Goal-Oriented End-User Programming of Robots

David Porfirio
NRC Postdoctoral Research Associate
U.S. Naval Research Laboratory
Washington, DC, United States
david.porfirio.ctr@nrl.navy.mil

Mark Roberts
U.S. Naval Research Laboratory
Washington, DC, United States
mark.roberts@nrl.navy.mil

Laura M. Hiatt
U.S. Naval Research Laboratory
Washington, DC, United States
laura.hiatt@nrl.navy.mil

ABSTRACT

End-user programming (EUP) tools must balance user control with the robot's ability to plan and act autonomously. Many existing task-oriented EUP tools enforce a specific level of control, *e.g.,* by requiring that users hand-craft detailed sequences of actions, rather than offering users the flexibility to choose the level of task detail they wish to express. We thereby created a novel EUP system, *Polaris*, that in contrast to most existing EUP tools, uses *goal predicates* as the fundamental building block of programs. Users can thereby express high-level robot objectives or lower-level checkpoints at their choosing, while an off-the-shelf task planner fills in any remaining program detail. To ensure that goal-specified programs adhere to user expectations of robot behavior, *Polaris* is equipped with a *Plan Visualizer* that exposes the planner's output to the user before runtime. In what follows, we describe our design of *Polaris* and its evaluation with 32 human participants. Our results support the *Plan Visualizer*'s ability to help users craft higher-quality programs. Furthermore, there are strong associations between user perception of the robot and *Plan Visualizer* usage, and evidence that robot familiarity has a key role in shaping user experience.

CCS CONCEPTS

• **Human-centered computing** → **Systems and tools for inter-action design**; • **Computer systems organization** → **Robotics**.

KEYWORDS

human-robot interaction, end-user programming, task planning

1 INTRODUCTION

As robots permeate our daily lives, there is a growing demand for efficient and reliable approaches that allow end users to specify tasks for these robots to perform. *End-user programming* (EUP) tools, *i.e.,* software environments that enable these users to create and customize robot applications, represent a viable class of
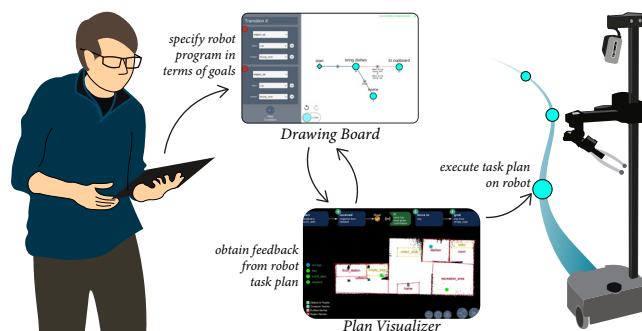
Figure 1: With *Polaris,* end-user programmers specify *goal automata* and view the resulting plan in the *Plan Visualizer*.

solutions. Given the autonomous capabilities of everyday robots, users should be free to omit certain details from their programs. To illustrate, consider the following scenario: *a caregiver needs a robot to deliver lunch to a resident in a care facility.* Rather than requiring the caregiver to specify a long string of actions (*e.g., go to cafeteria, pick up tray, go to food station, wait for food, go to resident,* and *give food*), the caregiver can leverage the robot's ability to plan and act autonomously if simply given a desired goal state: *lunch delivered.*

At the same time, these users must have the flexibility to express additional detail as needed, based on their own domain expertise. We define *flexibility* as being able to choose between *low* oversight (expressing minimal goals and letting the robot resolve the details) and *high* oversight (specifying more details to constrain the robot). Caregivers, in particular, can benefit from being able to access different levels of oversight [53]. Perhaps the caregiver in our example desires higher oversight due to their domain knowledge—the resident is usually in the recreation area midday, but they must be in their room to eat lunch due to care facility rules. In this case, there is an additional implied outcome that the resident should be in their room before the food is delivered. Constraining the robot with two goals in sequence will suffice: (1) *resident alerted* to ensure that the resident knows to travel to their room, and then (2) *lunch delivered.*

Unfortunately, there has been limited exploration of EUP tools that leverage robot autonomy while still affording users flexibility in program specification. Most existing EUP tools necessitate high oversight by requiring users to hard-code robot actions, which as evidenced by existing datasets of user-generated action sequences, exhibits high contextual conformity [36]. In this work, we challenge the action-oriented EUP paradigm for human-robot interaction (HRI) by proposing *goal predicates* as an alternative fundamental building block of robot programs. By selecting and parameterizing goal predicates, users can omit details on how an *intended effect* (*i.e.,* "goal state," or "goal" for brevity) is achieved. Furthermore, in

	Front-End Goal Preds	AI Planning	Domain	Interface Feedback	User Study
Polaris	✓	✓	Service	✓	✓
"Spbd" [9]	✓		Nav.	✓	
JESSIE [33]			Social		✓
Tabula [42]		✓	Service	✓	
RoVer [41]			Social	✓	✓
RoboFlow [3]			Service		✓

Table 1: A comparison of *Polaris* to closely related EUP tools. Polaris exposes goal predicates to users, explicitly incorporates AI planning techniques, provides visual feedback on user programs through its interface, and is user-evaluated.

recognizing that goals do not contain explicit information about which actions the robot will perform, we ask how goal-oriented EUP tools can ensure that user expectations match robot performance.

To address these gaps, we created a goal-oriented EUP system, *Polaris*, that represents our vision of flexibility, abstracting away unnecessary detail while still affording users appropriate control over the robot. Figure 1 depicts the high-level usage flow of *Polaris*, which exists as a handheld tablet interface. With *Polaris*, end users specify *goal automata*—a flow-based representation in which nodes in the flow represent goals rather than actions. This enables end-user programmers to specify programs at a level of detail with which they are comfortable or is required by their domain expertise. *Polaris* then automatically generates a branching task plan through off-the-shelf AI planning approaches. To ensure that plans match developer intent and to provide feedback for refinement, *Polaris* includes a *Plan Visualizer* interface that exposes the plan to users.

The *Polaris* system represents an ongoing research effort. This paper describes a snapshot of this effort, culminating in *Polaris* V1.0, and highlights our motivations and initial design decisions. Our evaluation tests these design decisions, finds evidence that the *Plan Visualizer* improves plan quality, and uncovers associations between user experience and both *Plan Visualizer* usage and self-reported robot familiarity. We conclude by offering design implications and discussing how these implications inform our own future work and future development of EUP systems in general.

Our contributions include: *Systems* — the *Polaris* system, a novel goal-oriented EUP tool and our primary contribution. *Empirical* — an evaluation of *Polaris*, namely the *Plan Visualizer*'s ability to assist end-user programmers with creating goal-oriented programs. *Design* — design implications that emerged from our evaluation.

2 RELATED WORK

Polaris' contributions and novelty are situated within end-user programming and draw heavily from goal-oriented task specification and automated planning in HRI.

2.1 Robot End-User Programming

End-user programming pertains to the creation of software applications by the application users themselves [7]. Contributions in robot EUP often focus on novel ways to capture user intent through visual programming environments [*e.g.,* 34, 50], augmented reality [*e.g.,* 12, 13], natural language [*e.g.,* 20, 23], or multimodal input [*e.g.,*

8, 43], to name a few examples. In HRI, end-user programmers[1] are typically (though not always) programming novices, and may also be domain experts specialized in specific fields [2]. *Polaris*' target end-user programmer includes domain experts in need of personalized (*e.g.,* through goal-oriented specification) yet reliable (*e.g.,* through the *Plan Visualizer*) robot execution, such as caregivers, military personnel, and disaster response teams.

Polaris' contribution lies primarily in its *goal-oriented* programming paradigm—users specify an intended effect in terms of goal state rather than an *action-oriented* description of robot behaviors to achieve that effect. Overwhelmingly, existing EUP systems for HRI are *action-oriented*. Action-oriented examples from the EUP literature include *block-based* [*e.g.,* 16, 28, 29, 50], *flow-based* [*e.g.,* 3, 44], and *event-based* [*e.g.,* 34] tools, in which the fundamental building block of a robot application is an action or command.

Table 1 characterizes *Polaris*' novelty against a representative selection of similar, existing EUP systems and programming approaches. Notably, Brageul et al. [9]'s *simple programming by demonstration* ("*spbd*") interface is similar to *Polaris*' goal-oriented nature in that it allows users to directly manipulate goal predicates. Unlike *Polaris*, however, *spbd* is limited to navigation domains and lacks a user study. Another tool, *JESSIE*, similarly captures goal state within its program logic, but this logic is not exposed to the user [33]. *Polaris* additionally distinguishes itself from prior work that views goals as high-level task commands (*e.g.,* "open a sliding door" as a goal in [1]) due to our strict definition of goals as expressing desired state rather than any information about the robot's actions.

Goal-oriented nature aside, *Polaris* draws heavily from other prior work. *Tabula*, in particular, exists within a handheld tablet, invokes a planner to determine robot behavior, and offers plan feedback through its user interface [42]. *Tabula*, however, only exposes actions to users and is not yet evaluated in a user study. Although not incorporating a planner, both *RoVer* [41] and *RoboFlow* [3] afford users a similar flow-based specification interface to *Polaris* and check pre and postconditions between consecutive robot actions. *RoVer* is additionally similar to *Polaris* in its user interface, namely through the inclusion of a dedicated feedback pane.

2.2 Goal-Oriented Specification Paradigm

Goals are a critical component of many formal representations, architectures, and models for autonomous agents. The *belief-desire-intention* (BDI) paradigm presents one such modeling approach for agent reasoning [10] and has led to numerous agent-oriented programming languages, including *AgentSpeak* [46] and variants of *CAN* [47], of which goals are of great importance. Goals are additionally critical to the specification of both classical and hierarchical planning problems [22, 51]. For expressing robot programs purely via goals, *Polaris* utilizes an approach most similar to *Agent Planning Programs* [18], in which programs are represented as transition systems with transitions between program states labeled by goals and guard functions.

Prior work demonstrates how goal-oriented languages improve user outcomes. In particular, Hu et al. [27] shows how decomposing a task into higher-level objectives ("goals") combined with block-based programming can improve learning outcomes among

---

[1]We often refer to end-user programmers as simply "end users" or "users" for brevity.

students. Additionally, Cox and Zhang [17] contrast two mixed-initiative planning approaches, *goal manipulation* versus *search* (*i.e.,* searching through action pre and postconditions), and find that goal manipulation surpasses search in terms of user performance and efficiency. These results support our design choice to expose goals to users through *Polaris*.

Although less common in robot EUP, goal-oriented specification is popular in related fields. Prior work within the Internet of Things (IoT), in particular, has produced numerous EUP tools, languages, and architectures for specifying smart home configurations in terms of predicates (*e.g., temperature is x*, in which *x* is a value in degrees) [32, 38]. An underlying motivation of goal-oriented specification in IoT is to reduce development time [39].

2.3 Automated Planning in HRI

Planning and acting involves selecting *what* an agent does and *how* it does it [22]. The scope of our work is on the former—*what.* Within this scope, *automated* planning solves for plans with respect to a *planning domain*, that is, constraints which are often specified in languages such as the *Planning Domain Definition Language* (PDDL) [21] or the *Action Notation Modeling Language* (ANML) [52]. Notable planning work in HRI involves reasoning about and responding to human behavior [4, 30, 40] and creating robot planning domains through demonstration [35]. Recently, Chakraborti et al. [15] investigated the use of plan explanations to improve the shared understanding of a robot's decisions.

Various interfaces exist for visualizing and enabling end users to interact with plans, *PDSim* being a notable example in robotics [19]. Of the existing planning interfaces in HRI, *Tabula* is most similar to *Polaris* but is action-oriented and focuses more on the mechanism for capturing the intent of end-user programmers [42]. A plethora of other such interfaces (*e.g.,* RADAR-X [55]) exist outside of HRI.

3 SYSTEM DESIGN

Our description of *Polaris* begins by elaborating on the caregiving scenario presented in *Introduction* (§1) to illustrate the user's perspective, followed by our technical approach for (1) specifying task objectives in terms of goals, (2) generating a task plan, (3) viewing the plan, (4) running the plan, and (5) *Polaris*' implementation.

3.1 User Perspective

Figure 2 depicts the user's perspective of *Polaris* with its various components described below.
World. The user begins by requesting a two-dimensional map of the environment from the robot and uploading semantic labels for key entities that the robot can recognize. The semantically labeled map (accessible though the *Plan Visualizer*, Figure 2b) depicts the *world* that the robot operates within, and includes the two-dimensional representation of the robot's environment and the entities therein. There are five general categories of entities—objects, containers, surfaces, regions, and people. Objects include anything that the robot can grab. Containers include anything within which an object can be placed. Surfaces are areas upon which objects can be placed and are non-traversable by the robot. Regions are traversable areas in the environment. People include the robot's potential interaction partners. Within the world, the locations of objects and people

represent initial positions, and these entities can be moved around throughout the course of a program's execution.

Figure 2b depicts the world within our caregiving scenario. The care facility has been labeled with regions such as the *recreation area*, the resident's *room*, and the *cafeteria*. The robot can place items onto and remove items from surfaces such as the *empty trays* surface and the resident's *table*. The *tray* is a manipulable object and the *resident* is an interactable person in the environment.
Creating a Goal Automaton. The user enters the *Drawing Board* to specify a program in terms of goals. Figure 2a depicts the *Drawing Board* with an example user-based solution, called a *goal automaton*. In the goal automaton, blue nodes represent *checkpoints*. Checkpoints contain goals, thereby indicating the state of the world that the user wants the robot to achieve at that point in the program. Connecting checkpoints with lines (called *transitions*) enforces an ordering and allows users to specify a *conditional*, or world state that must be true (possibly outside of the robot's control) for the robot to proceed from one checkpoint to another. Checkpoints contain no indication of the robot's geographical location and are purely intended to represent program flow.

Figure $2a_{1-4}$ shows the process of building the goal automaton to represent the caregiver's objectives. The resident must be informed of lunchtime before lunch is served, so the caregiver's first step (Figure $2a_1$) is to draw a new checkpoint and label it *alerted*. When a new checkpoint is created, a parameterization menu appears that prompts users to add goals to the checkpoint. Users may also click on existing checkpoints during the course of goal automata creation to modify these checkpoints' goals. Within *alerted*, the caregiver assigns *x* and *y* values to the "*x–alertedTo–y*" predicate to create a *ground predicate* and assert a goal: The resident has been alerted to lunch being served, namely (*resident–alertedTo–lunchtime*).[2]

Next (Figure $2a_2$), the caregiver draws a line from *alerted* to a new checkpoint, *fetched*, and inserts goals (*tray–at–table*) and (*tray–is full*), indicating that after alerting the resident, lunch must be delivered. The caregiver labels the transition from *alerted* to *fetched* with the ground predicate (*acknowledged–lunchalert*), in this case indicating a conditional that the robot can only proceed from the *alerted* checkpoint to the *fetched* checkpoint if the resident acknowledges the alert. Throughout the course of goal automata creation, the user may click on existing transitions to modify the transitions' conditionals. In order to handle the edge case in which the robot's lunchtime alert is dismissed (*e.g.,* if another caregiver has already served the resident lunch and wishes to cancel the robot's task), the next step taken by the caregiver (Figure $2a_3$) is to draw a transition from *alerted* to a new checkpoint, *cancelled*, assign a goal of (*robotAt–home*), and label the new transition with the conditional *dismissed*. Finally (Figure $2a_4$), the caregiver draws a new checkpoint from *fetched* called *home* and adds another (*robotAt–home*) goal. More information on the semantics of goal automata can be found in *Representing Goal Automata* (§3.2).
Viewing and Running a Plan. As the user makes progress on their goal automaton, *Polaris* computes a task plan behind the scenes, which contains the exact actions that the robot plans to take to achieve each of the caregiver's goals in sequence. *Polaris* presents this information to users via the *Plan Visualizer* interface (Figure 2b).

---

[2]Our goal notation parenthesizes **bolded** predicate symbols and *italicized* terms.
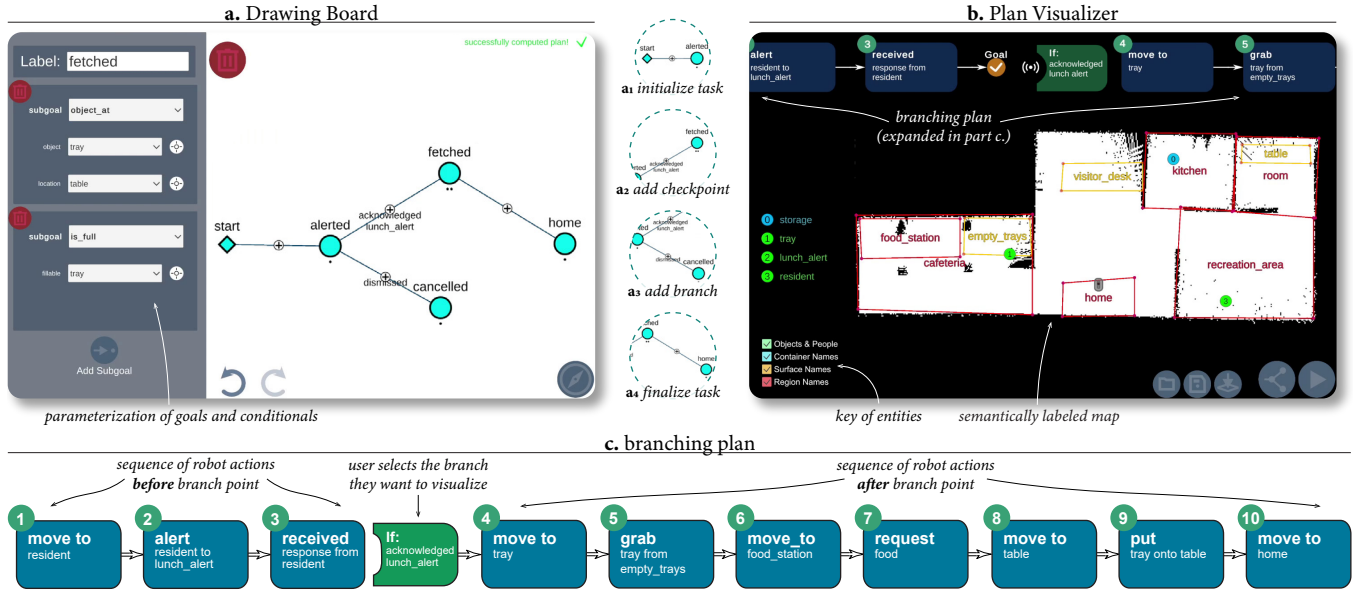
Figure 2: The *Polaris* user interface, which includes (a) the *Drawing Board* for specifying goal automata and (b) the *Plan Visualizer* for displaying the robot's plan. (c) A single branch of the branching plan from the caregiving scenario is displayed.

Figure 2c depicts the plan based on the caregiver's goal automaton as it is presented to the user. At any point during goal automaton creation, the user can flip back and forth between the *Drawing Board* and *Plan Visualizer* to iterate on receiving plan feedback and performing modifications to their goal automaton. Once the user presses the play button (Figure 2b, bottom right), the robot begins to execute the plan. Branching plan creation, viewing, and execution are detailed further in *Creating a Branching Plan* (§3.3), *Viewing the Branching Plan* (§3.4), and *Plan Execution* (§3.5).

3.2 Representing Goal Automata

Formally, a goal automaton is a transition system [6] that guides the robot in achieving goals during its task and is represented by the tuple $(P, C, L_c, \longrightarrow, c_0)$:

Predicates. P is a set of *ground predicates*, *i.e.,* predicates with assigned variable values. Predicates primarily represent goals, but can also represent conditionals, namely world state that must be true for the robot to proceed. Intuitively, conditionals indicate that the robot must wait for a particular outcome that may be out of the robot's control.

Checkpoints. C is a set of *checkpoints*. Intuitively, checkpoints represent points in the program in which the robot has achieved a desired set of goals $p \in 2^P$, in which 2^P is the power set of P.

Goals. $L_c : C \to 2^P$ maps checkpoints to goals.

Transitions. $\longrightarrow \subseteq C \times 2^P \times C$ is the transition relation between checkpoints subject to a conditional being true. For example, $p \in 2^P$ is a conditional within the transition $c_i \xrightarrow{p} c_j$. Intuitively, a transition labeled with conditional p means, "wait for p to be true before transitioning between c_i and c_j." A transition with no conditional annotation (*e.g.,* $c_i \to c_j$) means "transition from c_i to c_j if no other transitions from c_i are able to be taken."

Initial Checkpoint. c_0 is the always-empty "start" checkpoint. c_0 represents initial state and does not have goals: $L_c(c_0) = \emptyset$.

Figure 2a depicts the interface for specifying goal automata, the *Drawing Board*. Initially, the *Drawing Board* contains the empty checkpoint c_0. Each new checkpoint c_j must be drawn as $c_i \to c_j$ such that $c_i \in C$ (*i.e.,* new checkpoints must connect to existing checkpoints), $L_c(c_j) = \emptyset$ (*i.e.,* new checkpoints *initially* contain no goals), and new transitions *initially* contain no conditionals. While the intention is to support loops in future versions of *Polaris*, goal automata are presently drawn as trees.

3.3 Creating a Branching Plan

Branching plans are *compiled* in real-time as changes are made to the goal automaton, provided that there are no underspecified transitions (*i.e.,* two transitions with the same conditional extending from the same checkpoint). If a checkpoint in the goal automaton contains conflicting goals (*e.g.,* the user asserts that a single-arm robot must hold two items at the same time), *Polaris* will omit that checkpoint and further checkpoints in its subtree from compilation.

Formally, a branching plan is similar in tree structure to the goal automaton but consists of *actions* rather than goals. Let A be the set of tree nodes in the plan and $L_a : A \to 2^P$ be the world state after a node's action has been executed. Let $\longrightarrow \subseteq A \times 2^P \times A$ be the transition relation between nodes. Given a goal automaton, *Polaris* creates a branching plan such that there is an injective non-surjective mapping between checkpoints and plan nodes $f_m : C \to A$. For convenience, let $f_s : A \to 2^A$ map node a_i to the set of nodes in a_i's subtree. Subject to the following additional constraints, *Polaris* leverages an off-the-shelf planner for plan creation.

Constraint on Transitions—For each transition $c_i \xrightarrow{p} c_j$ in the tree-like goal automaton, there must be a corresponding transition in the
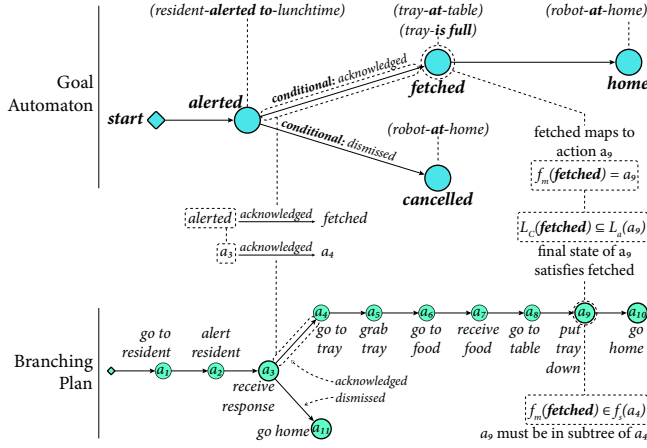
Figure 3: Computing branching plans from goal automata.

branching plan $a_i \xrightarrow{p} a_j$ such that $f_m(c_i) = a_i$ and $f_m(c_j) \in f_s(a_j)$, that is, c_i maps to a_i and the mapping of c_j is in the subtree of a_j. Figure 3 illustrates this constraint—the transition between *alerted* and *fetched* maps to the transition between a_3 and a_4, and *fetched* maps to a descendent of a_4.

Constraint on Goal Achievement—A checkpoint's goals must match the state of the world after the completion of its corresponding action in the branching plan: $L_c(c) \subseteq L_a(f_m(c))$. Figure 3 illustrates this constraint: The goals of *fetched* match the end effect of its corresponding action in the plan, a_9.

3.4 Viewing the Branching Plan

At any point during the creation of a goal automaton, end users may view the branching plan computed by *Polaris* within the *Plan Visualizer* interface. The *Plan Visualizer* draws from existing "timeline" interfaces in HRI [48, 49] in that it displays one branch of the plan at a time from left to right, and within a horizontal scrollable pane overlaying the semantically labeled map. Initially, the plan is displayed up to when a conditional is encountered. Users then select the conditional corresponding to the branch they wish to visualize via a dropdown menu. Following the user's selection, the *Plan Visualizer* displays the corresponding branch up to the next conditional. Actions within each branch can be clicked to depict the world state that results from the execution of the clicked action.

3.5 Plan Execution

When the user is satisfied with their goal automaton and resulting plan, they may execute the plan on the robot. During plan execution, *Polaris* enters a feedback-execution loop with the robot. Rather than sending the robot actions directly from the plan, *Polaris* converts actions to goals (*i.e.,* by using the end effects of an action). This enables the robot to compute a new plan to achieve the end effect of each action, rendering the robot flexible to minor perturbations in the environment. The robot is thereby able to re-plan when its perceived state of the world changes and repeat this process until the goal-converted action has been achieved. The robot then sends

a confirmation back to *Polaris*, which converts the next action in the plan to a goal and sends the new goal to the robot.

3.6 Implementation

Polaris exists within a front-end tablet and a back-end planner communicating over a RESTful API. The front end is implemented in Unity version 2022.2.1f1 [54] and is compiled to Android. While *Polaris* is primarily intended for handheld use, its implementation in Unity has enabled us to deploy it on a web browser and as a desktop application.

The *Polaris* back end is implemented within Python 3.8 and accesses a planning domain expressed in PDDL [21].[3] At a high level, the planning domain consists of (1) a set of predicates that operate over both the robot's state and entities in the world, and (2) a set of operations that the robot can perform, including how these operations affect both the robot and the world. We integrated an off-the-shelf planner, *Fast Downward* [25, 26], within the back end.

We use the Hello Robot Stretch RE2 robot [31] as our runtime platform. Communication between *Polaris* and the robot occurs through the Noetic version of the Robot Operating System [45].

4 SYSTEM EVALUATION

To evaluate our systems-level contribution and understand the interaction between the core components of the system, we conducted an IRB-approved laboratory study that compares the full version of *Polaris* with an ablated baseline without the *Plan Visualizer*. Our hypotheses are that exposure to the *Plan Visualizer* improves the quality of the resulting plans (H1), helps match user expectations to robot task performance (H2), improves the perceived competence of the robot (H3), and improves *Polaris'* usability (H4).

4.1 Study Design

We conducted an experiment with two conditions—*plan-vis*, in which participants were exposed and allowed access to *Polaris'* *Plan Visualizer*, and *no-vis*, in which participants were neither exposed nor allowed access to the *Plan Visualizer*. Participants specified a goal automaton and executed the resulting plan on a robot.

4.1.1 Study Scenario. Our evaluation centered on a *tidying* scenario. Participants were informed the following:

> You are finished hosting a dinner for some friends, and now it is time to clean up. While you wash dishes in the kitchen, you want your robot to help deliver dirty dishes to you and deliver clean dishes to the cupboard.

Figure 4 (right) depicts the physical layout of the study. Participants were informed that their job was to wash a dirty plate and cup. Participants were also informed of the robot's capabilities—it can deliver dishes to and from the participant's vicinity and open the cupboard. Participants were not allowed to step out from behind the countertop (see *participant reachable area* in Figure 4, right).

Participants were informed that to clean a dish, they needed to access the dish (presumably by having the robot deliver the dish to their vicinity) and place the dish on the drying rack. Once on the drying rack, the dish is clean and ready to be put away. The

---

[3]Planning domains are interchangeable within *Polaris*. Example planning domains can be found at https://osf.io/ewfd5/.
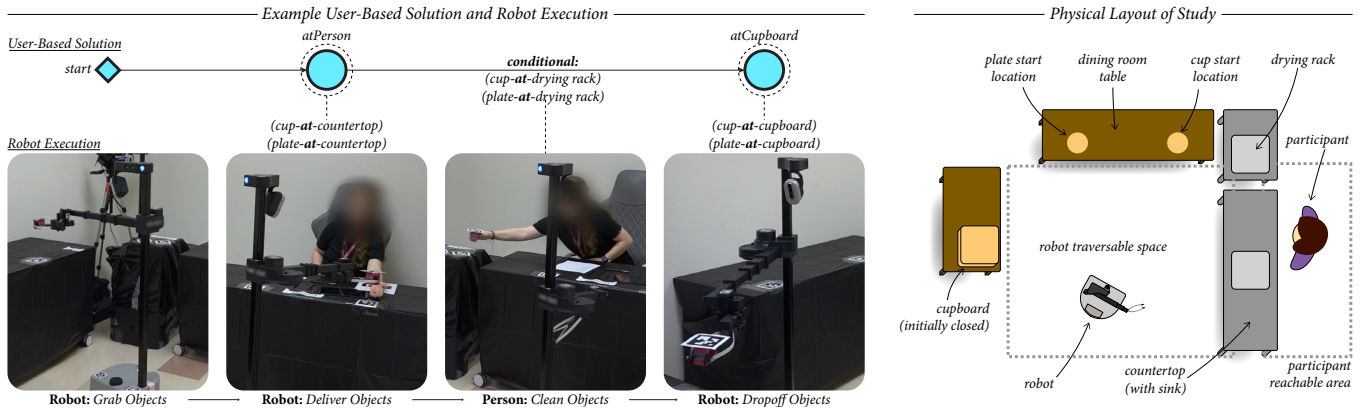
Figure 4: The smallest solution to the *tidying* scenario and its execution (left). The physical layout of the study room (right).

smallest solution for the *tidying* scenario is depicted in Figure 4 (left). On execution of this solution, the robot delivers the dishes to the countertop, waits until the dishes are on the drying rack, then opens the cupboard, and finally moves the dishes to the cupboard.

4.1.2 Measures. To measure plan quality, we first enumerate four basic objectives of the *tidying* scenario—(1) cup clean, (2) plate clean, (3) clean cup in the cupboard, and (4) clean plate in the cupboard. Giving each objective equal weight, we then compute (1) a *runtime score*, or how many objectives the robot meets during plan execution; and (2) a *feasibility score*, or the maximum number of objectives that the robot *could* meet at runtime. The *runtime* score may be lower than the *feasibility* score if the participant acts suboptimally at runtime, *e.g.,* if the participant removes the cup and plate from the drying rack before the robot has had the chance to grab them. Higher values are better for *runtime* and *feasibility* scores. We additionally created (3) a third and more fine-grained analysis of task quality—*human effort*. Given a participant's task plan, the measure asks: What is the minimum number of independent actions that a human would have to perform during the robot's execution for all four objectives in the task to be achieved? To compute this measure, we relax the assumption that participants stay behind the countertop. Lower values of *human effort* are better.

We measure usability via the SUS questionnaire (10 items, 5-point Likert scale) [11] and the usefulness (8 items), ease of learning (4 items), and satisfaction (7 items) factors of the USE questionnaire (7-point Likert scale) [37]. To measure perceived robot competence, we include the competence factor of the RoSAS scale (6 items, 7-point Likert scale) [14]. We developed our own *expectations* questionnaire to measure the degree to which expectations of robot performance are matched in terms of four factors (5 items each, 7-point Likert scale)—expectations *overall* (Cronbach's α = 0.80), expectations of *what* the robot did (Cronbach's α =0.78), and expectations of *where* (Cronbach's α = 0.91) and *why* (Cronbach's α = 0.87) it did it. Although not part of our hypotheses, we measured task load through the NASA TLX (7 items, 7-point Likert scale) [24].[4]

4.1.3 Procedure. Study sessions lasted for one hour. After giving their consent to participate, participants completed a self-guided

browser-based *Polaris* tutorial. Participants were encouraged to ask questions at this stage, to which the experimenter responded within the scope of the tutorial. *Plan-vis* participants were exposed to the *Plan Visualizer* through an additional tutorial step.

After the tutorial, participants were briefed on the *tidying* scenario and given 10 minutes to specify a goal automaton within *Polaris*. Within the 10 minutes, *plan-vis* participants had unlimited access to the *Plan Visualizer* at their discretion. After 10 minutes or when participants indicated that they had finished, we administered the TLX, USE, and SUS questionnaires.

Participants were then given instructions for executing their plans on the robot. Figure 4 (left) depicts a sample execution. During execution, participants observed the status of the robot on the tablet. If the robot encountered a conditional, it waited for confirmation from the participant before proceeding. Participants were informed that they could move any items around in their vicinity, *e.g.,* to or from the countertop and the drying rack.

During execution, deviations from the robot's expected world state were engineered to cause the robot to prematurely halt execution. Deviations could occur, for example, if the participant failed to specify in their goal automaton that clean dishes would be placed on the drying rack (*i.e.,* by failing to insert a conditional that instructs the robot to wait until the dishes are on the drying rack before proceeding to put them in the cupboard). In this case, the robot would believe the dish to be on the countertop, but without seeing the dish on the countertop, it would be unable to proceed. Participants would be given a few seconds to realize their mistake and put the clean dish back on the countertop, but if deviations remained uncorrected, the robot halted execution.

At the end of execution, participants filled out the questionnaires for expectations matched and perceived competence. To adhere to the one-hour time limit, one participant was administered these questionnaires prior to when the robot had finished execution. If time permitted, participants underwent semi-structured interviews. Interview durations varied based on the remaining time in the hour.

4.2 Results

Participants. We recruited 33 volunteers (19 male, 14 female) from within the U.S. Naval Research Laboratory in Washington,

---

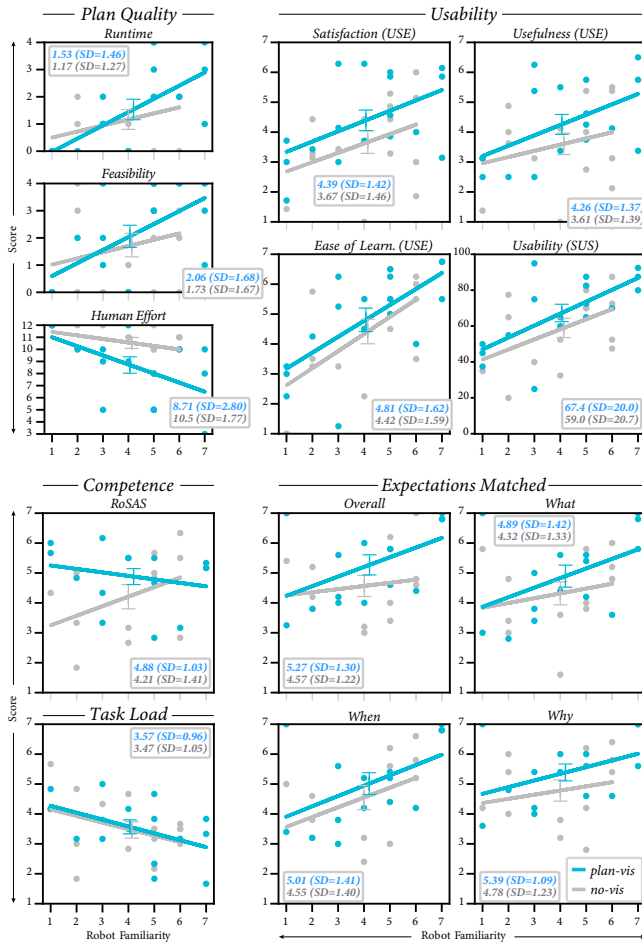[4]Copies of the study materials can be found at https://osf.io/ewfd5/.

Figure 5: Plan quality (top left), usability (top right), competence and task load (bottom left), and expectations matched (bottom right) plotted against robot familiarity. Linear trendlines indicate the direction of the relationship. Blue is *plan-vis*. Grey is *no-vis*. Lower is better only for *human effort* and task load. Error bars represent standard error.

	Correlation with Robot Familiarity		Correlation with *Plan Visualizer* Usage	
Measure	**Spearman's *r***	***p***	**Spearman's *r***	***p***
Runtime score	**0.556**	**<0.01**	0.468	0.079
Feasibility score	**0.445**	**0.011**	0.387	0.125
Hum. effort cost	**-0.460**	**<0.01**	-0.377	0.136
Expect. *overall*	**0.398**	**0.040**	**0.831**	**<0.001**
Expect. *what*	**0.422**	**0.028**	**0.737**	**<0.01**
Expect. *when*	**0.484**	**0.011**	**0.784**	**<0.001**
Expect. *why*	0.371	0.057	**0.614**	**0.015**
Competence	0.066	0.745	0.051	0.857
Usability (SUS)	**0.568**	**<0.001**	**0.496**	**0.043**
Usefulness	**0.398**	**0.024**	0.433	0.083
Ease of Learn.	**0.639**	**<0.001**	0.453	0.068
Satisfaction	**0.390**	**0.027**	0.226	0.383
Task Load	**-0.399**	**0.024**	-0.092	0.724

Table 2: Spearman's rank coefficient with robot familiarity (all data) and *Plan Visualizer* usage (*plan-vis* only) for each measure. Bold indicates statistical significance ($p < 0.05$).

compare usability, competence, and expectations matched between conditions. We observed a significant difference in plan quality in terms of *human effort* between the *plan-vis* and *no-vis* conditions ($U = 182.5$, $p = 0.017$). We observed marginal effects ($p < 0.1$) for *plan-vis* performing better than *no-vis* participants for our measures of perceived usefulness of *Polaris* ($p = 0.097$, $t(30) = 1.33$), satisfaction with *Polaris* ($p = 0.081$, $t(30) = 1.43$), perceived competence of the robot ($p = 0.083$, $t(25) = 1.43$), and expectations matched both *overall* ($p = 0.082$, $t(25) = 1.43$) and for *why* the robot acted ($p = 0.095$, $t(25) = 1.35$). Additionally, *plan-vis* participants reported SUS scores of 67.35 ($SD = 19.99$), whereas *no-vis* participants reported SUS scores of 59.00 ($SD = 20.68$). In the other measures that we compared, the *plan-vis* condition generally performed better on average than the *no-vis* condition within our sample (Figure 5). Average values for each measure under both conditions can be found in Figure 5. These results support the *Plan Visualizer* in increasing overall plan quality, but further investigation is required before accepting our hypotheses.

Robot Familiarity. We additionally analyzed each of our measures for associations with self-reported robot familiarity. Table 2 (left) shows the resulting Spearman's rank correlations, and Figure 5 visualizes these associations for each condition. It can be seen that as robot familiarity increases, plan quality, usability, and expectations matched also increase, while task load decreases. These correlations suggest that end-user programmers' past familiarity with robots may impact almost every interaction that they have with *Polaris*. Our manipulation may be competing with robot familiarity.

Plan Visualizer Usage. We grouped *plan-vis* participants (referred to as P*X*, with *X* being a unique identifier) into various categories based on how they used the *Plan Visualizer*. We categorized eight participants (P2, P5, P6, P7, P22, P25, P27, and P29) as "*intended use.*" These participants appeared to use the *Plan Visualizer* to validate their work or guide them in fixing errors and produced plans of high quality (feasibility score mean of 3.625 out of 4). We categorized an additional two participants (P16 and P30) as "*unsuccessful use*"

D.C. We discarded one participant's data (*plan-vis*) due to a rare but experience-altering software bug. Of the remaining 32 participants (17 *plan-vis*, 15 *no-vis*), the average age was 30.7 years ($SD = 11.7$) and the average self-reported familiarity with robots was 4.09 ($SD = 1.89$) on a seven-point, single-item Likert scale (low=1, high=7). Six participants reported participating in past robotics studies.

Within our sample, 14 participants (8 *plan-vis*, 6 *no-vis*) produced plans that were either correct or nearly correct (*feasibility* score of 3-4). Two participants (1 *plan-vis*, 1 *no-vis*) created goal automata that failed to compile. An additional three participants (1 *plan-vis*, 2 *no-vis*) experienced equipment failure at the time of plan execution. Due to these five participants not executing their plans on the robot, our post-execution measures (competence, expectations matched, and *runtime* score) include 27 participants in total.

Hypothesis Testing. We performed one-tailed Mann-Whitney U tests to compare plan quality and one-tailed Student's t-tests to

due to heavy reliance on the *Plan Visualizer* but an inability to fix errors in their goal automata (feasibility score mean of 0.5 out of 4). Four more participants (P4, P9, P13, and P21) fall into the "*no-use*" category due to not accessing the *Plan Visualizer* at all or accessing it but not interacting with it (*e.g.,* by not scrolling through or clicking on actions in the plan). *No-use* participants produced plans of low quality (feasibility score mean of 0.75 out of 4). We grouped the remaining three participants into a category of "*unknown use,*" for whom the role of the *Plan Visualizer* is unclear (feasibility score mean of 0.66 out of 4).

Within the *plan-vis* condition, there are strong positive associations between *Plan Visualizer* usage (equal to how many times a participant accessed the *Plan Visualizer*, and while accessing it, interacted with the *Plan Visualizer* as well) and expectations matched. Table 2 (right) depicts these correlations.

5 DISCUSSION

In our experiment, we found evidence that the *Plan Visualizer* increases plan quality and marginal effects for the *Plan Visualizer* increasing satisfaction, expectations matched, perceived usefulness of *Polaris*, and perceived competence of the robot. On average, the *plan-vis* condition generally performed better than the *no-vis* condition. We find these results encouraging, but further investigation is required to fully understand the *Plan Visualizer*'s effectiveness, and more generally, *Polaris* overall. To guide our further investigation and provide guidance to future research within the wider EUP community, we propose three design implications.

Design Implication: *Feedback is critical for goal-oriented EUP.* This implication is evidenced by the significant and marginal effects of the *Plan Visualizer* despite its underuse by *"no use"* and *"unknown use"* users. There also exists a strong positive association between *Plan Visualizer* usage and both expectations matched and usability. Although this association is not causal, we believe that improving *Plan Visualizer* access would have increased our observed effect. The importance of feedback and how information is presented to users is further supported by prior work [5, 41]. **Recommendation:** *Feedback should be provided proactively (rather than passively) by goal-oriented EUP tools. Polaris* users should be exposed to feedback as soon as changes to their programs occur, which could result in higher expectations matched for *"no use"* and *"unknown use"* users.

Design Implication: *Although goal-oriented programming allows for greater flexibility in theory (see §1) and has shown benefit in prior work [17, 27], users' ability to leverage this flexibility in practice should not be presumed.* As evidence of this implication, less than half of the participants in either condition produced correct or nearly correct plans. Participant interviews reveal a potential explanation: Goal predicates are difficult to reason about and require a shift in thinking from a potentially more intuitive (albeit less flexible) action-oriented paradigm (P9, P17, P19). **Recommendation:** *EUP researchers must investigate user-interface techniques that improve user comprehension of goal-oriented programming.* Crucially, EUP researchers should avoid assuming that current interface norms for EUP seamlessly translate to the goal-oriented paradigm.

Design Implication: *Robot familiarity strongly predicts perceptions and use of robot EUP tools.* As evidence of this implication,

our evaluation uncovers significant correlations between robot familiarity and usability, expectations matched, plan quality, and task load. We note that our study population is critical to uncovering this finding—our sample includes both professionals and students at both ends of the spectrum of robot familiarity. At the same time, the observed effects of our manipulation are potentially diluted due to the breadth of our study population. **Recommendation:** *EUP researchers for human-robot interaction need to choose their study population more carefully and deliberately than is often done in current practice.* Robot familiarity should factor into this choice.

Limitations and Future Work. Polaris' design poses various opportunities for improvement. Most notably, users need assistance creating "correct" plans (*i.e.,* plans with high feasibility scores). Future work should thus employ formal methods to help improve plan feasibility, such as by automatically detecting and fixing contradictory goals. Other limitations include that our planning approach does not account for uncertainty, such as if the location of an entity in the task context (*e.g.,* a person) is unknown or if there is an unknown number of multiple items of the same type. *Polaris*' inability to support loops or enumeration (*e.g.,* tasking the robot to deliver food to *all* rooms in the care facility) further limits the task contexts within which it can operate. For greater applicability in the wild, *Polaris* can also support plan adaptation, *e.g.,* by learning action costs and re-planning at runtime.

Further limitations exist in our evaluation. Primarily, our study is systems-level, focusing on the interaction between core components rather than exploring the benefit of increased flexibility from goal-oriented programming. Future component-level testing is already in preparation to understand the benefit of flexibility in practice. Additionally, we tested *Polaris* with just one scenario and a broad user group and did not collect data about how much training time is required for *Polaris*. Although our sample is critical to revealing significant associations with robot familiarity, future work must explore *Polaris* with its target user base, more realistic scenarios, and explore approaches to user training. We believe that our present study provides an excellent foundation for future testing, such as by deploying *Polaris in situ* with actual caregivers.

6 CONCLUSION

We present *Polaris*, a novel goal-oriented end-user programming (EUP) system. The purpose of *Polaris* is to provide flexibility to robot end users in the level of detail that programs are specified while ensuring that user expectations match robot performance. Our evaluation of *Polaris* uncovers evidence that plan feedback increases the quality of user-created programs. The evaluation also uncovers strong associations between plan feedback, robot familiarity, and participant experience and performance. We conclude with various design implications for the future development of EUP tools.

REFERENCES

[1] Angeline Aguinaldo, Jacob Bunker, Blake Pollard, Ankit Shukla, Arquimedes Canedo, Gustavo Quiros, and William Regli. 2022. RoboCat: A Category Theoretic Framework for Robotic Interoperability Using Goal-Oriented Programming. *IEEE Transactions on Automation Science and Engineering* 19, 3 (2022), 2637–2645. https://doi.org/10.1109/TASE.2021.3094055

[2] Gopika Ajaykumar, Maureen Steele, and Chien-Ming Huang. 2021. A Survey on End-User Robot Programming. *Comput. Surveys* 54, 8, Article 164 (oct 2021), 36 pages. https://doi.org/10.1145/3466819

[3] Sonya Alexandrova, Zachary Tatlock, and Maya Cakmak. 2015. RoboFlow: A Flow-based Visual Programming Language for Mobile Manipulation Tasks. In *2015 IEEE International Conference on Robotics and Automation (ICRA '15)*. IEEE, New York, NY, USA, 5537–5544. https://doi.org/10.1109/ICRA.2015.7139973

[4] Samir Alili, Rachid Alami, and Vincent Montreuil. 2009. *A Task Planner for an Autonomous Social Robot*. Springer Berlin Heidelberg, Berlin, Heidelberg, 335–344. https://doi.org/10.1007/978-3-642-00644-9_30

[5] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. 2019. Guidelines for Human-AI Interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300233

[6] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. MIT press, Cambridge, MA, USA.

[7] Barbara Rita Barricelli, Fabio Cassano, Daniela Fogli, and Antonio Piccinno. 2019. End-user development, end-user programming and end-user software engineering: A systematic mapping study. *Journal of Systems and Software* 149 (2019), 101–137. https://doi.org/10.1016/j.jss.2018.11.041

[8] Sara Beschi, Daniela Fogli, and Fabio Tampalini. 2019. CAPIRCI: A Multi-modal System for Collaborative Robot Programming. In *End-User Development (IS-EUD '19)*, Alessio Malizia, Stefano Valtolina, Anders Morch, Alan Serrano, and Andrew Stratton (Eds.). Springer International Publishing, Cham, 51–66. https://doi.org/10.1007/978-3-030-24781-2_4

[9] David Brageul, Slobodan Vukanovic, and Bruce A. MacDonald. 2008. An intuitive interface for a cognitive programming by demonstration system. In *2008 IEEE International Conference on Robotics and Automation*. IEEE, New York, NY, USA, 3570–3575. https://doi.org/10.1109/ROBOT.2008.4543757

[10] Michael Bratman. 1987. *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, USA.

[11] John Brooke. 1996. SUS-A 'quick and dirty' usability scale. *Usability Evaluation in Industry* 189, 194 (1996), 4–7. https://doi.org/10.1201/9781498710411

[12] Yuanzhi Cao, Tianyi Wang, Xun Qian, Pawan S. Rao, Manav Wadhawan, Ke Huo, and Karthik Ramani. 2019. GhostAR: A Time-Space Editor for Embodied Authoring of Human-Robot Collaborative Task with Augmented Reality. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 521–534. https://doi.org/10.1145/3332165.3347902

[13] Yuanzhi Cao, Zhuangying Xu, Fan Li, Wentao Zhong, Ke Huo, and Karthik Ramani. 2019. V.Ra: An In-Situ Visual Authoring System for Robot-IoT Task Planning with Augmented Reality. In *Proceedings of the 2019 on Designing Interactive Systems Conference* (San Diego, CA, USA) *(DIS '19)*. Association for Computing Machinery, New York, NY, USA, 1059–1070. https://doi.org/10.1145/3322276.3322278

[14] Colleen M. Carpinella, Alisa B. Wyman, Michael A. Perez, and Steven J. Stroessner. 2017. The Robotic Social Attributes Scale (RoSAS): Development and Validation. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction* (Vienna, Austria) *(HRI '17)*. Association for Computing Machinery, New York, NY, USA, 254–262. https://doi.org/10.1145/2909824.3020208

[15] Tathagata Chakraborti, Sarath Sreedharan, Sachin Grover, and Subbarao Kambhampati. 2019. Plan Explanations as Model Reconciliation – An Empirical Study. In *Proceedings of the 14th ACM/IEEE International Conference on Human-Robot Interaction* (Daegu, Republic of Korea) *(HRI '19)*. IEEE Press, New York, NY, USA, 258–266. https://doi.org/10.1109/HRI.2019.8673193

[16] Michael Jae-Yoon Chung, Justin Huang, Leila Takayama, Tessa Lau, and Maya Cakmak. 2016. Iterative Design of a System for Programming Socially Interactive Service Robots. In *Proceedings of Social Robotics: 8th International Conference (ICSR 2016)*, Arvin Agah, John-John Cabibihan, Ayanna M. Howard, Miguel A. Salichs, and Hongsheng He (Eds.). Springer International Publishing, Cham, 919–929. https://doi.org/10.1007/978-3-319-47437-3_90

[17] Michael T. Cox and Chen Zhang. 2007. Mixed-Initiative Goal Manipulation. *AI Magazine* 28, 2 (Jun. 2007), 62. https://doi.org/10.1609/aimag.v28i2.2040

[18] Giuseppe De Giacomo, Alfonso Emilio Gerevini, Fabio Patrizi, Alessandro Saetti, and Sebastian Sardina. 2016. Agent planning programs. *Artificial Intelligence* 231 (2016), 64–106. https://doi.org/10.1016/j.artint.2015.10.001

[19] Emanuele De Pellegrin and Ronald P.A. Petrick. 2021. Automated Planning and Robotics Simulation with PDSim. In *Proceedings of the ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

[20] Maxwell Forbes, Rajesh P. N. Rao, Luke Zettlemoyer, and Maya Cakmak. 2015. Robot Programming by Demonstration with Situated Spatial Language Understanding. In *2015 IEEE International Conference on Robotics and Automation (ICRA '15)*. IEEE, New York, NY, USA, 2014–2020. https://doi.org/10.1109/ICRA.2015.7139462

[21] Maria Fox and Derek Long. 2003. PDDL2. 1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research* 20 (2003), 61–124. https://doi.org/10.1613/jair.1129

[22] Malik Ghallab, Dana Nau, and Paolo Traverso. 2016. *Automated Planning and Acting*. Cambridge University Press, Cambridge, England.

[23] Javi F. Gorostiza and Miguel A. Salichs. 2011. End-user programming of a social robot by dialog. *Robotics and Autonomous Systems* 59, 12 (2011), 1102–1114. https://doi.org/10.1016/j.robot.2011.07.009

[24] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In *Human Mental Workload*, Peter A. Hancock and Najmedin Meshkati (Eds.). Advances in Psychology, Vol. 52. Elsevier B.V., Amsterdam, Netherlands, 139–183. https://doi.org/10.1016/S0166-4115(08)62386-9

[25] Malte Helmert. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26, 1 (2006), 191–246. https://doi.org/10.1613/jair.1705

[26] Malte Helmert. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *Artif. Intell.* 173, 5–6 (apr 2009), 503–535. https://doi.org/10.1016/j.artint.2008.10.013

[27] Minjie Hu, Michael Winikoff, and Stephen Cranefield. 2012. Teaching Novice Programming Using Goals and Plans in a Visual Notation. In *Proceedings of the Fourteenth Australasian Computing Education Conference - Volume 123* (Melbourne, Australia) *(ACE '12)*. Australian Computer Society, Inc., AUS, 43–52. https://dl.acm.org/doi/abs/10.5555/2483716.2483722

[28] Justin Huang and Maya Cakmak. 2017. Code3: A System for End-to-End Programming of Mobile Manipulator Robots for Novices and Experts. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction* (Vienna, Austria) *(HRI '17)*. Association for Computing Machinery, New York, NY, USA, 453–462. https://doi.org/10.1145/2909824.3020215

[29] Justin Huang, Tessa Lau, and Maya Cakmak. 2016. Design and Evaluation of a Rapid Programming System for Service Robots. In *The 11th ACM/IEEE International Conference on Human Robot Interaction* (Christchurch, New Zealand) *(HRI '16)*. IEEE Press, New York, NY, USA, 295–302. https://doi.org/10.1109/HRI.2016.7451765

[30] Silvia Izquierdo-Badiola, Gerard Canal, Carlos Rizzo, and Guillem Alenyà. 2022. Improved Task Planning through Failure Anticipation in Human-Robot Collaboration. In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, New York, NY, USA, 7875–7880. https://doi.org/10.1109/ICRA46639.2022.9812236

[31] Charles C. Kemp, Aaron Edsinger, Henry M. Clever, and Blaine Matulevich. 2022. The Design of Stretch: A Compact, Lightweight Mobile Manipulator for Indoor Human Environments. In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, New York, NY, USA, 3150–3157. https://doi.org/10.1109/ICRA46639.2022.9811922

[32] Matthias Kovatsch, Yassin N. Hassan, and Simon Mayer. 2015. Practical semantics for the Internet of Things: Physical states, device mashups, and open questions. In *2015 5th International Conference on the Internet of Things (IOT)*. IEEE, New York, NY, USA, 54–61. https://doi.org/10.1109/IOT.2015.7356548

[33] Alyssa Kubota, Emma I. C. Peterson, Vaishali Rajendren, Hadas Kress-Gazit, and Laurel D. Riek. 2020. JESSIE: Synthesizing Social Robot Behaviors for Personalized Neurorehabilitation and Beyond. In *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction* (Cambridge, United Kingdom) *(HRI '20)*. Association for Computing Machinery, New York, NY, USA, 121–130. https://doi.org/10.1145/3319502.3374836

[34] Nicola Leonardi, Marco Manca, Fabio Paternò, and Carmen Santoro. 2019. Trigger-Action Programming for Personalising Humanoid Robot Behaviour. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300675

[35] Ying Siu Liang, Damien Pellier, Humbert Fiorino, and Sylvie Pesty. 2019. End-User Programming of Low-and High-Level Actions for Robotic Task Planning. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE, New York, NY, USA, 1–8. https://doi.org/10.1109/RO-MAN46459.2019.8956327

[36] Yuan-Hong Liao, Xavier Puig, Marko Boben, Antonio Torralba, and Sanja Fidler. 2019. Synthesizing Environment-Aware Activities via Activity Sketches. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, New York, NY, USA, 6284–6292. https://doi.org/10.1109/CVPR.2019.00645

[37] Arnold M Lund. 2001. Measuring Usability with the USE Questionnaire. *Usability and User Experience Newsletter of the STC Usability SIG* 8, 2 (2001), 3–6.

[38] Simon Mayer, Ruben Verborgh, Matthias Kovatsch, and Friedemann Mattern. 2016. Smart Configuration of Smart Environments. *IEEE Transactions on Automation Science and Engineering* 13, 3 (2016), 1247–1255. https://doi.org/10.1109/TASE.2016.2533321

[39] Mahda Noura, Sebastian Heil, and Martin Gaedke. 2018. GrOWTH: Goal-Oriented End User Development for Web of Things Devices. In *Web Engineering*, Tommi Mikkonen, Ralf Klamma, and Juan Hernández (Eds.). Springer International

Publishing, Cham, 358–365. https://doi.org/10.1007/978-3-319-91662-0_29

[40] Ronald P. A. Petrick and Mary Ellen Foster. 2013. Planning for Social Interaction in a Robot Bartender Domain. In *Proceedings of the Twenty-Third International Conference on International Conference on Automated Planning and Scheduling* (Rome, Italy) *(ICAPS'13)*. AAAI Press, Washington, DC, USA, 389–397. https://doi.org/10.1609/icaps.v23i1.13589

[41] David Porfirio, Allison Sauppé, Aws Albarghouthi, and Bilge Mutlu. 2018. Authoring and Verifying Human-Robot Interactions. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology* (Berlin, Germany) *(UIST '18)*. Association for Computing Machinery, New York, NY, USA, 75–86. https://doi.org/10.1145/3242587.3242634

[42] David Porfirio, Laura Stegner, Maya Cakmak, Allison Sauppé, Aws Albarghouthi, and Bilge Mutlu. 2023. Sketching Robot Programs On the Fly. In *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction* (Stockholm, Sweden) *(HRI '23)*. Association for Computing Machinery, New York, NY, USA, 584–593. https://doi.org/10.1145/3568162.3576991

[43] David J. Porfirio, Laura Stegner, Maya Cakmak, Allison Sauppé, Aws Albarghouthi, and Bilge Mutlu. 2021. Figaro: A Tabletop Authoring Environment for Human-Robot Interaction. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3411764.3446864

[44] Emmanuel Pot, Jérôme Monceaux, Rodolphe Gelin, and Bruno Maisonnier. 2009. Choregraphe: a Graphical Tool for Humanoid Robot Programming. In *The 18th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN '09)*. IEEE, New York, NY, USA, 46–51. https://doi.org/10.1109/ROMAN.2009.5326209

[45] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Y Ng. 2009. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*.

[46] Anand S. Rao. 1996. AgentSpeak(L): BDI Agents speak out in a logical computable language. In *Agents Breaking Away: 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Walter Van de Velde and John W. Perram (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 42–55. https://doi.org/10.1007/BFb0031845

[47] Sebastian Sardina and Lin Padgham. 2011. A BDI Agent Programming Language with Failure Handling, Declarative Goals, and Planning. *Autonomous Agents and Multi-Agent Systems* 23, 1 (jul 2011), 18–70. https://doi.org/10.1007/s10458-010-9130-9

[48] Allison Sauppé and Bilge Mutlu. 2014. Design Patterns for Exploring and Prototyping Human-Robot Interactions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 1439–1448. https://doi.org/10.1145/2556288.2557057

[49] Andrew Schoen, Curt Henrichs, Mathias Strohkirch, and Bilge Mutlu. 2020. Authr: A Task Authoring Environment for Human-Robot Teams. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 1194–1208. https://doi.org/10.1145/3379337.3415872

[50] Andrew Schoen, Nathan White, Curt Henrichs, Amanda Siebert-Evenstone, David Shaffer, and Bilge Mutlu. 2022. CoFrame: A System for Training Novice Cobot Programmers. In *Proceedings of the 2022 ACM/IEEE International Conference on Human-Robot Interaction* (Sapporo, Hokkaido, Japan) *(HRI '22)*. IEEE Press, New York, NY, USA, 185–194. https://doi.org/10.1109/HRI53351.2022.9889345

[51] Vikas Shivashankar, Ugur Kuter, Dana Nau, and Ron Alford. 2012. A Hierarchical Goal-Based Formalism and Algorithm for Single-Agent Planning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2* (Valencia, Spain) *(AAMAS '12)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 981–988. https://dl.acm.org/doi/abs/10.5555/2343776.2343837

[52] David E Smith, Jeremy Frank, and William Cushing. 2008. The ANML Language. In *The ICAPS-08 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

[53] Laura Stegner and Bilge Mutlu. 2022. Designing for Caregiving: Integrating Robotic Assistance in Senior Living Communities. In *Proceedings of the 2022 ACM Designing Interactive Systems Conference* (Virtual Event, Australia) *(DIS '22)*. Association for Computing Machinery, New York, NY, USA, 1934–1947. https://doi.org/10.1145/3532106.3533536

[54] Unity Technologies. 2023. Unity Real-Time Development Platform. https://unity.com/.

[55] Karthik Valmeekam, Sarath Sreedharan, Sailik Sengupta, and Subbarao Kambhampati. 2022. RADAR-X: An Interactive Mixed Initiative Planning Interface Pairing Contrastive Explanations and Revised Plan Suggestions. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 32. AAAI Press, Washington, DC, USA, 508–517. https://doi.org/10.1609/icaps.v32i1.19837