A Multi-Robot Architecture Framework for Effective Robot Teammates in Mixed-Initiative Teams

Matthias Scheutz
Tufts University
Medford, USA
matthias.scheutz@tufts.edu

Bradley Oosterveld
Thinking Robots
Boston, USA
brad@thinkingrobots.ai

John Peterson
Thinking Robots
Boston, USA
pete@thinkingrobots.ai

Eric Wyss
Thinking Robots
Boston, USA
eric@thinkingrobots.ai

Evan Krause
Tufts University
Medford, USA
evan.krause@tufts.edu

ABSTRACT

Effective robotic teammates should be able to interact with humans in natural language about all task aspects, keep track of task and team states to coordinate their actions, and handle unexpected events autonomously. In this paper, we introduce a multi-robot architectural framework for effective robot teammates that allows robots to learn new tasks on the fly and monitor task execution to be able to detect unexpected faults and events. It enables robots to generate recovery plans, assess their effectiveness, and engage with human teammates in problem solving dialogues. We demonstrate the capabilities and operation of the framework in a complex mixed-initiative human-robot medical assembly and delivery task.

CCS CONCEPTS

• **Computing methodologies → Discourse, dialogue and pragmatics**; **Robotic planning**.

KEYWORDS

cognitive multi-robot architecture, human-machine teaming, HRI

ACM Reference Format:
Matthias Scheutz, Bradley Oosterveld, John Peterson, Eric Wyss, and Evan Krause. 2024. A Multi-Robot Architecture Framework for Effective Robot Teammates in Mixed-Initiative Teams. In *2024 International Symposium on Technological Advances in Human-Robot Interaction (TAHRI 2024), March 9–10, 2024, Boulder, CO, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3648536.3648545

1 INTRODUCTION

Mixed-initiative human-robot teams are envisioned to be deployed in many application domains on the ground, in the air, under water, on Earth and in Space, in a wide variety of teaming tasks ranging from collaborative manufacturing, to search and rescue missions, to exploratory expeditions on remote planets, and many others. Common to all applications is the hope that robots will reach a level of task understanding, problem solving, proactivity and natural interactions that makes them genuine teammates, not just tools.

Despite exciting progress in robot perceptions, navigation and manipulation, when it comes to teaming contexts, current autonomous systems fundamentally lack almost everything required from a genuine teammate: they are unaware of team capabilities, tasks, and goals, they fail to cope with execution failures that need to be addressed on the fly to proceed with the task performance, and they lack the ability to interact with humans and adapt their behaviors based on team dynamics. As a result, mixed-initiative human-robot teams are still limited by the robots' lack of team awareness and lack of autonomous problem solving. Genuine artificial teammates, instead, would understand dynamically shifting team and teammate goals, anticipate problems and plan around them, cope with errors when they occur, and proactively intervene in order to ensure successful task performance.

What is needed is an AI architectural framework that allows for the integration of the cutting-edge robotic algorithms for perception, navigation, and manipulation, while providing high-level AI capabilities for planning and reasoning as well as natural language dialogue interaction for operating effectively in human teams. This paper introduces such an architecture and details in a concrete multi-human multi-robot mixed-initiative team task how the various architectural features come together to support human-machine teaming in light of unexpected events during task performance.

We describe a novel integrated multi-robot architectural framework that can operate any number of heterogeneous autonomous robots with different hardware, operating systems, and APIs, that has the following capabilities:

- interactive task learning from human instructions with immediate task performance and execution monitoring
- detection of plan execution failures and natural language failure explanations
- introspection into system-wide resources to develop recovery plans and provide automatic performance assessments
- problem solving dialogues with human teammates to determine the best possible plan alternative

After motivating key architectural capabilities, and showing that no current architecture provides all of them, we describe the proposed framework in detail and demonstrate its operation and features in a mixed-initiative multi-human multi-robot medical supply assembly and delivery task.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

TAHRI 2024, March 9–10, 2024, Boulder, CO, USA

© 2024 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1661-4/24/03
https://doi.org/10.1145/3648536.3648545

2 MOTIVATION AND RELATED WORK

In some sense, a robotic teammate would ideally be like many of the fictional robots featured in sci-fi movies: while possibly different from humans in appearance, they would be human-like in their cognitive capabilities, their understanding of the mission, and their ability to cope with impediments to task performance. While human-like cognition is too high a bar to reach in the foreseeable future, it is very feasible to develop some key capabilities that will make human-robot interactions in mixed-initiative human-robot teams more natural for humans and more effective for the team.

First and foremost, robots should allow for **task-based natural language interactions** because natural language is the most comprehensive, preferred way of human communication (although there are also other ways including facial expressions such frowning to signal disapproval, gestures to point to locations in the environment where actions need to be performed, and others). This includes dialogues about mission goals and task assignments, execution status and progress, and potential performance impediments and alternatives (e.g., [22]).

Second, robots should build and maintain **shared mental models** (SMMs) (e.g., [19]) that allow them to track the state of the team, the environment, and ongoing activities in order to adapt their behaviors to the team's needs. This includes the ability to proactively develop new plans, adapt existing plans when changes occur, and to monitor plan execution to detect such deviations and faults. This also includes using their SMM to update human teammates on information they might require for their activities but which they are not aware of or do not have (e.g., [15]).

Finally, robots should exhibit a certain level of **autonomy**, not only in terms of their navigation and manipulation behaviors, but more importantly at the level of their cognitive abilities to propose hypothetical and alternative plans when execution of the current one is no longer possible, and provide assessments of their effectiveness through problem solving dialogues with humans. This also includes being able to handle any hardware or software adaptations necessary for the pursuit of those plans so as to not bother human teammates with the minutia of robot configurations and potentially overburden them with additional robot troubleshooting tasks unless absolutely necessary (e.g., [9]).

What kind of architecture provides the above capabilities? A first place to look might be classical cognitive architectures enabled to operate on robots, like SOAR (e.g., [13]) and ACT-RE (e.g., [21]) as they provide higher-level capabilities such as task learning through natural language dialogues (SOAR) or the development of mental models (ACT-RE). But the autonomy they provide to robots is limited as they cannot deal well with real-time action or different types of execution failures, and they cannot operate multiple robots.

Robotic architectures, on the other hand, have traditionally been distributed (due to real-time requirements), using middleware to support the distribution (such as ROS [18], JAUS [20], YARP [14], etc.). They integrate various algorithms for real-time perceptual, planning and action processing, from 3D object recognition, to simultaneous localization and mapping, to navigation, task, and manipulation planning, to action sequencing. But they typically lack components for high-level cognition such as common sense reasoning, problem solving, and any form of team cognition.

Most recently, proposals have been advanced to combine large language models (LLMs) with robotic architecture to make robots instructible (e.g., see the attempts by Google, Microsoft [23],and others [2]). Yet, there are two main challenges to be addressed with such systems: (1) There is no currently no proposal for a sufficient integration of perceptual streams coming from robots into LLMs in a way that would allow them to detect and react to task-relevant changes in the environment without overwhelming them (e.g., changes that should trigger re-planning, informing the human, etc.). The same goes for integrating control processes (the current model seems to be to just issue high-level commands to the platform, but that limits the utility of LLMs and the reason why one would want to use them in the first place, e.g., for their knowledge about "how to do something"). And (2), natural language interactions are *far too slow* for spoken human dialogues, especially in potentially time-sensitive situations (a response time upwards of 6 seconds when humans usually only tolerate at most one second because of the processing time it takes for the LLM to generate outputs and verbal feedback). In general, there is an intrinsic lack of real-time appreciation (shared with classical cognitive architectures) that is essential in many robotic domains as the world does not unfold according to LLM update cycles. This is all in addition to their lack of predictability (e.g., LLMs generate their outputs based on distributional information and might "hallucinate" results) which can obviously be dangerous in robotic domains.

In contrast to the above, we propose a polylithic architecture framework that combines the strengths of high-level cognition such as reasoning, planning, introspection, and natural language with a real-time-aware distributed robotic architecture that can operate multiple heterogeneous robots in parallel and coordinate their actions through a shared mental model that also keeps track of task and environmental states as well as human task-based beliefs. The architecture is based on "deep introspection" into its operation that allows human teammates to have dialogues about its operational status, the capabilities of the robots it controls, and the possible problem solving strategies it can pursue with the robots. Like SOAR, the architecture allows for interactive task learning through natural language dialogues, but unlike SOAR it can coordinate the actions of multiple robots through online reasoning about their capabilities and report success likelihoods and time estimates about possible plans. Moreover, the architecture monitors its task execution in order to detect potential faults, and different from cognitive architectures, has elaborate failure recovery mechanisms to get past unexpected events during task execution.

3 ARCHITECTURE FUNCTIONALITY FOR EFFECTIVE TEAMING

After a brief overview of the architecture which enable the three essential features of effective robotic teammates, i.e., task-based natural language interactions, shared mental models, and autonomy, we focus on interactive task specifications, multi-robot planning, plan execution monitoring and fault recovery.

3.1 The Multi-Robot Architecture Framework

Fig. 1 shows the proposed multi-robot shared-mental model architecture framework for effective robot teammates with white

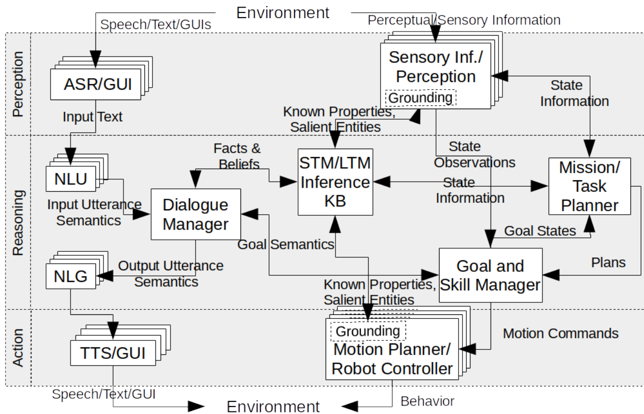

Figure 1: The architecture framework for effective robot teammates. Labels on white boxes indicate component functionality, the labels on arrows indicate the types of data sent across components. Stacked components indicate that multiple instances of the component type can be present in the architecture (see text for details).

boxes representing components that operate in parallel in a distributed fashion (i.e., potentially on different computers) with labels indicating component functionality (e.g., ASR/GUI=automatic speech recognizer/graphical user interface, NLU=natural language understanding, STM/LTM=short-term memory/long-term memory, KB=knowledge base, NLG=natural language generation, TTS= text to speech). *Stacked components* (e.g., everything but the Dialogue Manager, STM/LTM, the Goal and Skill Manager, and the Mission/Task planner in this particular configuration) can have multiple instances of the same component type present in an architecture instance which enables multi-human multi-robot interactions in a systematic way.[1] For each human teammate, the architecture instantiates separate ASR, NLU, NLG, and TTS components which can run on a separate device (e.g., a cell phone carried by the human or a desktop PC) to distribute computational load and parallelize natural language processing, which is essential to keep the response time as short as possible as the number of human teammates grows (as mentioned, this is a major problem for current LLMs). Similarly, for each robot, the architecture instantiates separate Sensory Inference/Perception components and Motion Planning/Robot Controller components, which usually run directly on their target platform with direct access to devices, enabling a critical level of load distribution required for system scalability as the number of robot teammates increases.

The four single-instance components then form the nexus between the natural language and the perception-action subsystems, enabling shared mental models and team coordination. The Dialogue Manager ("DM") is in charge of managing all dialogue interactions with human teammates and can track those interactions, i.e., what each person said at what time and what their utterance means for the team.[2] Such state updates can then be committed to the STM/LTM/Inference/KB ("SLIK") which implements a shared knowledge repository and inference system used by all robots that directly implements shared mental models by keeping track of task and team states, but also beliefs and goals of human teammates.[3]

The Goal and Skill Manager ("GSM") contains a database of "Skills" which are semantically annotated sequences of calls to effector components (TTS/GUI, Robot Controllers) that result in interaction with the relevant environment, as well as a collection of "Observations" which are semantically annotated calls to perceptual components that can verify if the execution of a Skill resulted in the intended change to the world state (e.g. did the execution of the "putOn(object,table)" skill result in the object being visible on the table?). GSM also keeps track of all system goals and plans and coordinates their execution on the connected robotic hardware.

The Mission and Task Planner ("MTP") generates plans for goals and constraints sent by the GSM. MTP handles the conversion of state and goal information represented in architecture semantics into the relevant representation required for the task planner being used by the architecture instance. The resulting plans are then translated back from the planner specific representation by MTP, and added to GSM as new Skills. In addition to introspecting on the individual steps which make up the execution of a Skill, GSM also introspects on the overall execution of the Skills required to achieve a given goal. It generates and updates performance assessments of plans, based on current execution and monitors their execution to detect any potential faults via Observations. If a fault is detected, the GSM attempts to mitigate it (e.g., through re-planning).

All four components—DM, SLIK, GSM, and MTP—interact closely during problem solving dialogues, when human teammates, for example, ask for hypothetical plans that address changes in goals or in environmental settings.

New architectural components can be instantiated and existing ones removed at run-time allowing, for example, for a set of natural language processing components to be instantiated when a new team human member joins with their communication device. Similarly, a new set of robot components will be instantiated when a new robot joins the running architecture instance in which case the GSM will discover the capabilities of the robot platform through introspection on the functionality of the robot's perception and actuation components. In this sense, different robots are treated as different "bodies" of the "meta-agent" implemented by the multi-robot architecture framework.

3.2 Knowledge Representation

To utilize the functionality of a task planner, introspect on its own performance, and accommodate the dynamic addition and removal

[1] In general, any component can be present multiple times in the system based on teaming needs, e.g., with multiple goal and dialogue managers, reasoners and planners for groups of robots that do not always need information from other groups and can plan independently, allowing for parallel planning, reasoning, and interactions with humans–in the limit case, all robots would have their own components/architectures without any shared components. But having the above four shared components allows for effective information sharing as required for SMMs without the need for explicit communication and synchronization among robots. It is, however, sometimes useful to have for each shared component a "backup" component that can step in should the host on which the shared component resides become unreachable.

[2] Note that since interactions can occur in parallel, conflicts can occur and must be resolved by the centralized SLIK components where assertions and retractions of facts and rules are forced to be sequential and can thus be deconflicted. With multiple SLIKs, a distributed truth maintenance would be needed to ensure consistent information, which is beyond the scope of this paper.

[3] SLIK in our case uses Prolog or ASP, but any KB/reasoning system can be used as long as its representations can be translated into formats used by other components.

of components, the architecture must adhere to a consistent knowledge representation which can accommodate both symbolic and non-symbolic knowledge. And the architecture must provide mechanisms for this information to be accessed in a distributed fashion as the architecture itself is distributed.

SLIK is the locus of symbolic knowledge, storing not only information about the world, but also the format it uses to represent that symbolic information (which is based on an augmented first-order logical language). The architecture provides mechanisms which allow individual components to map component-specific information about the world into this shared symbolic representation.

This core symbolic knowledge is composed of the Skills the architecture can perform, the "Conditions" of those skills (the world state that needs to be true in order for the Skill to be executed), the "Effects" of executing a given Skill (changes to the world state), the collection of Observations about world states that the architecture can make, as well as the real world "Entities" on which Skills, Conditions, Effects, and Observations operate, and the perceivable "Properties" of those entities. Additionally, SLIK contains purely symbolic knowledge that defines the semantic type system used by the Skills, Conditions, Effects, Observations and Entities, as well as any additional purely symbolic knowledge that is relevant to the task domain (e.g., Alex is a trusted human interactant, and the architecture should carry out their commands).

Since the above symbolic information can represent non-symbolic component type information that is specific to the operation of a component, the architecture provides generic mechanisms that individual components can use to appropriately ground their relevant internal, possibly non-symbolic information.

In the case of Skills, the architecture provides a generic mechanism whereby component method calls that result in a given behavior on the associated robot can be denoted, via semantic annotations, as executable Skills by the GSM. The GSM can associate Conditions and Effects, represented as logical predicates that are valid within the given instance's knowledge representation defined in SLIK, with these atomic "Primitive Skills". The GSM tracks which Skills are associated with which components and the associations between those components and individual hardware instances.

Where Skills are denoted by semantic annotations on effector components, Entities and their Properties are denoted by semantic annotations on perceptual components. The architecture provides an interface to perceptual components, whose implementation guarantees that their resulting precepts can be used within the architecture instance's knowledge representation. This interface includes methods for getting a collection of all of the Properties a component can perceive, getting a collection of all known entities with a given Property or set of Properties, and checking whether a given entity has a, heretofore unknown, Property. The types of Entities that a given component can ground, and their relevant Properties may vary significantly depending on the component. One component may use a vision sensor to detect objects and their grasp points for manipulation tasks. In this case, Properties would include the type of object (e.g., box, bottle, basket, etc.) as well as features of the object (e.g., color, shape). Another component may handle Entities which represent coordinates on the map within a robot's navigation system. Their properties might include names for waypoints or other semantic mapping information like the room that it is in.

Some Conditions and Effects are purely symbolic, and their truth values can be assesed by directly querying SLIK, however others represent changes to the state of the world resulting from robot behavior. In these cases they cannot be assumed to hold, and instead their truth value must be observed from the environment. Similar to Skills, the architecture provides mechanisms by which component method calls can be given semantic annotations allowing them to act as Observations of specified world states. Depending on its component and associated methods, the form of an Observation may vary drastically. A robot component may observe the state predicate $at(robot, location)$ which may be the Effect of executing a navigation Skill. In this predicate $robot$ represents the identifier of the robotic hardware of the component, and $location$ is an Entity representing a waypoint in the robot's navigation system. When the Observation is performed, the component checks that the robot's current position on the map is within a threshold distance of the coordinates of the waypoint. If the difference in positions is beneath the threshold, the Effect holds, and is asserted to SLIK, if not the execution of the navigation action has not had the intended effect, and the architecture deems it a failure. Other Observations may take different forms like $holding(robot, object)$ or $on(table, object)$, but their role in the system is the same.

3.3 Interactive Task Specification

We have previously developed mechanisms that enable autonomous agents to learn how to recognize new Entities using one-shot object learning and how to use one-shot learning to acquire new Skills through natural language dialogues, and how this new knowledge could later be easily modified and updated.[5, 6, 12, 16] However, the past work was limited to the instruction of individual Skills and teaching interactions were completely driven by human instructors, requiring them to have a good understanding of the architecture's capabilities and knowledge representation to be able to specify Conditions and Effects for each Skill.

To alleviate this burden on human instructors, we have extended the architecture's task learning capabilities through the use of "Interaction Templates", which allow the architecture to take a more active role in the learning process. Additionally, these Interaction Templates allow for task representations to be structured as goal world states to be achieved by MTP. Re-framing a Task as a desired world state instead of a step by step process decreases the amount of knowledge the human teacher needs to impart, and as such decreases the amount of knowledge required of the human.

Interaction Templates are themselves implemented as Skills which provides them access to knowledge distributed throughout the architecture. Interaction Templates can be tailored to specific task domains to improve task performance through a set of task-relevant assumptions. The remainder of this section describes the general form of Interaction Templates. A task specific interaction template can be found in the Demonstration section.

As with prior methods for human-prompted teaching, execution is triggered with an initial utterance from the human instructor. This prompt should contain information about **which template to use** and a **unique label** for the new goal state that will be produced. The architecture then takes over and leverages GSM and NLG to ask for relevant semantic information to verify that that

resulting goal state is valid and usable. The first piece of relevant information is a definition of the **goal state**, a proposition or conjunction of propositions that define the desired world state the teacher wants the system to achieve. In order for this new goal state to be meaningful for the system, each predicate that occurs in goal descriptions must be **groundable** onto a piece of knowledge with in the architecture's knowledge representation, which in this case means it must be the **Effect of a Skill** because the execution of Skills is the manner in which the architecture instance can update its information about the world. As such, the architecture instances checks if every conjunct in the new goal state is present in SLIK. If not, it requests that information from the human.

Like an Interaction Template, the process of requesting information from a human is also implemented as a Skill. This Skill takes in a target to request the information from, semantics it can use to represent the request in Natural Language, and the form of the resulting response semantics.

Once all of the predicates in the goal state have been validated, the interaction template may then use coarse heuristics to validate that goal is potentially achievable by the system. Specific heuristics may vary based on the task domain, but typically when Entities or Properties are involved in the goal state the Interaction Template will check if instances of those Entities or Properties are currently present in SLIK. If not, it will request additional information from the human about the steps required for it to perceive those entities, and assert the resulting goal representation to SLIK. The Interaction Template also provides information to NLU and GSM about how the new goal state can be instructed by the human.

Algorithm 1 Generic Interaction Template

```
1: given label from human
2: g = askQuestion(human,utterance,goalForm)           ▷ ask human for goal state
3: for predicate p in g do
4:     if notValid(p) then
5:         askQuestion(human,utterance,pForm)          ▷ ask for clarification on unknown component of goal state
6:     end if
7:     for predicate a in p.getArgs()) do
8:         if notKnown(a) then
9:             askQuestion(human,utterance,aForm)      ▷ ask how currently unknown Entity can be found
10:        end if
11:    end for
12: end for
13: assertDefintion(label)                             ▷ Adds knowledge to SLIK
14: updateVocuabulary(label)                           ▷ Updates GSM and NLU
```

3.4 Multi-Robot Task Planning

Planning problems are represented using the Planning Domain Definition Language (PDDL) [3] by a planning *domain* and a *problem instance*. A PDDL domain $\langle T, P, A\rangle$ describes the structure of a class of problems where T is a set of types, P is a set of predicates, and A is a set of actions. A predicate $p(t_1, \ldots, t_n)$ for types $t_1, \ldots, t_n \in T$ defines a relation that may hold between grounded objects of the corresponding types. A PDDL action $a \in A$ similarly has a signature with typed arguments $a(t_1, \ldots, t_k)$, along with preconditions $\mathrm{pre}(a)$, positive effects $\mathrm{add}(a)$ and negative effects $\mathrm{del}(a)$.

A PDDL problem instance $\langle \Omega, I, G\rangle$ provides a set of typed objects Ω, initial state I and goal condition G specified as grounded predicates. Along with its domain, a problem instance fully specifies a grounded planning problem. PDDL problems are solved using planners, which find valid grounded plans to achieve the goal condition. The architecture can dynamically generate planning domains and problem instances, and subsequently execute grounded plans, by introspectively accessing its internal knowledge representations and mapping it onto planning problems. Specifically, when a goal state g is submitted to GSM, the current world state as represented in SLIK as well as the knowledge about available skills (that are mapped onto planning domain actions in GSM) at submission time are used to generate a PDDL planning domain $D_{goal} = \langle T, P, A\rangle$ and problem instance $\Theta_{goal} = \langle \Omega, I, G\rangle$ where G corresponds to the submitted goal.[4]

The first step in this process involves introspecting on GSM in order to define A, the set of actions in the domain by mapping Skills to PDDL actions. A skill in the GSM can be denoted $\langle s, c, e\langle \sigma, f\rangle, b\rangle$, where s represents the Skill's signature including its arguments and their types, c represents the conditions that need to be true in order for the skill to be executed, e defines the effects of executing the Skill successfully (σ) or failing to execute it (f), and b represents the body of the skill. Each Skill in GSM is used to generate a new PDDL action a where s maps on to the signature of a, c onto $\mathrm{pre}(a)$, and e onto $\mathrm{add}(a)$ and $\mathrm{del}(a)$.

In the case where multiple robot components exist in the architecture instance, all of their Skills are available in GSM. Typically, the representation of the correspondence between skills and their associated robot agent is accomplished through the standardized use of an *actor* role in the signature of a skill. If the system has heterogeneous hardware platforms connected, a semantic type can be provided for the actor role to define which Skills will be executed on which hardware.

While A is being generated, the system creates a record of all of the predicate forms present in the conditions and effects used to populate A. This set of predicates K represents the symbolic knowledge and corresponding world state information that could potentially be modified and observed through the execution of a plan to accomplish the specified goal. As such, the elements of K contribute to the definition of P and I for the domain and problem instance.

Entities and their Properties are incorporated into the PDDL problem and instance via the standard interface provided by the architecture for representing non-symbolic information symbolically. We can denote the set of components in an architecture instance that provide grounding information Φ, where for $\phi \in \Phi$, $\phi = \langle \Gamma, E\rangle$, where Γ represents the set of Properties for which the component can produce symbolic/non-symbolic bindings, given as descriptive logical predicates whose arguments are semantically-typed free variables, and where E represents the component's set of currently known Entities. For each ϕ, the subset of E whose properties appear in K are added to Ω, their associated semantic types to T, and their properties which hold to I.

For information that is purely symbolic and cannot be grounded onto the external world, the system adds to I and P again using the

[4] Note that while the SMM enables centralized planning and thus simplifies the planning problem, nothing hinges on it and decentralized multi-agent planning methods could be employed instead.

information in K. The architecture queries SLIK for information in K that was not provided via grounding components. SLIK stores facts and rules used for inference in various locations within the system. SLIK additionally stores the semantic type hierarchy used by the predicates in K, and T and P are populated with the relevant type information based on the contents of K.

A particular planning problem generated for a state-based goal submitted either by a human, or as part of a larger learned behavior, captures a *deterministic planning problem* with a *joint action set* across the *multiple embodied robots* contributing to the human-robot team. A problem solution takes the form of a deterministic joint plan, where robot agents are coordinated within the problem solution to take advantage of skills available to all robots, and optimizing for which robots can be most efficiently used to satisfy different requirements within a planning goal.

While plan solutions to these problems necessarily assume deterministic state transitions and a closed-world, they are effective as a tool for problem solving, communication, and autonomy for robotic agents when working as a teammate with a human. Deterministic joint plans can be easily communicated to the human as a linear sequence of actions that will be taken, and each action taken as part of the joint plan is associated with the embodied robotic agent which will execute the corresponding skill. The compactness and communicability of these plans enable direct communication with human teammates, allowing the system to incorporate behaviors such as informing the human about plans before execution, and asking permission to execute plans if authorization is needed, including by estimating the execution time of a plan based on historical data.

3.5 Execution Monitoring and Fault Recovery

When executing generated plans, the architecture accounts for exogenous changes in state and external sensing through the use of Observations of the conditions and effects of the Skills corresponding to actions in the plan. These Observations work to ensure that when closed-world assumptions made in plans are violated, discrepancies can be caught early, and with specific knowledge about which facts failed to hold based on perceptions.

The architecture is able to introspectively determine the success or failure of every Skill it executes, including cases where an Observation triggers a failure. In cases where a Skill fails, the "Recovery Manager", a sub component of GSM, may execute one of a set of recovery polices which dictates how the architecture instance might proceed to resolve the failure. GSM is able to detect failure during several stages of execution: (1) during planning, and during plan execution as (2) skill pre-condition checks, (3) primitive skill execution failures, and (4) skill effect verification. When a plan fails for any of these reasons, the Recovery Manager is consulted to determine if recovery should be attempted.

The Recovery Manager keeps a history of failures and attempted recoveries, as well as termination criteria based on how many times the agent has attempted recovery from a particular failure. This is necessary to avoid endless recovery attempts when the system fails to make progress towards a goal. If the agent has not reached the termination criteria, the Recovery Manager uses the Recovery Policy Database (RPDB) to select an applicable Recovery Policy based on each policy's usage constraints and the available failure information. The constraints defining when a particular Recovery Policy is applicable can include (1) the failed goal, (2) the failed skill during execution, (3) the semantic failure reason(s), and (4) the failure status (i.e., PLAN, PRECONDITON, EFFECT, EXECUTION). The policy constraints allow for a policy to be widely applicable (e.g., do this for all precondition failures), or highly customized (e.g., only do this if skill S fails for reason B during execution of goal G).

If a valid Recovery Policy cannot be found, goal execution terminates and the goal is marked as a failure. If, however, a Recovery Policy is found, the policy is executed in the same way as a top-level goal. Recovery policies are specified in the same scripting language used to define non-recovery skills, which critically allows for policies to execute skills and submit sub-goals that make use of planning and recursive failure recovery. Note that since recovery policies are represented as scripts, they can be learned through interactions the same way as tasks.

Once a policy is successfully executed, control is returned to the failed parent goal, where GSM generates a new plan for the goal before resuming execution. Replanning is necessary to account for any changes that the Recovery Policy might have caused, either in the world (e.g., by executing skills), or internal state (e.g., by modifying planning operators). If a policy fails during execution (and no recovery policy can handle the failure), an attempt is still made to replan for the failed goal as a last effort to achieve the goal state before giving up.[5] If no progress was made during the failed recovery policy, it is likely that the goal will continue to fail in the same way, but the Recovery Manager's termination criteria will prevent endless recovery attempts.

At any time during plan execution and failure recovery, the system will generate performance assessments of plans that it can communicate to human teammates, e.g., about the likelihood of plan success and the expected duration. This is particularly important for problem solving dialogues about how to address execution failures to support human teammates in finding acceptable solutions (e.g., when a previous plan failed or is no longer applicable because of changes in goals).

4 DEMONSTRATION

For the multi-robot architecture demonstration, we selected an indoor **medical assembly and delivery task** (common in care settings) that involves three human team members—a **task teacher**, a **task instructor** requiring a medical kit, and a medical **supplies worker**—and three heterogeneous robots—the **Fetch** assuming the role of pharmacy kitting robot, the **Temi** assuming the role of medical delivery robot, and the **Spot** being available as a helper.

The task performance shows (1) the initial learning of the assembly task through interactive teaching dialogues, (2) the immediate instruction to perform the task and the detection of a missing item for assembly, (3) the notification of the human instructor of the problem and the proactive generation of an alternative plan together with an automatic plan assessment, and finally (4) the execution of the alternative plan involving a helper robot. We will discuss all

[5] Note that there are cases where even replanning does not help, e.g., if an object is too slippery for the gripper to pick up, repeated pick-up attempts are not helpful, or if the robot needs to go to a location but cannot access it. In those cases, asking humans for help might be another recovery policy.

Figure 2: A scene from the demonstration.

four phases below in detail, showing how the four central architectural components—the DM, GSM, SLIK, and MTP—work together to enable the problem solving dialogue interactions. A video of the demonstration can be found at: https://youtu.be/_PJIkCyULjQ. The source code used to execute this demonstration can be found at: https://github.com/mscheutz/diarc.

4.1 Task Setting

We used a previous architecture framework for all implementations [17] which supports and integrates the three robots with their specific software and operating systems through architectural perceptual and motor control components.

This instance of the architecture is deployed on a collection of devices all connected to a local network and uses a middleware [11] to communicate over the network. The core architecture (DM, SLIK, GSM, MTP) and NLU & NLG, run on desktop PC running Ubuntu 18; the Fetch robot runs ROS Melodic on Ubuntu 18 [24] and utilizes MoveIt [1] for motion planing; the Spot Robot uses ROS Noetic on Ubuntu 20 and the Boston Dynamics Spot API [6]; and the Temi uses the RoboTemi SDK [7] running on Android 11. The human team members are using Android phones running Android version 11 or greater which contain ASR, TTS, and GUI components.

We utilize the *Metric-FF planner* [10], a forward-chaining heuristic state space planner as the architecture requires much of the functionality supported by Metric-FF, including the use of numeric fluents: extensions to the planning domain representation that allow for numeric operations including counting and (in)equality operations in state specification and operator effects. We also utilize the methods described in [7] for the assessment of expected plan success and duration for hypothetical plans generated during fault recovery which, given a plan, will generate the likelihood of the plan execution succeeding by sampling from success and failure distributions of stored plan actions, using similar sampling techniques to determine the expected plan execution duration.

The task environment consists of several indoor rooms connected through doorways. The medical kit assembly station in the "pharmacy" is part of a large room based on the 2019 ICRA FetchIt! task [4, 8].

[6]https://github.com/heuristicus/spot_ros
[7]https://github.com/robotemi/sdk

4.2 Interactive Task Teaching

The teaching starts with the human teacher telling the robot "Define new kit 'medkit'" which prompts the retrieval of a "kit teaching" Interaction Template. This is a domain specific Interaction Template that was designed to reduce the burden of knowledge on the human teacher when teaching a new type of kit. Below is the algorithm used by the "kit teaching" Interaction Template.

Algorithm 2 Kit Teaching Interaction Template

1: given *label* from *human*
2: Predicate *container* = askQuestion(*human*, "what container does it use?" , $object(X)$)
3: **if** notValid(*container*) **then**
4: askQuestion(*human*, "could you show me what *container* looks like?" , $obectDefintion(X)$)
5: **end if**
6: **if** notKnown(*a*) **then**
7: askQuestion(*human*, "where is there a *container*?" , $locationOf(X, Y)$)
8: **end if**
9: List<Predicate> contents
10: Predicate *content* = askQuestion(*human*,"what does a *label* contain?" , $object(X)$)
11: **while** $content \neq nothing$ **do**
12: **if** notValid(p) **then**
13: askQuestion(*human*, "could you show me what *content* looks like?" , $obectDefintion(X)$)
14: **end if**
15: **if** notKnown(a) **then**
16: askQuestion(*human*, "where is there a *content*?" , $locationOf(X, Y)$)
17: **end if** *content* = = askQuestion(*human*, "does a Medkit contain anything else?" , $object(X)$)
18: **end while**
19: assertKitDefintion(*label*,*container*, *content*) ▷ specialized method to update SLIK, GSM, and NLU for new kit based goal state

After executing the above Interaction Template and the associated dialogue, the newly learned knowledge is stored in SLIK and is ready for execution. When a task instructor asks the robot to "deliver a medkit to alpha", the Fetch robot can get to work immediately.

4.3 Task Instruction

The task instruction prompts the DM to pass the goal semantics for the medkit assembly goal to the GSM:

```
(and (fluent-equals(amount(physobj0area:area,painkiller:property),2),
    (fluent-equals(amount(physobj0area:area,bandagebox:physobj),1))))
```

which, in turn, calls the MTP to generate a plan for the Fetch to assembly a medkit as previously taught:

1: (goto fetch mblocation6 mblocation4 room1 room1)
2: (perceiveobject fetch physobj3 mblocation4 tablee bandagebox)
3: (goto fetch mblocation4 mblocation1 room1 room1)
4: (perceiveobject fetch physobj1 mblocation1 tableb painkiller)
5: (perceiveobject fetch physobj2 mblocation1 tableb painkiller)
6: (pickup fetch physobj2 painkiller tableb mblocation1)
7: (goto fetch mblocation1 mblocation0 room1 room1)
8: (perceiveobject fetch physobj0 mblocation0 tablea medicalcaddy physobj0area)
9: (putin fetch physobj2 painkiller physobj0 physobj0area mblocation0)
10: (goto fetch mblocation0 mblocation4 room1 room1)
11: (pickup fetch physobj3 bandagebox tablee mblocation4)
12: (goto fetch mblocation4 mblocation0 room1 room1)
13: (putin fetch physobj3 bandagebox physobj0 physobj0area mblocation0)
14: (goto fetch mblocation0 mblocation1 room1 room1)
15: (pickup fetch physobj1 painkiller tableb mblocation1)
16: (goto fetch mblocation1 mblocation0 room1 room1)
17: (putin fetch physobj1 painkiller physobj0 physobj0area mblocation0)

However, the plan fails at step 2 of the execution when attempting to perceive a bandage box at table E (as there are no bandage boxes on table E). Since the robot cannot detect any object at this location, the action fails, and a failure justification of

```
not(objectAt(physobj3:physobj,tableE:area))
```

is generated. The DM uses the failure justification to inform the human task instructor right away: "I failed execution because the bandage box object is not at Table E" (this utterance also uses the fact that the human task instructor did not know that there was no bandage box on Table E). The action failure then also triggers the failure recovery mechanism which, based on the recovery policies, inspects the failure justification to attempt to create a new plan to fix the failure, generating the new state-based goal

```
objectAt(physobj3:physobj,tableE:area)
```

that is then submitted to the GSM which gets the recovery plan for it from the MTP:

1: (goto spot spotlocation0 spotlocation5 room1 room3)
2: (receiveitem spot physobj3 bandagebox pharmacy spotlocation5)
3: (goto spot spotlocation5 spotlocation1 room3 room1)
4: (putdownspot spot physobj3 bandagebox tableg spotlocation1)
5: (goto fetch mblocation4 mblocation6 room1 room1)
6: (pickup fetch physobj3 bandagebox tableg mblocation6)
7: (goto fetch mblocation6 mblocation4 room1 room1)
8: (putdown fetch physobj3 bandagebox tablee mblocation4)

The human task instructor is then informed of the recovery plan: "I have found a plan to recover". The GSM also uses its performance assessment projection to determine the expected run-time: "It will take about two minutes and nine seconds. Do you want me to execute it?" At this point, the human task instructor wants to know the details of the recovery plan: "Describe your plan", which prompts the robot to narrate the plan. Upon human approval—"execute your plan"—the robot embarks on getting bandage boxes onto Table E, so that it can subsequently resume its plan. Because it needs a mobile platform with a gripper to pick up and transport a bandage box, it tasks the Spot robot to travel to the pharmacist's office where it knows it can ask the pharmacist for a bandage box: "Could I please have a bandage box?" Upon receiving the box, the Fetch robot can finish its plan and the expected world state of a bandage box at table E has been restored. The system can now re-plan for the original goal, adjusted for the new state of the world:

```
(and (fluent-equals(amount(physobj0area:area,painkiller:property),2),
    (fluent-equals(amount(physobj0area:area,bandagebox:property),1))))
```

which results in the following adjusted plan:

1: (goto fetch mblocation4 mblocation1 room1 room1)
2: (perceiveobject fetch physobj1 mblocation1 tableb painkiller)
3: (perceiveobject fetch physobj2 mblocation1 tableb painkiller)
4: (pickup fetch physobj2 painkiller tableb mblocation1)
5: (goto fetch mblocation1 mblocation0 room1 room1)
6: (perceiveobject fetch physobj0 mblocation0 tablea medicalcaddy physobj0area)
7: (putin fetch physobj2 painkiller physobj0 physobj0area mblocation0)
8: (goto fetch mblocation0 mblocation4 room1 room1)
9: (pickup fetch physobj3 bandagebox tablee mblocation4)
10: (goto fetch mblocation4 mblocation0 room1 room1)
11: (putin fetch physobj3 bandagebox physobj0 physobj0area mblocation0)
12: (goto fetch mblocation0 mblocation1 room1 room1)
13: (pickup fetch physobj1 painkiller tableb mblocation1)
14: (goto fetch mblocation1 mblocation0 room1 room1)
15: (putin fetch physobj1 painkiller physobj0 physobj0area mblocation0)

Once the caddy is packed with all required items, a new goal is submitted for delivery of the caddy to its final delivery location, and a plan is generated. This plan includes both the placement of the caddy on the Temi robot by the Fetch robot, and also makes use of the Spot robot to open the closed door to the delivery location alpha as the system knows that the door is closed and that alpha is not accessible to the Temi (through introspection on the robot's capabilities) with a closed door (as the Temi cannot open a door). The final delivery goal thus is

```
delivered(physobj0:physobj,temilocation0)
```

and the corresponding plan is

1: (goto spot spotlocation0 spotlocation2 room1 room1)
2: (pickup fetch physobj0 medicalcaddy tablea mblocation0)
3: (goto fetch mblocation0 mblocation7 room1 room1)
4: (opendoor spot spotlocation2 spotlocation3 room1 room2)
5: (goto temi temilocation1 temilocation2 room1 room1)
6: (handover fetch temi mblocation7 temilocation2 physobj0)
7: (goto temi temilocation2 temilocation0 room1 room2)
8: (deliverkit temi physobj0 temilocation0)

The plan is executed and the medical caddy is successfully delivered to the task instructor in Alpha.

5 DISCUSSION AND CONCLUSION

The demonstration showed that the proposed multi-robot architecture is effective in interacting with human teammates in task-based natural language dialogues: learning new tasks on the fly, informing them of unexpected events, and proposing proactively developed alternative plans to mitigate their effects. The demonstration also showed that it can coordinate multiple heterogeneous robots with different hardware and software capabilities in a task that can serve as a proxy for many similar tasks (e.g., shopping in stores, kitting in factories, etc.). By treating the multi-robot system as a "super agent" with different bodies, the multi-robot architecture provided a "single agent perspective" that is not only helpful for humans as they do not have to keep track of the individual robots in the team, but also for naturally implementing a shared mental model and for reducing a multi-robot planning and coordination problem to a single agent planning and execution task. This is also very practical for fault recovery, as demonstrated by the Spot robot getting the missing bandage item, and for supporting other robot actions, as demonstrated by the Spot opening the door for the Temi.

While the demonstration highlighted important features of the architecture, it by no means touched on all of them. For example, it did not show multi-robot task instructions, online task modifications, or tracking and synchronizing human teammates' belief states, all of which are important capabilities of the architecture supporting mixed-initiative teams. Yet, we believe that it showed the effectiveness of the proposed framework for the development of future robotic teammates. For one, the framework can be easily extended by additional architectural components as it is intrinsically parallel and distributed, and can be used on any number of heterogeneous robots that will immediately be able to work in concert on the team task. Finally, being fully instructible, the framework can be employed in many team settings as long as the involved robots have the basic actions needed to perform their tasks.

ACKNOWLEDGMENTS

This work was supported by grants from AFOSR (FA9550-18-1-0465) and ONR (N00014-22-1-2206 and N00014-24-1-2024).

REFERENCES

[1] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. 2014. Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785* (2014).

[2] Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. 2023. Task and Motion Planning with Large Language Models for Object Rearrangement. *arXiv preprint arXiv:2303.06247v4* (2023).

[3] 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.

[4] Tyler Frasca, Zhao Han, Jordan Allspaw, Holly Yanco, and Matthias Scheutz. 2020. Going cognitive: A demonstration of the utility of task-general cognitive architectures for adaptive robotic task performance. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 8110–8116.

[5] Tyler Frasca, Bradley Oosterveld, Meia Chita-Tegmark, and Matthias Scheutz. 2021. Enabling fast instruction-based modification of learned robot skills. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 6075–6083.

[6] Tyler Frasca, Bradley Oosterveld, Evan Krause, and Matthias Scheutz. 2018. One-Shot Interaction Learning from Natural Language Instruction and Demonstration. *Advances in Cognitive Systems* 6 (2018), 159–176.

[7] Tyler Frasca and Matthias Scheutz. 2022. A Framework for Robot Self-Assessment of Expected Task Performance. *IEEE Robotics and Automation Letters* 7, 4 (2022), 12523–12530. https://doi.org/10.1109/LRA.2022.3219024

[8] Zhao Han, Jordan Allspaw, Gregory LeMasurier, Jenna Parrillo, Daniel Giger, S Reza Ahmadzadeh, and Holly A Yanco. 2020. Towards mobile multi-task manipulation in a confined and integrated environment with irregular objects. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 11025–11031.

[9] Jamison Heard and Julian Fortune andJulie A. Adams. 2020. SAHRTA: A Supervisory-Based Adaptive Human-Robot Teaming Architecture. In *IEEE Conference on Cognitive and Computational Aspects of Situation Management (CogSIMA)*.

[10] Jörg Hoffmann. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *Journal of Artificial Intelligence Research* 20 (2003), 291–341.

[11] James Kramer and Matthias Scheutz. 2007. Robotic Development Environments for Autonomous Mobile Robots: A Survey. *Autonomous Robots* 22, 2 (2007), 101–132.

[12] Evan Krause, Michael Zillich, Thomas Williams, and Matthias Scheutz. 2014. Learning to recognize novel objects in one shot through human-robot interactions in natural language dialogues. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 28.

[13] John E Laird. 2019. *The Soar cognitive architecture*. MIT press.

[14] Microsoft Open SOurce. [n.d.]. *Yet Another Revers Proxy*. https://github.com/microsoft/reverse-proxy

[15] Matthias Scheutz, Scott DeLoach, and Julie Adams. 2017. A Framework for developing and using shared mental models in human-agent teams. *Journal of Cognitive Engineering and Decision Making* 11, 3 (2017), 203–224. https://doi.org/10.1177/1555343416682891

[16] Matthias Scheutz, Evan Krause, Brad Oosterveld, Tyler Frasca, and Robert Platt. 2017. Spoken Instruction-Based One-Shot Object and Action Learning in a Cognitive Robotic Architecture. In *Proceedings of the 16th International Conference on Autoomous Agents and Multiagent Systems*.

[17] Matthias Scheutz, Thomas Williams, Evan Krause, Brley Oosterveld, Vasanth Sarathy, and Tyler Frasca. 2019. An Overview of the Distributed Integrated Affect and Reflection Cognitive DIARC Architecture. In *Cognitive Architectures*, Maria Isabel Aldinhas Ferreira, Joao Silva Sequeira, and Rodrigo Ventura (Eds.). Springer International Publishing.

[18] Stanford Artificial Intelligence Laboratory et al. [n.d.]. *Robotic Operating System*. https://www.ros.org

[19] Aaquib Tabrez, Matthew B. Luebbers, and Bradley Hayes. 2020. A Survey of Mental Modeling Techniques in Human–Robot Teaming. *Current Robotics Reports* 1 (2020), 259 – 267.

[20] Robert Touchton, Daniel Kent, Tom Galluzzo, Carl D Crane III, David G Armstrong II, Nick Flann, Jeff Wit, and Phil Adsit. 2005. Planning and modeling extensions to the Joint Architecture for Unmanned Systems (JAUS) for application to unmanned ground vehicles. In *Unmanned Ground Vehicle Technology VII*, Vol. 5804. SPIE, 146–155.

[21] J Gregory Trafton, Laura M Hiatt, Anthony M Harrison, Franklin P Tamborello, Sangeet S Khemlani, and Alan C Schultz. 2013. Act-r/e: An embodied cognitive architecture for human-robot interaction. *Journal of Human-Robot Interaction* 2, 1 (2013), 30–55.

[22] Vaibhav V. Unhelkar, Shen Li, and Julie A. Shah. 2020. Decision-Making for Bidirectional Communication in Sequential Human-Robot Collaborative Tasks. In *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*. Association for Computing Machinery, New York, NY, USA, 329–341.

[23] Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. 2023. *ChatGPT for Robotics: Design Principles and Model Abilities*. Technical Report MSR-TR-2023-8. Microsoft. https://www.microsoft.com/en-us/research/publication/chatgpt-for-robotics-design-principles-and-model-abilities/

[24] Melonee Wise, Michael Ferguson, Derek King, Eric Diehr, and David Dymesich. 2016. Fetch and freight: Standard platforms for service robot applications. In *Workshop on autonomous mobile service robots*. 1–6.