder in Figure 2 might well improve operative performance but drastically decrease the response to failures or mistakes. Hence, to meet

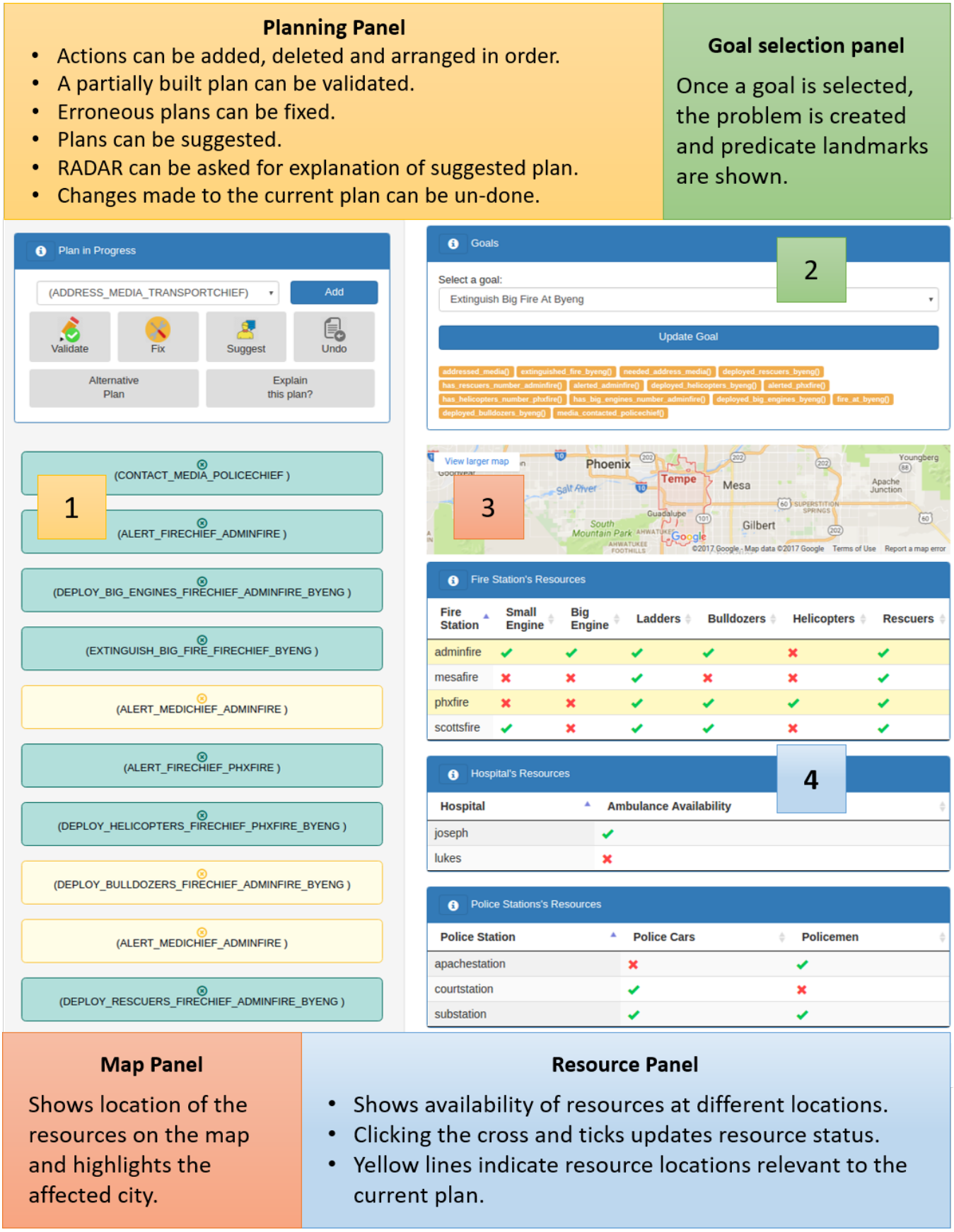

Figure 3: RADAR interface showing decision support for the human commander making plans in response to a fire.

the requirement of naturalistic decision making, we observe a downward trend in automation levels (in Figure 2) as we progress from data acquisition and analysis (which machines are traditionally better at) to decision making and execution.

Interpretation & Steering For the system to collaborate with the commanders effectively, in the context of a mixed-initiative setting,[1] it must have two broad capabilities - *Interpretation* and *Steering* (Manikonda et al. 2014). Interpretation means understanding the actions done by the commanders, while steering involves helping the commanders to do their actions. Interpretation involves, for instance, extraction of sub-goals from the task description, to be addressed in the situation, or recognizing what specific activities that the commanders are up to, in order to reason with its own internal model, or recognizing the plans that the commanders are intending to execute, to provide automatic explanation and awareness to the collaborating agents. Steering can involve suggesting new actions to guide the planning process. This can be done either by generating a plan based on the available resources, and outstanding sub-goals and constraints, or by recognizing the plans of the commanders and helping them fulfill their goals. Steering also involves assessing the currently executed plan and critiquing parts of it, which might need further attention due to insufficient resources or failed execution. For example, the system can throw an alert that the plan under construction fails due to insufficient beds available at the chosen hospital, and provide possible alternatives to the commander. The current system mainly addresses the decision making aspect, which requires the ability to both interpret as well as steer effectively, even as it situates itself in the level of automation it can provide in the context of naturalistic decision making.

RADAR

We will now go into details of the RADAR interface and its integration with planning technologies to enable different forms of proactive decision support. A video walkthrough demonstrating the different capabilities of the system is available at https://goo.gl/YunA21.

The Fire-fighting Domain For the remainder of the discussion, we will use a fire-fighting scenario to illustrate our ideas. The domain model used by the system (assumed to be known and available for a well-defined task such as this) is represented in PDDL (McDermott et al. 1998) and is assumed to be very close, if not identical, to that of the expert in the loop. The scenario plays out in a particular location (we use Tempe as a running example) and involves the local fire-fighting chief, who along with the local police, medical and transport authorities, is trying to build a plan in response to the fire using the given platform augmented with decision support capabilities. The PDDL domain file and a problem scenario can be found at https://goo.gl/htrmLQ.

Overview of the Interface The interface consists of four main components, as shown in Figure 3. This includes -

(1) **Planning Panel** - This is the most critical part of the system. It displays the plan under construction, and provides the human with abilities to reorder / add / delete actions in the plan, validate a partial plan, fix a broken plan, suggest new better ones, provide explanation on the current one, etc. by accessing the options at the top of the panel. This will be the primary focus for our discussion in the upcoming sections.

(2) **Goal Selection Panel** - This lets the user set high level goals or tasks to be accomplished (e.g. "Extinguish fire at BYENG"). Once a goal is selected, the system sets up the corresponding planning problem instance given its knowledge of the then state of the world. It also summarizes this task to the user by displaying the necessary landmarks to be attained in order to achieve the goal.

(3) **Map Panel** - This provides visual guidance to the decision making process, thereby reducing the information overload and improving the situational awareness of the human. The map can be used to point of areas of interest, location and availability of resources, routes, etc. Note that this part of the UI can also be used to display other relevant information for different domains by simply changing a template file.

(4) **Resource Panel** - The human commanders have access to the resources that they can use to control the fire outbreak (as can be seen from the tables to the right in Fig. 3). For example, the police can deploy police cars and policemen, and the fire chief can deploy fire engines, ladders, rescuers, etc. if available. They can also acquire or update the availability of these on the go by clicking on the red crosses or green tick respectively, if the system's data is stale. The system also highlights parts of the table that are relevant to the plan currently under construction.

These plans are valid, of course, depending on the availability of the appropriate resources introduced above, and certain actions can only be executed when the required preconditions are satisfied. For example, in order to dispatch police cars from a particular police station, the police chief needs to make sure that the respective police station has enough police cars and it has been notified of the demand previously. Given this knowledge, RADAR keeps an eye on the planning process of the human commanders to make sure that the partial plan build is likely to succeed in achieving the goal going forward. In the following sections, we will see how it can achieve this, using techniques from the automated planning community, yielding different stages of automation of the decision support process.

Information Acquisition

For effective decision support, the importance of data cannot be understated. While on one hand it must support proactive data retrieval and integration capabilities, it must also have abilities to generate and recognize plans, and support the decision-making tasks of the commanders, with the help of this data. Thus, PDS can be seen to consist of two main capabilities, *data driven decision-making* and *decision driven data-gathering*. We call this the **Data-Decision Loop**.

[1] Note that traditional notions of mixed-initiative planning represent systems where the human helps the automated planner. In our case, it is the opposite where the planner helps the human.

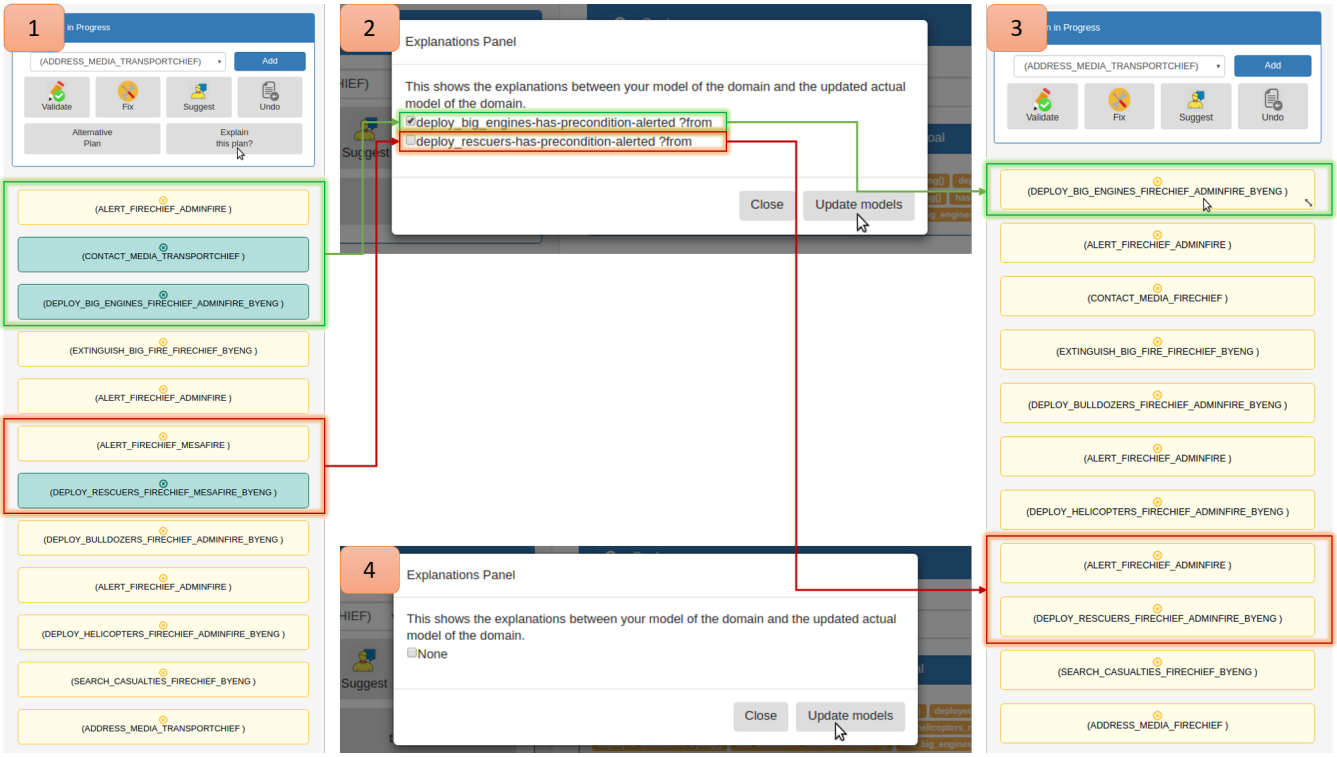

Figure 4: (1) RADAR knows that in the environment, the commander needs to inform the Fire Station's Fire chief before deploying big engines and rescuers. In green, Adminfire's Fire Chief is alerted to deploy big engines from Admin Fire Station. In red, Mesa fire stations' Fire Chief is alerted to deploy rescuers from Mesa Fire Station. (2) The human's model believes that there is no need to inform Fire Chiefs and questions RADAR to explain his plan. RADAR finds these differences in the domain model and reports it to the human. The human acknowledges that before deploying rescuers one might need to alert the Fire Chief and rejects the update the Fire Chief needs to be alerted before deploying big engines. (3) In the alternative plan suggested by RADAR, it takes into account the humans knowledge and plans with the updated model. (4) Clicking on 'Explain This Plan' generates no explanations as there are none (with respect to the current plan) after the models were updated.

In the current version, we assume that RADAR acquires relevant information regarding the availability of resources pertaining to the task at hand. We will also assume that the system can keep track of drifting models (Bryce, Benton, and Boldt 2016) in the background. This firmly places it in Degree 7 of automation. While we cannot expect the human to gather data for the system (after all, the entire purpose of the system is to reduce the cognitive load due to an excess of data), the system can ostensibly choose to acquire but not display the irrelevant information at all, and climb up to Degree 10. *In the current version of the system, we do not integrate any data sources yet, but instead only focus on the decision making aspect in the next upcoming sections.* We discuss briefly about the salient challenges of the information acquisition in the section on future works.

Information Analysis

Now, we will present details on how the proposed system can leverage planning technologies to provide relevant suggestions and alerts to the human decision maker with regards to the *information needed to solve the problem*. The planning problem itself is given by $\Pi = \langle M, \mathcal{I}, \mathcal{G} \rangle$ where M is the action model, and $\mathcal{I}, \mathcal{G}$ are the current and goal states representing the current context and task description respectively. Finally the plan $\pi = \pi_e \circ \pi_h \circ \pi_s$ is the solution to the planning problem, which is represented as concatenation of three sub-plans - π_e is the plan fragment that the commander has already deployed for execution, and π_h is the set of actions being proposed going forward. Of course, these two parts by themselves might not achieve the goal, and this is the role of the plan suffix π_s that is yet to be decided upon. We will demonstrate below how planning technology may be used to shape each of these plan fragments for the better.

Model Updates. As an augmentable system, the system must support update to the rules that govern its decision support capabilities, as required by the user, or by itself as it interacts with the environment. Of course, such models may also be learned (Zhuo, Nguyen, and Kambhampati 2013) or updated (Bryce, Benton, and Boldt 2016) on the fly in cases of failures during execution of π_h or actions of the human in response to excuses generated from the system, or to account for model divergence due to slowly evolving conditions in the environment. Further, the system should be, if possible, act in a fashion that is easily understandable to the human in

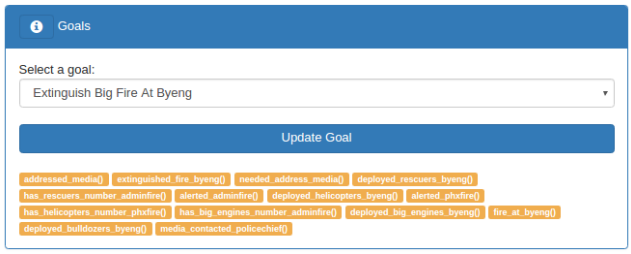

Figure 5: Once a goal is selected, the problem file is generated and the landmarks are computed to help the commander be on track to achieve the goal.

the loop (Zhang et al. 2016), or be able to explain the rationale behind its suggestions if required (Kambhampati 1990; Sohrabi, Baier, and McIlraith 2011). Finally, such explanations need to conveyed in a fashion that is easily received or understood by the human user (Perera et al. 2016).

Often a key factor in these settings is the difference in the planner's model of the domain, and the human expectation of it. Thus, a valid or satisfactory explanation may require a *model reconciliation process* where the human model needs to be updated, as shown in Figure 4 in order to explain a suggestion. Here the system performs model-space search to come up with *minimal explanations* that explain the plan being suggested while at the same time not overloading the human with information not relevant to the task at hand (refer to (Chakraborti et al. 2017) for more details). Note that here the human has the power to veto the model update if (s)he believes that the planner's model is the one which is faulty, by choosing to approve or not approve individual parts of the explanation provided by the system. Thus, the system here displays Degree 5 of automation.

Plan Summarization. As we mentioned before, when a task or high level goal is selected by the human, RADAR automatically generates the corresponding planning problem in the background, analyses the possible solution to it, and highlights resources required for it to give the human an early heads-up. It can, however, do even more by using landmark analysis of the task at hand to find bottlenecks in the future. Briefly, landmarks (Hoffmann, Porteous, and Sebastia 2004) are (partial) *states* such that *all* plans that can accomplish the tasks from the current state must go through it during their execution, or *actions* that *must* be executed in order to reach the goal. These are referred to as state landmarks and action landmarks respectively. Clearly, this can be a valuable source of guidance in terms of figuring out what resources and actions would be required in future, and may be used to increase the decision maker's situational awareness by summarizing the task at hand and possible solutions to it in terms of these landmarks. In the current system, we use the approach of (Zhu and Givan 2003) for this purpose. Figure 5 illustrates one such use case, where the system automatically computes and displays the landmarks after the human selects the goal, thus exhibiting characteristics of Degree 7 automation of information analysis.

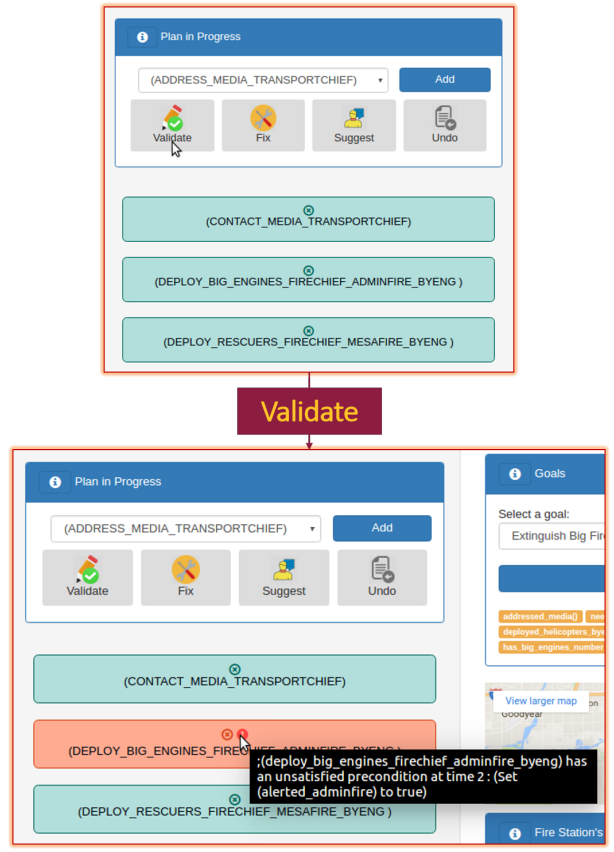

Figure 6: RADAR does plan validation of a partial plan made by the user and shows reasons as to why it is invalid.

Plan Validation Plan failure occurs when the plan fragment π_e that has already been dispatched for execution and/or the sub-plan π_h currently under construction are not valid plans, i.e. $\delta(\mathcal{I}, \pi_e \circ \pi_h) \models \perp$. From the point of view of planning, this can occur due to several reasons, ranging from unsatisfied preconditions to incorrect parameters, to the model itself being incorrect or incomplete. Errors made in π_h that can be explained by the model can be easily identified using plan validators like VAL (Fox, Howey, and Long 2005; Howey, Long, and Fox 2004), while errors in π_e should be used as feedback (context-sensitive) so that the system, in looking forward, may have to re-plan (adaptive) from a state $s \neq \delta(\mathcal{I}, \pi_e)$.

Of course, the goal itself may be unreachable given the current state (for example, due to insufficient resources). This can be readily detected via *reachability analysis* using *planning graph* techniques. This is supported by most planners, including Fast-Downward (Helmert 2006). Once the system detects a state with no solution to the planning problem, apart from alerting the human to this situation itself, it can choose to suggest an alternative state $\mathcal{I}^*$ where a solution does exist, i.e. $\exists \pi$ s.t. $\delta(\mathcal{I}^*, \pi) \models \mathcal{G}$. This can provide guidance to the human in how to fix the problem in situations beyond the system's control/knowledge, and may be achieved using *excuse generation* techniques stud-

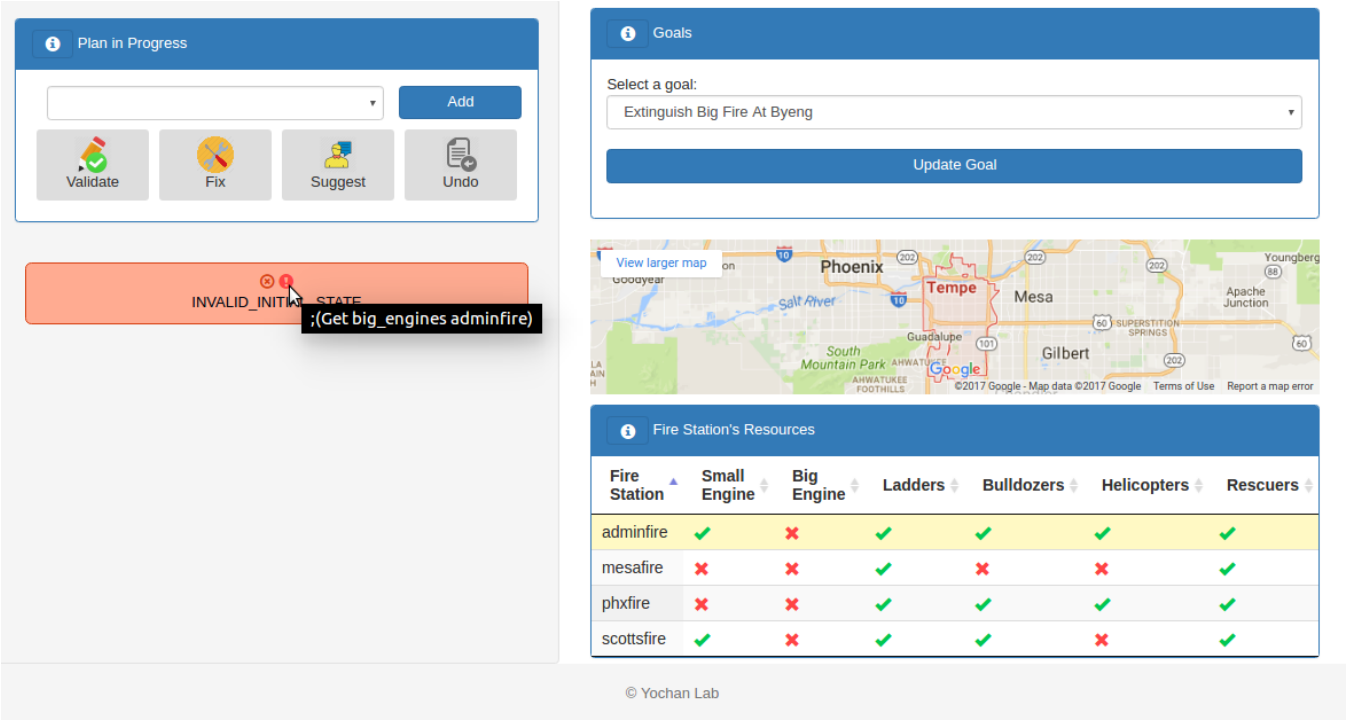

Figure 7: The lack of big engines at all the fire stations results in an initial state for the planning problem from which no plan is possible to achieve the goal of Extinguishing Big Fire at BYENG. RADAR reports this as a warning and suggests the minimal number of resources the commander needs to gather to arrive at a start state from which a plan is actually possible.

ied in (Göbelbecker et al. 2010) and *plan revision* problems (Herzig et al. 2014). We achieved this using a slightly modified version of the model-space search technique introduced by (Chakraborti et al. 2017) - here the faulty model is replaced with a initial state with all resources available, and a minimum distance to it is computed to guarantee feasibility.

Decision Selection

The decision selection process is perhaps closest to home for the planning community. Referring back to our discussion on naturalistic decision making, and the need for on-demand support, we note that the system is mostly restricted to Degree 3 and 4 of automation with respect to decision selection. We will go through some salient use cases below.

Plan Correction or Repair In the event π_h is invalid and may be repaired with additional actions, we can leverage the compilation `pr2plan` from (Ramírez and Geffner 2010) for a slightly different outcome. The compilation, originally used for plan recognition, updates the current planning problem Π to $\Pi^* = \langle M^*, \mathcal{I}^*, \mathcal{G}^* \rangle$ using π_h as a set of *observations* such that $\forall a \in \pi_h$ is *preserved in order* in the (optimal) solution π of Π^*. The actions that occur in between such actions in the solution π to the compilation may then be used as suggestions to the user to fix the currently proposed plan π_h. Figure 8 illustrates one such use case, demonstrating Degree 3 of automation - i.e. the system only complements the decision process when asked, and provides the human an option to undo these fixes at all times. Note that since the deployed actions are required to be preserved (and the suggested actions preferably so) when looking ahead in the plan generation process, we will use Π^* for all purposes going forward.

Action Suggestions The most basic mode of action suggestion would be to solve the current planning problem Π^* using an optimal planner such as `Fast-Forward` (Helmert 2006) and suggest the plan suffix π_s as the best course of action. Of course, the actions suggested by the commander in π_h may themselves be part of a sub-optimal plan and may thus be improved upon. Here we again use an existing compilation from (Ramírez and Geffner 2010) for a slightly different purpose than originally intended. Given a known goal, we find out if the choice $a \in \pi_h$ is sub-optimal using the difference in cost $\Delta = C(\hat{\pi}) - C(\pi)$ where $\hat{\pi}$ is the solution to the planning problem $\langle M^*, \mathcal{I}^*, \mathcal{G}^* + a \rangle$ as given by `pr2plan`. This is again shown in Figure 8.

Monitoring Plan Generation Of course (Ramírez and Geffner 2010) may be used also for its intended purpose. In cases where there are multiple ways to achieve the goal, and the system is not aware of the user's implicit preferences $\mathcal{P}$, `pr2plan` can be used to compile the goal into $\mathcal{G}^* \leftarrow \mathcal{G}^* + \mathcal{P}$ and check for correctness or likelihood $P(\mathcal{G}|\pi_e \circ \pi_h)$ of the current hypothesis. This is implicitly used by RADAR in determining the response to suggest or fix any hypothesis, as described before. The lack of alternative goals or tasks in the present context somewhat limit the scope of traditional goal, as opposed to plan, recognition.

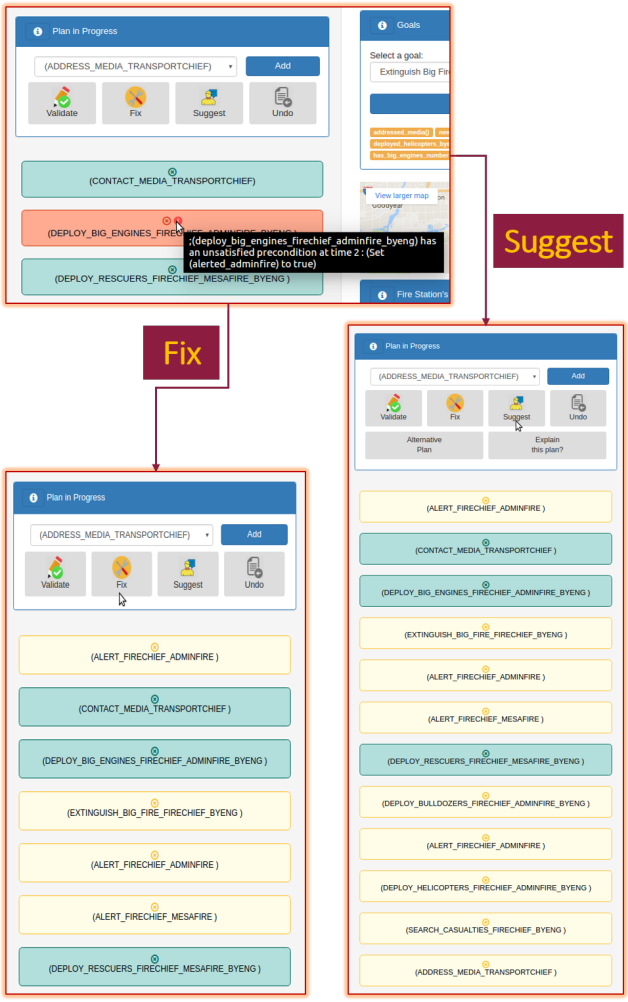

Figure 8: RADAR's 'Fix' button does plan correction, providing action suggestions. The 'Suggest' provides actions and plan suggestions to help achieve the goal.

Plan Suggestions One useful way of increasing the situational awareness of the human decision maker is to make him/her aware of the different, often diverse, choices available. Currently, when asked for alternative plans, RADAR provides an optimal plan as a suggestion. This may not be always desired. Specifically, with the existence of *disjunctive landmarks* (i.e. landmarks such that presence of any one of them are sufficient for existence of a valid plan), just alerting the commander of the landmarks may not be enough to tell how they contribute to the planning choices. In such cases, the concept of *diverse plans* (Srivastava et al. 2007; Nguyen et al. 2012) and *top-K plans* (Riabov, Sohrabi, and Udrea 2014) become useful. We are exploring avenues of integrating these techniques into our current system.

Action Implementation

Going back to our previous discussion on naturalistic decision making, we reiterate the need to let the human decision maker make the final call at execution time. In the case of current system, the platform does not provide any endpoints to external facilities and thus lies at Degree 1 of automation in the Action Implementation phase. Some of these tasks can however be automated - e.g. in our fire-fighting domain the human can delegate the tasks for alerting police-stations and fire-stations to be auto-completed. Thus RADAR can ostensibly range from Degree 1 to a maximum of 6 in the final Action Implementation phase. However, given how often such systems have been known to fail to capture the exact context and complexity of these scenarios, including some of the mixed initiative schedulers from NASA, the final execution phase is often times just left to the human operators completely, or at least firmly at the lower spectrum of the automation scale. Recent attempts (Gombolay et al. 2015) at *learning* such preferences in mixed-initiative schedulers might provide interesting insights into climbing the automation levels at the final stage of decision support for planning, without significant loss of control.

Conclusion and Future Work

In conclusion, we motivated the use of automated planning techniques in the role of an assistant in the deliberative process of an expert human decision maker, and provided a detailed overview of our platform RADAR to demonstrate different ways this can be achieved. We also showed how these capabilities complement the design principles laid out in the human computer interface community for such softwares. We look forward to conducting human studies with domain experts to evaluate the effectiveness of the system.

For future work, integration of data sources remains one of the key priorities. Although our system can provide information on the resources useful to the plan, it can be more proactive in providing information that might be needed in the future, based on the plans it recognizes. We believe a tight integration of the data-driven decision-making and decision-driven data-gathering loop will be crucial to the success of decision support systems such as RADAR.

Acknowledgments This research is supported in part by the NASA grant NNX17AD06G and the ONR grants N00014161-2892, N00014-13-1-0176, N00014-13-1-0519, N00014-15-1-2027.

References

[Ai-Chang et al. 2004] Ai-Chang, M.; Bresina, J.; Charest, L.; Chase, A.; Hsu, J.-J.; Jonsson, A.; Kanefsky, B.; Morris, P.; Rajan, K.; Yglesias, J.; et al. 2004. Mapgen: mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems* 19(1):8–12.

[Allen 1994] Allen, J. F. 1994. Mixed initiative planning: Position paper. In *ARPA/Rome Labs Planning Initiative Workshop*.

[Bryce, Benton, and Boldt 2016] Bryce, D.; Benton, J.; and Boldt, M. W. 2016. Maintaining evolving domain models. In *IJCAI*.

[Chakraborti et al. 2017] Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *IJCAI*.

[Feigh et al. 2007] Feigh, K. M.; Pritchett, A. R.; Jacko, J. A.; and Denq, T. 2007. Contextual control modes during an airline rescheduling task. *Journal of Cognitive Engineering and Decision Making* 1:169–185.

[Fox, Howey, and Long 2005] Fox, M.; Howey, R.; and Long, D. 2005. Validating plans in the context of processes and exogenous events. In *AAAI*, 1151–1156.

[Göbelbecker et al. 2010] Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming up with good excuses: What to do when no plan can be found. *Cognitive Robotics*.

[Gombolay et al. 2015] Gombolay, M.; Gutierrez, R.; Clarke, S.; Sturla, G.; and Shah, J. 2015. Decision-making authority, team efficiency and human worker satisfaction in mixed human-robot teams. *Autonomous Robots* 39(3):293–312.

[Helmert 2006] Helmert, M. 2006. The fast downward planning system. *Journal for Artificial Intelligence Research (JAIR)*.

[Herzig et al. 2014] Herzig, A.; Menezes, V.; de Barros, L. N.; and Wassermann, R. 2014. On the revision of planning tasks. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, ECAI'14.

[Hoffmann, Porteous, and Sebastia 2004] Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *J. Artif. Int. Res.* 22(1):215–278.

[Howey, Long, and Fox 2004] Howey, R.; Long, D.; and Fox, M. 2004. Val: automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*, 294–301.

[Kambhampati and Talamadupula 2015] Kambhampati, S., and Talamadupula, K. 2015. Human-in-the-loop planning and decision support. In *AAAI Tutorial*.

[Kambhampati 1990] Kambhampati, S. 1990. A classification of plan modification strategies based on coverage and information requirements. In *AAAI 1990 Spring Symposium on Case Based Reasoning*.

[Klein 2008] Klein, G. 2008. Naturalistic decision making. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 50(3):456–460.

[Laskey, Marques, and da Costa 2016] Laskey, K. B.; Marques, H. C.; and da Costa, P. C. G. 2016. *High-Level Fusion for Crisis Response Planning*. Cham: Springer International Publishing.

[Manikonda et al. 2014] Manikonda, L.; Chakraborti, T.; De, S.; Talamadupula, K.; and Kambhampati, S. 2014. Ai-mix: Using automated planning to steer human workers towards better crowdsourced plans. In *IAAI*, 3004–3009. AAAI Press.

[McDermott et al. 1998] McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. Pddl-the planning domain definition language.

[Morrison et al. 2013] Morrison, J. G.; Feigh, K. M.; Smallman, H. S.; Burns, C. M.; and Moore, K. E. 2013. The quest for anticipatory decision support systems (panel). *Human Factors and Ergonomics Society Annual Meeting* 1:169–185.

[Narayanan et al. 2015] Narayanan, V.; Zhang, Y.; Mendoza, N.; and Kambhampati, S. 2015. Automated planning for peer-to-peer teaming and its evaluation in remote human-robot interaction. In *HRI Extended Abstracts*, 161–162. ACM.

[Nguyen et al. 2012] Nguyen, T. A.; Do, M.; Gerevini, A. E.; Serina, I.; Srivastava, B.; and Kambhampati, S. 2012. Generating diverse plans to handle unknown and partially known user preferences. *Artif. Intell.* 190:1–31.

[Parasuraman and Manzey 2010] Parasuraman, R., and Manzey, D. H. 2010. Complacency and bias in human use of automation: An attentional integration. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 52(3):381–410.

[Parasuraman and Riley 1997] Parasuraman, R., and Riley, V. 1997. Humans and automation: Use, misuse, disuse, abuse. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 39(2):230–253.

[Parasuraman 2000] Parasuraman, R. 2000. Designing automation for human use: empirical studies and quantitative models. *Ergonomics* 43(7):931–951.

[Perera et al. 2016] Perera, V.; Selvaraj, S. P.; Rosenthal, S.; and Veloso, M. 2016. Dynamic Generation and Refinement of Robot Verbalization. In *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication*.

[Ramírez and Geffner 2010] Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *AAAI*.

[Riabov, Sohrabi, and Udrea 2014] Riabov, A.; Sohrabi, S.; and Udrea, O. 2014. New algorithms for the top-k planning problem. In *Proceedings of the Scheduling and Planning Applications woRKshop (SPARK) at the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 10–16.

[Sheridan and Parasuraman 2005] Sheridan, T. B., and Parasuraman, R. 2005. Human-automation interaction. *Reviews of human factors and ergonomics* 1(1):89–129.

[Sheridan and Verplank 1978] Sheridan, T. B., and Verplank, W. L. 1978. Human and computer control of undersea teleoperators. Technical report, DTIC Document.

[Sohrabi, Baier, and McIlraith 2011] Sohrabi, S.; Baier, J. A.; and McIlraith, S. A. 2011. Preferred explanations: Theory and generation via planning. In *Proceedings of the 25th Conference on Artificial Intelligence (AAAI-11)*, 261–267.

[Srivastava et al. 2007] Srivastava, B.; Nguyen, T. A.; Gerevini, A.; Kambhampati, S.; Do, M. B.; and Serina, I. 2007. Domain independent approaches for finding diverse plans. In *IJCAI*, 2016–2022.

[Warm, Parasuraman, and Matthews 2008] Warm, J. S.; Parasuraman, R.; and Matthews, G. 2008. Vigilance requires hard mental work and is stressful. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 50(3):433–441.

[Wickens et al. 2010] Wickens, C. D.; Li, H.; Santamaria, A.; Sebok, A.; and Sarter, N. B. 2010. Stages and levels of automation: An integrated meta-analysis. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 54, 389–393. SAGE Publications Sage CA: Los Angeles, CA.

[Zhang et al. 2015] Zhang, Y.; Narayanan, V.; Chakraborti, T.; and Kambhampati, S. 2015. A human factors analysis of proactive support in human-robot teaming. In *IROS*, 3586–3593.

[Zhang et al. 2016] Zhang, Y.; Sreedharan, S.; Kulkarni, A.; Chakraborti, T.; Zhuo, H. H.; and Kambhampati, S. 2016. Plan explicability for robot task planning. In *Proceedings of the RSS Workshop on Planning for Human-Robot Interaction: Shared Autonomy and Collaborative Robotics*.

[Zhu and Givan 2003] Zhu, L., and Givan, R. 2003. Landmark extraction via planning graph propagation. *ICAPS DC*.

[Zhuo, Nguyen, and Kambhampati 2013] Zhuo, H. H.; Nguyen, T.; and Kambhampati, S. 2013. Refining incomplete planning domain models through plan traces. In *IJCAI*, 2451–2457.

[Zsambok and Klein 2014] Zsambok, C. E., and Klein, G. 2014. Naturalistic decision making. *Psychology Press*.

Workflow Complexity for Collaborative Interactions: *Where are the Metrics?* – A Challenge

Kartik Talamadupula and **Biplav Srivastava** and **Jeffrey O. Kephart**
Human-Agent Collaboration Group
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
{krtalamad, biplavs, kephart} @ us.ibm.com

Abstract

In this paper, we introduce the problem of denoting and deriving the complexity of workflows (plans, schedules) in collaborative, planner-assisted settings where humans and agents are trying to jointly solve a task. The interactions – and hence the workflows that connect the human and the agents – may differ according to the domain and the kind of agents. We adapt insights from prior work in human-agent teaming and workflow analysis to suggest metrics for workflow complexity. The main motivation behind this work is to highlight metrics for human comprehensibility of plans and schedules. The planning community has seen its fair share of work on the synthesis of plans that take diversity into account – what value do such plans hold if their generation is not guided at least in part by metrics that reflect the ease of engaging with and using those plans?

1 Introduction

The emergence of the Internet and application (app) centric service-oriented platforms for various kinds of consumer tasks have resulted in an explosion in the interactions between humans and automated agents that assist them in tasks. Given their large number, a formal measurement of the inherent complexity of these interactions is desirable to assist in the design of useful and efficient decision-making algorithms and systems.

We present a usecase that illustrates the kinds of interactions we are discussing – consider a person living in New York who wants to book a travel itinerary for a short personal trip to Seattle. The workflow for planning this trip will consist of a flight reservation, hotel reservation, and reservation for local travel in the source and destination cities. These bookings could each be made via websites, over a dialog interface, via IoT interfaces, or manually over the phone with a travel agency. Each communication modality introduces its own constraints and complexity. We highlight the workflow complexity in this specific example using Figure 1. Here, three action instances are shown for booking a flight, a hotel, and local travel. The data artifacts are the booking confirmations, whose variables constrain the other actions in the workflow. In the workflow fragment that is shown, local travel at the destination is most constrained as it depends on the flight's arrival time, as well as the location of the hotel at the destination. In general, the flight booking will result in dates (and times) which create a dependency for the hotel reservation. Finally, flight and hotel reservations give the date and locations for which local travel needs to be booked. The overall complexity of booking this short leisure trip may differ from a business trip, where meeting schedules have to be taken into account; and may further differ from an international trip where the processing of travel documents has to be taken into account.

In such scenarios, automation faces two main challenges. The first is the problem of knowledge acquisition and engineering pertaining to the domain of interest – in the travel scenario above, this knowledge would constitute the various actions available to the agent to create a successful workflow, and the dependencies between those actions. This kind of problem is the purview of the flourishing Knowledge Engineering for Planning & Scheduling (KEPS) community. The second major problem is that of explaining the plan and the interactions underlying it to the (human) user/consumer of the plan. An important sub-problem in this is *measuring* the complexity of the said interaction – without such measures, an automated system that is trying to aid in such interactions will be unable to distinguish between and rank workflows of vastly differing complexities that all achieve the same goal. Complexity measures provide the ability to rank the planner's mediation in such scenarios, and allow the planner to produce directed help that will enable easier achievement of the user's goals. We highlight this second problem in this paper.

2 Prior Work

There is a rich body of work on workflow representation, composition, and execution (van der Aalst and van Hee 2004). Over the past decade, there have been approaches for semi or fully automated composition of workflows using planning that look at control and data driven issues (Srivastava and Koehler 2003). However, much of this work is in the context of single agent decision-making. There is no prior work, to our knowledge, that characterizes the complexity of workflows in a collaborative setting.

The planning community has also seen advances in the problems of measuring the distance between plans (Roberts *et al.* 2014; Goldman and Kuter 2015), and generating diverse plan alternatives (Nguyen *et al.* 2012). However, very

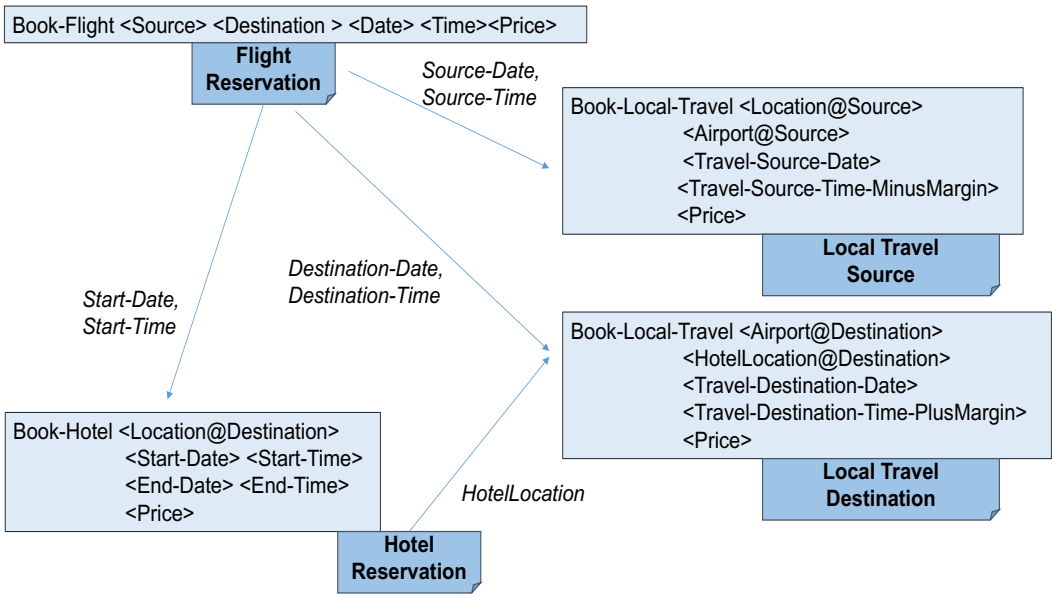

Figure 1: Actions, data (parameters) and constraint variables in a small travel example.

little research has focused on exactly what the different metrics that go into creating diverse plans should be. Such work has mostly looked at measures (cost, duration, robustness, etc.) that treat the plan as an artifact disconnected from humans, who must execute, understand, or participate in that workflow. Humans typically perceive complexity both from interaction issues, as well as from the actions in a workflow. Indeed, there is a long history of prior work from a linguistic and structural perspective for the former (Liao *et al.* 2017). However, there has been no focus on creating a class of metrics that attempt to define the complexity of a plan or workflow. We intend this paper as a challenge to the community to do exactly that.

3 Workflow Complexity: Example Usecases

We described the Travel Booking usecase in Section 1; here, we describe some other collaborative examples to highlight complexities that an automated decision making system can help reduce.

Scheduling Meetings A common collaborative task in the workplace is deciding a meeting time and venue, given a topic. This mundane task is complicated by the fact that there are different roles for participants in the meeting, hard and soft scheduling constraints, and limited access to participant information which changes with context. In such scenarios, setting up a meeting between colleagues who are at the same level organizationally may be more complex than one convened by the head of the organization – in the former there may be more hard constraints and various alternatives have to be considered, while in the latter everyone is likely to mark their (conflicting) constraints as soft. An automated agent (Cranshaw *et al.* 2017) can play a crucial role in improving the efficiency of meeting scheduling[1]. Specifically, it can verify participants and roles, identify potential conflicts from existing schedules, ask (the fewest number of) people to re-visit their constraints, and explain alternative time-slots.

Evaluating Hiring Choices Another workplace example is the evaluation of a set of candidates by a multi-disciplinary panel of experts. The experts may evaluate the candidate's technical skills, non-technical (soft) skills, organizational fit, HR concerns, career progression etc. Depending on the role, the process may involve many interview rounds, evaluations, and discussion. Further complexity is added by variations in the evaluation scales, disagreements among the experts, and relative weights of selection criteria. An automated agent can make this process more efficient by formalizing the contributions of the experts, focusing the team on key decision factors, retrieving relevant candidate data, eliminating human bias, and providing justifications to the stakeholders when asked.

Human-Robot Teaming Planning for human-robot teaming (HRT) (Talamadupula *et al.* 2010; Chakraborti *et al.* 2016b) considers the problem of humans and robots in goal-oriented environments, and the planner's mediation through control of the robotic agent. HRT scenarios usually involve extensive interaction between the human and the robot. Automated mediation can make the teaming more efficient in

[1]This is distinct from the actual scheduling problem, which is to find a satisfying assignment given everyone's constraints – our problem considers the workflow of scheduling the meeting.

METRIC / USECASE	NT	IT	AD	FO	Com	EC	PC	MC
Travel Booking	**L**	**H**	**H**	**H**	**H**	**H**	**M**	**M**
Scheduling Meetings	**H**	**M**	**M**	**L**	**H**	**L**	**H**	**H**
Evaluating Hiring Choices	**L**	**H**	**H**	**L**	**H**	**M**	**H**	**H**
Human-Robot Teaming	**M**	**H**	**M**	**M**	**L**	**M**	**L**	**L**
Medical Treatment	**H**	**L**	**L**	**L**	**H**	**M**	**H**	**H**
Personal Finance	**M**	**M**	**H**	**L**	**H**	**H**	**H**	**H**

Table 1: Workflow complexity metrics and their footprint; H - High, M - Medium, L - Low; Metrics described in Section 4.

a number of ways, including coordination to reduce communication (Talamadupula *et al.* 2014), and restricting the number of agents that a human has to deal with.

Deciding a Medical Treatment Plan Another illustrative collaborative task, from the area of health, is deciding a medical plan for a person given a health condition (initial state) and a desirable new condition (goal state). For example, if a pregnant person has to be operated on for a planned child birth, specialists of the concerned medical fields need to coordinate specific procedures; schedule it with relevant nursing staff; complete insurance formalities; and reserve resources like the operation room. Some of these processes follow standardized or regulated workflows, while others are case-specific depending on patient risk factors, etc. Furthermore, the data in such scenarios must be controlled due to confidentiality and regulatory reasons (Leyens *et al.* 2017). An automated decision maker can help by focusing the attention of the medical team on ensuring compliance, examining risk factors and medical requirements, and avoiding costly mistakes that may foreclose future remedial actions.

Personal Finance Increasingly, personal finance has emerged as an area of great opportunity as well as challenge for decision making systems and decision assistants. Use-cases like buying a house, saving for retirement, or filing one's taxes are important decisions with long-term life implications. A number of characteristics must be considered including the various alternatives available, their costs (both immediate and future), legal and compliance issues, etc. A specific example of such a decision making scenario is an automated tax assistant – such an assistant must be aware of the tax code which prescribes various rules and regulations that must be followed, and must recommend the best tax plan while optimizing a number of metrics including minimizing amount paid as tax, minimizing the complexity of the plan, and maximizing compliance (to minimize the chances of audits and fines).

4 Metrics

We now list some metrics from prior work that can be adapted to the problem we consider. Chakraborti et al. (2016a) provide a framework for studying and evaluating interaction between human and robot team-members in goal-oriented environments. Some useful metrics that can be adapted from that work are:

1. **Neglect Tolerance (NT)**: How long the agent is able to perform well without human intervention.
2. **Interaction Time (IT)**: Time spent in communication.
3. **(Robot) Attention Demand (AD)**: Measures the attention demanded by the agent.
4. **Fan Out (FO)**: Communication load on the humans; proportional to the number of agents.
5. **Compliance (Com)**: How well the actions of an agent convey its intention to comply.

Separately, Keller et al. (2007) consider the problem of workflow complexity relating to configuring Information Technology (IT) infrastructure, e.g. a web application. They define configuration complexity as "the complexity of carrying out a configuration procedure as perceived by a human system manager"; and track information along three dimensions, which are (respectively) analogous to control flow, data flow, and space complexity in software engineering:

6. **Execution Complexity (EC)**: Number of actions and context switches.
7. **Parameter Complexity (PC)**: Number of parameters used by actions, and their usage variations.
8. **Memory Complexity (MC)**: Number of configuration values which need to be remembered along the workflow, and over the actions.

These measures are all relevant from a human-agent collaboration perspective, as they relate to the effort needed to review a plan and to gain human trust.

5 Discussion

In Table 1, we present the above metrics juxtaposed with their footprint in the collaborative domains introduced in Section 3. The footprint itself is quantized into three categories – **High (H)**, **Medium (M)**, and **Low (L)**. We address

a number of points in relation to the table. First and foremost, the table should be read column-wise, for each metric. Second, the High/Medium/Low annotations denote the typical or average-case profile for that metric in the respective usecase, and may vary depending on the specific problem instance etc.

Third, these values do not represent any intrinsic *goodness* – high neglect tolerance is good in scenarios like Human-Robot Teaming, because it shows that the automated agent is more independent; while low compliance might be a bad thing if the human wants constant confirmation or reassurance from the agent, like in medical treatment scenarios. However, these can easily switch depending on the domains and users in question: medical professionals may want a less independent agent (lower neglect tolerance), while meeting scheduling agents may not be required to show all the steps of their work. A general rule-of-thumb is that if the metric profile of a particular usecase is reflected in the plans that a planner produces, overall team success is more likely.

We now discuss the metrics from Table 1 in the context of creating new metrics that define the complexity of plans or workflows in terms of the interaction issues, as well as the complexity of the actions that constitute those workflows. The first set of metrics informally represent interaction issues: Neglect Tolerance (NT), Interaction Time (IT), and Attention Demand (AD) are related to each other, and are concerned with the demands that a workflow imposes on the user/human via the agent's roles in the workflow. Similarly, IT and Fan Out (FO) offer a measure of the communication that is expected from the user, and how many different agents the user must accommodate (the assumption being that communication load increases as a function of the number of such agents). The second set of metrics represents the complexity of the actions in the workflow itself: while a scenario that involves scheduling meetings might feature a number of possible alternative workflows and each action might consist of multiple parameters, other scenarios like human-robot teaming might in fact feature relatively fewer alternatives and action parameters. These are all important to track in the final plan that is generated for the human-agent team, since they contribute to the difficulty of explaining the workflow and its constituent parts (as required).

6 Conclusion & Future Work

We conclude by reiterating that the metrics we discuss in this paper differ from the traditional metrics used in the planning community, which apply specifically to actions and goal-states; the optimal profiles for these metrics are instead at least partially determined by the usecase in question. We would like to use these as a starting point in ultimately creating metrics that explain the complexity of the workflow cumulatively from the perspective of the agent that must understand, explain, or execute it. Our hope is that this paper will spur action in two directions: (1) the post-processing of plans from existing planners to take cumulative plan complexity metrics into account; and eventually, (2) the creation of new planners that can handle such complexity metrics directly in the state-space search and plan synthesis processes.

References

Tathagata Chakraborti, Kartik Talamadupula, Yu Zhang, and Subbarao Kambhampati. A formal framework for studying interaction in human-robot societies. In *AAAI 2016 Workshop on Symbiotic Cognitive Systems (SCS)*, 2016.

Tathagata Chakraborti, Yu Zhang, David E Smith, and Subbarao Kambhampati. Planning with resource conflicts in human-robot cohabitation. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1069–1077. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

Justin Cranshaw, Emad Elwany, Todd Newman, Rafal Kocielnik, Bowen Yu, Sandeep Soni, Jaime Teevan, and Andrés Monroy-Hernández. Calendar. help: Designing a workflow-based scheduling agent with humans in the loop. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2382–2393. ACM, 2017.

Robert P Goldman and Ugur Kuter. Measuring Plan Diversity: Pathologies in Existing Approaches and A New Plan Distance Metric. In *AAAI*, pages 3275–3282, 2015.

A. Keller, A. B. Brown, and J. L. Hellerstein. A Configuration Complexity Model and Its Application to a Change Management System. *IEEE Transactions on Network and Service Management*, 4(1):13–27, June 2007.

Lada Leyens, Matthias Reumann, Nuria Malats, and Angela Brand. Use of big data for drug development and for public and personal health and care. *Genetic Epidemiology*, 41(1):51–60, 2017.

Q. Vera Liao, Biplav Srivastava, and Pavan Kapanipathi. Tailoring Conversational UX through the Lens of Dialogue Complexity. In *CHI Workshop on Conversational UX Design*, 2017.

Tuan Anh Nguyen, Minh Do, Alfonso Emilio Gerevini, Ivan Serina, Biplav Srivastava, and Subbarao Kambhampati. Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence*, 190:1–31, 2012.

Mark Roberts, Adele E Howe, and Indrajit Ray. Evaluating Diversity in Classical Planning. In *ICAPS*, 2014.

Biplav Srivastava and Jana Koehler. Web Service Composition - Current Solutions and Open Problems. In *In: ICAPS 2003 Workshop on Planning for Web Services*, pages 28–35, 2003.

Kartik Talamadupula, J Benton, Subbarao Kambhampati, Paul Schermerhorn, and Matthias Scheutz. Planning for human-robot teaming in open worlds. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 1(2):14, 2010.

Kartik Talamadupula, Gordon Briggs, Tathagata Chakraborti, Matthias Scheutz, and Subbarao Kambhampati. Coordination in human-robot teams using mental modeling and plan recognition. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2957–2962. IEEE, 2014.

Wil M.P. van der Aalst and Kees van Hee. Workflow Management: Models, Methods, and Systems. In *ISBN:978-0262720465, MIT Press*, 2004.