Communicating, Interpreting, and Executing High-Level Instructions for Human-Robot Interaction

Nishant Trivedi
Pat Langley
Computer Science and Engineering
Arizona State University, Tempe, AZ 85287

Paul Schermerhorn
Matthias Scheutz
Cognitive Science Program
Indiana University, Bloomington, IN 47406

Abstract

In this paper, we address the problem of communicating, interpreting, and executing complex yet abstract instructions to a robot team member. This requires specifying the tasks in an unambiguous manner, translating them into operational procedures, and carrying out those procedures in a persistent yet reactive manner. We report our response to these issues, after which we demonstrate their combined use in controlling a mobile robot in a multi-room office setting on tasks similar to those in search-and-rescue operations. We conclude by discussing related research and suggesting directions for future work.

Introduction

Advances in robotics hardware and software have taken robots to the point where they can play an important role in extended activities like surveillance, exploration, and rescue operations. Yet an important remaining bottleneck is the need for robots to interact efficiently with human team members. Effective human-robot interaction requires a middle ground between fully autonomous agents that operate entirely on their own and detailed but tedious teleoperation by human controllers. Many mixed-initiative settings would benefit from human-robot coordination that operates at the same level as occurs in human teams.

A key characteristic of human teams is that they coordinate behavior at the level of natural language. This lets the members communicate their beliefs, goals, and intentions in terms abstract enough to be conveyed rapidly, yet unambiguous enough to transform them into operational activity. We believe that similar levels of communication support team coordination whether members are co-located or remote, and whether they adopt a flat or hierarchical command structure. We maintain that reproducing the ability to coordinate such joint activity at this abstract level will make human-robot teams as effective as ones that are composed entirely of humans.

In this paper, we report an innovative approach to supporting human-robot interaction in this manner. The next section describes the search-and-rescue scenario that has driven our research, along with the physical setting and robotic platform we have used to pursue it. After this, we present three facets of our approach to enabling joint human-robot activity: stating and representing complex tasks, converting them into operational procedures, and executing those procedures in a teleo-reactive manner. Next we report successful robot runs on tasks communicated at an abstract level. In closing, we review work on related topics and outline directions for additional research.

Coordination for Search and Rescue

Consider a situation in which a human-robot team must traverse a partially known environment to find objects. For example, the members might need to jointly explore a damaged building in search of injured people who need medical attention or evacuation. The human may have access to a map, but it could be unreliable and in any case it does not include locations for the objects being sought.

Suppose further that the human team member remains outside for reasons of safety, since portions of the building may be unstable. However, he can see the output of a video camera mounted on the robot and he can communicate with the robot through spoken or written language. For instance, based on available video and his estimate of the robot's location, the human might give the instruction:

> Go down the hallway until you find an open door and then go through it.

The robot would interpret this abstract command, transform it into operational procedures, and report on its progress and completion. The human would then give another command, the robot would continue on this new mission, and so forth. In this manner, the team could jointly explore the environment, finding injured people that could be treated or evacuated in a more targeted operation.

We have developed a laboratory setting that captures many important features of such scenarios: a building with several rooms connected by a long hallway. We have placed boxes of various colors throughout the en-

vironment that serve as surrogates for injured people. We have control of which doors are open, lighting conditions, and other factors that make the joint exploration task more or less challenging.

We have used a Pioneer P3AT platform for our robotics development and evaluation. The robot is equipped with a number of sensors relevant to the task, including a SICK laser range finder for detecting obstacles and doorways, bumper sensors and emergency stop buttons, and two video cameras, one visual-light camera (for use in daylight) and one infrared camera (for use at night), both for detecting objects and for streaming video to the operator. The robot also has an analog wireless audio link that lets it communicate in natural language with the remote human operator. All control software runs on a Dell quadcore Pentium laptop mounted on the robot, so it can range throughout the test environment.

Supporting human-robot interaction in this type of scenario poses a number of technical challenges. First, we must specify and represent the complex tasks that the human team member might ask his robot collaborator to carry out. Second, we must translate these instructions into procedures that an intelligent agent can interpret. Third, we must enable execution of these procedures in a flexible yet task-directed way within an integrated robotic system. In the sections that follow we describe our responses to these technical issues, which we claim are sufficient to enable high-level control of robots in a variety of task-oriented settings. Later we will support these claims with demonstrations of their integrated use in directing behavior of the Pioneer robot described above.

Specifying Complex Tasks

The first step in developing any intelligent system involves specifying the content over which it will operate. We can conveniently divide this content into the system's beliefs about its situation, the tasks it desires to achieve, and the knowledge it uses to achieve them. We discuss the perceptual and inference mechanisms that provide our agent's beliefs in a later section, once we describe the knowledge that supports them. Here we focus on the more basic research question of how to encode the tasks themselves.

Traditional task representations, like those used in the AI planning literature (Ghallab, Nau, & Traverso, 2004), focus on goals that describe static or instantaneous features of the state, such as location of the robot or having an object placed in a box. Clearly, the tasks from our target scenario are more complex in that they describe sequences of states that should occur, as well as the conditions under which they should take place. Yet these descriptions are not fully specified; they are not themselves executable robot programs in that they retain the high-level, abstract character of language.

In response, we have developed a command language for specifying complex task-oriented behavior. The language combines a small set of command predicates with domain concepts and actions to let one describe activities in terms of application conditions, termination criteria, and orderings on subactivities. Elements in the language are composable, letting one specify complex commands, and thus a wide range of agent behaviors, by combining simpler structures. Our goal is not to reproduce the full expressive power of natural language, but rather to provide a surrogate that serves the more limited functions needed to express high-level commands, just as parents use restricted language when communicating with children.

For this reason, the command language uses a constrained English syntax to describe activities. For example, the statement

> Go down the hallway until you find an open door and then go through it.

would be rephrased as

> *First Until you find open door D*
> *Holds you go down the hallway*
> *Next you go through D.*

This command demonstrates the use of two distinct paired sets of command predicates.

One pair, *Until/Holds*, indicates an activity that the agent should carry out until its halting criteria are met.[1] The other, *First/Next*, specifies that the agent should first pursue one activity (the *Until/Holds* subcommand) and then carry out another (going through the door). In more complex tasks, this might involve three or more ordered activities. The expression *open door* refers to a known domain concept, while *go down* and *go through* refer to known domain activities. The words *find*, *the*, *are*, and *should* are 'stop' words included for readability, while the remaining terms, *D* and *you*, denote pattern-match variables.

The constrained English syntax maps directly onto an internal notation that uses list structures. Here the corresponding internal representation would be

```
(before (until (open-door ?d)
               (go-down-hallway ?you))
        (go-through ?you ?d))
```

In this structure, the *before* predicate denotes an ordering on two or more subcommands, while *until* specifies conditions (in this case, only one) for terminating a subcommand. This notation differs from the English syntax in that it uses parentheses as delimiters rather than pairs of command words. In addition, it omits stop words and marks variables with question marks.

The internal notation also clarifies the embedded character of commands, with the *until* clause occurring within the *before* statement and with the *go-down-hallway* action occurring within the *until* clause. Both *before* and *until* take two or more arguments. For *before*, each entry is a subcommand that the agent should

[1] The *Holds* here indicates that the preceding description should be true, although the phrasing is slightly awkward.

carry out in order. For *until*, the final entry is either another command or a domain action, whereas the earlier arguments specify a set of conditions that must match consistently (with shared variables denoting the same values) for the final statement to apply.

Another function of commands is to allow conditional statements. For example, consider the instruction

First If D is a door in front of you
Then you go through D
Next you turn left
Next you go down the hallway.

Here use of *First* and *Next* indicates that the agent should execute the three subcommands in the specified order, but the *If/Then* expression indicates that the first subcommand is conditional on the agent detecting a door in front of it.

We can also specify this embedded conditional statement in our internal syntax; in this case, we would write

```
(before (if (door-in-front ?you ?d)
            (go-through ?you ?d))
        (turn-left ?you)
        (go-down-hallway ?you))
```

Here the *if* predicate indicates that its final argument should be carried out only if the preceding condition is satisfied. More generally, such statements may include multiple conditions, all of which must match to execute the final subcommand. As before, *door-in-front* is a domain concept, while *go-through*, *turn-left*, and *go-down-hallway* are activities.

The above example is slightly problematic, since it does not specify what the agent should do if there is no door in front of it. This clarifies the need to express contingent courses of action. For instance, consider the command

If B is a box
Then Either If B is blue, Then you turn right
Or If B is not blue, Then you turn left.

Here the *Either* predicate indicates the start of the first alternative, while *Or* marks its end and the start of the second option. Although not shown in this example, the syntax allows three or more alternatives.

Again, we can state an equivalent command in the internal syntax, with the English instructions becoming

```
(if (box ?b)
    (or (if (blue ?b) (turn-right ?you))
        (if (not (blue ?b)) (turn-left ?b))))
```

In this case, the *or* predicate has two arguments, but it can take anny number of subcommands or expressions with domain actions like *turn-right* and *turn-left*.

We have also implemented a routine that transforms English command expressions into the internal representation already described. This process involves detecting command predicates that mark the beginning and end of subcommands to determine the hierarchical structure of the instruction. The software also removes stop words, detects word sequences that correspond to domain predicates and actions, and replaces the former with the latter. The routine transforms any remaining terms into pattern-match variables. The result is a list structure like those we have already seen.

The above statements are imperative in that they indicate the agent should carry out the specified instructions. However, both the English and internal languages also incorporate a nonimperative *define* predicate that lets the system encapsulate procedures for later reuse. This function takes as arguments the activity's name, its arguments, and its specification. The ability to define such named procedures has two benefits. First, it supports the specification of recursive activities that otherwise could not be stated. Second, it enables the specification of generalized behaviors, such as moving toward or away from an entity, followed by repeated use of this command with different objects as arguments. As a result, this capacity reduces considerably the effort needed to produce complex behavior.

Some readers may question whether stating instructions in constrained English provides advantages over writing programs in a traditional procedural language. However, note that, although our command language can specify desired behavior in great detail, it also allows very abstract instructions that refer to known procedures. Moreover, commands may be nondeterministic and require the agent to select among different expansions. Taken together, these features support the ability to accept and interpret high-level specifications for complex activities that the agent should carry out, supporting *taskability* in the sense that Langley, Laird, and Rogers (2009) describe.

Our first claim is that *the command language described above suffices to specify a broad range of complex tasks that involve extended activities*. More specifically, the four pairs of command predicates, when combined with domain concepts and actions, are enough to describe a reasonable subset of behaviors that arise in tasks like search-and-rescue operations. We will return to this claim when we report experimental results with the Pioneer robot. We will not argue that our notation is necessary, since other formalisms may offer equivalent coverage. We also believe that human users will find the constrained English syntax more usable than traditional languages for robot programming, but we are not yet ready for studies with human subjects, so for now this must remain a conjecture.

Translating Tasks into Procedures

Although statements in the internal syntax are more formal than English instructions, they would be useless without an interpreter that executes them in the environment or converts them into executable procedures for an existing interpreter. We have chosen the second alternative of transforming internal command statements into knowledge structures for ICARUS (Langley, Choi, & Rogers, 2009), an agent architecture that supports teleoreactive execution. We will delay discus-

sion of this execution mechanism until the next section. Here we focus on the structures it utilizes and our methods for transforming commands into them.

ICARUS distinguishes between two types of knowledge. The first – *concepts* – specifies classes of situations that can arise in the agent's environment. Each conceptual predicate is associated with one or more Horn clauses that define it as logical combinations of lower-level concepts and percepts, the latter describing objects in the environment. This imposes a hierarchical organization on concept memory, with higher levels denoting more abstract relations. Our command language assumes relevant domain concepts have been defined, so that instructions can refer to them without ambiguity.

The second form of knowledge in ICARUS – *skills* – describes activities that the agent can carry out to alter its environment. Each skill predicate is associated with one or more decomposition rules that break into down into an ordered set of component skills or actions, the latter describing primitive activities the agent can execute directly. Each skill clause also includes conditions that determine when it can apply and a set of effects that describe how it alters the environment. The architecture organizes skills in a hierarchy, with higher-level structures encoding more abstract procedures that typically cover longer periods.

We have developed a translator that turns commands in the internal format described earlier into a set of ICARUS skills for the task. The mapping is reasonably straightforward, with each list or sublist in the internal command representation leading to one skill clause. The translation algorithm operates recursively, stepping downward through the embedded lists of the internal notation and creating skills as it returns upward. For this reason, it creates lower-level skills first, then ones that refer to them, and finally a single top-level skill that corresponds to the entire command.

We can clarify the translator's operation with an example. As we have seen, the English instruction

First Until you find open door D
Holds you go down the hallway
Next you go through D.

is converted to the internal notation

```
(before (until (open-door ?d)
               (go-down-hallway ?you))
        (go-through ?you ?d))
```

In this case, the translator produces two ICARUS skills

```
((skill-2 ?d ?you)
 :conditions ((robot ?you))
 :subskills  ((skill-1 ?you ?d)
              (go-through ?you ?d)))
((skill-1 ?you ?d)
 :subskills  ((go-down-hallway ?you))
 :effects    ((open-door ?you ?d)))
```

organized in a two-level hierarchy, with *skill-2* referring to *skill-1*. Because the first structure corresponds to an *If/Then* command, it includes conditions but no effects. In contrast, because the second came from an *Until/Holds* statement, it contains effects but no conditions. The predicates *robot* and *open-door* are predefined concepts, whereas *go-through* and *go-down-hallway* are predefined primitive skills.

We have noted that the mechanism for converting English commands to the internal encoding assumes that relevant domain concepts and actions have been defined, and our translator relies on the same assumption. However, we should also mention that, if a command refers to skills already defined, the system can incorporate them into new structures. This lets a human instructor state high-level commands in terms of simpler specifications he has given earlier.

Our second claim is that *the translation mechanism just described supports the conversion of commands stated in our constrained English syntax into a set of equivalent* ICARUS *skills*. More specifically, the four pairs of command predicates map directly onto different aspects of the ICARUS skill syntax, and that embedding these commands leads naturally to the hierarchical organization of skills that plays a central role in the architecture. The internal details of the translation mechanism matter little, and other implementations are certainly possible, but we hold that the mapping process supports the same broad range of physical activities as does the command language. We will revisit this claim when we present the results of robot demonstration runs in a later section.

Executing Translated Commands

Once it has transformed constrained English instructions into a set of hierarchical skills, the command interpreter invokes ICARUS to carry out the specified activities in the current situation. To clarify the details of this process, we should briefly review how the architecture utilizes its concepts and skills to produce complex activity over time.

ICARUS operates in discrete cycles that involve calling on two main modules.[2] The first carries out a process of conceptual inference. This matches the antecedents of conceptual clauses to percepts (descriptions of objects visible to the agent) to produce beliefs (instances of defined concepts) that the module adds to a belief memory. These trigger matches of higher-level conceptual clauses that produce more abstract inferences. This process continues in a bottom-up manner until the architecture has generated all beliefs implied by its concepts and percepts. In this manner, the agent maintains an up-to-date set of beliefs about its environment.

The second ICARUS module is responsible for skill execution. This examines its top-level intentions (instances of known skills) and selects the highest-priority candidate that it has not yet completed. The architecture retrieves skill clauses with heads that match this

[2]The full architecture includes additional modules for problem solving and skill acquisition, but we will not deal with them in this paper.

intention and selects one with matched conditions that make it applicable. If this skill clause is nonprimitive, then ICARUS creates a new intention for its first subskill and continues to the next cycle. This continues until it reaches a primitive skill, which it executes in the environment. Once this has completed, it moves on to the next subskill, expands it if necessary, and continues until none remain, in which case it pops back to the parent intention. If all goes well, ICARUS eventually completes the top-level intention, but unexpected environmental changes cause it to adapt reactively to the new situation, accessing other parts of the skill hierarchy.[3]

The notion of an intention plays a central role in carrying out the English commands provided to the robot. Once the translator has produced a corresponding set of hierarchical skills, it also generates a top-level intention that is an instance of the highest-level skill. The command interpreter then calls on ICARUS to carry out this intention, which brings the architecture's machinery into play. Upon completion, the system awaits further instructions, then translates and executes them upon arrival.

In previous work, we have provided ICARUS with domain actions that it can execute directly in simulated environments, but this approach is not available when dealing with physical robots. To give it robust means for controlling a robot, we have integrated it with DIARC, a distributed robot control architecture. This supports lower-level activities such as multi-modal perceptual processing using color and shape detection, object detection and tracking using SIFT features, face detection and person tracking, gesture and pointing behavior detection, navigation, and overall behavior coordination (Scheutz & Andronache, 2004). DIARC is implemented in the distributed multi-agent robotic development infrastructure ADE (Scheutz, 2006), which allows for straightforward extension and integration of new software components by "wrapping" them in "ADE agents", in this case the ICARUS architecture.

DIARC provides a convenient way for executing behaviors on the robot by virtue of a *goal manager*, which is a priority-based scheduler for actions scripts that run in parallel in their own threads. Whenever a script is ready for execution, the *goal manager* instantiates an *action interpreter*, whose job it is to execute all atomic and complex actions in the script. Atomic actions are typically either requests for (raw) sensory information from the robot's sensors (e.g., distance data from the laser range finder) or motor commands that are sent to various robot effectors (e.g., rotational and translational velocities sent to the wheels of the base). Complex actions are (possibly conditional) sequences of atomic actions. Each instantiated action script has a priority associated with it, which can be based on various factors, typically including the utility of accomplishing the goal associated with the script and the time available for the script to finish. DIARC uses these priority values for *priority-based behavior arbitration* (Scheutz & Andronache, 2004), in which the goal manager continually recalculates the priority of each running script to let those with higher priority access resources (e.g., effectors) when there is contention.

Because DIARC interfaces directly with the robot's sensors, it is responsible for detecting entities like boxes, doors, and hallways, then depositing descriptions of relevant objects into ICARUS' perceptual buffer to drive the inference process. This occurs automatically in some situations, such as when an obstacle appears in the robot's path, letting them serve as interrupts to ongoing activities. However, ICARUS can also deliberately focus attention on classes of objects like doors or boxes by invoking an *attend* action, which causes DIARC to deposit descriptions of any visible instances of that class into the perceptual buffer. ICARUS must explicitly call this action on each cycle for it to have an effect; otherwise, DIARC will not provide it with information about objects when they are detected, which can cause the former to omit key inferences about the environment.

Another aspect of the ICARUS-DIARC integration concerns the selection and execution of hierarchical skills. This is straightforward when ICARUS first attempts to find an applicable path through the hierarchy. The process works as described earlier except that, upon reaching a primitive skill, the architecture passes the associated actions to its companion system as DIARC goals, which invokes modules that attempt to achieve them. However, because both components have reactive interpreters, the integrated system also requires information to flow upward from the former to the latter about the status of these actions/goals, which may take many cycles to complete. For this reason, each DIARC goal is marked as *ongoing*, *succeeded*, or *failed*. In the first case, ICARUS does not invoke any more actions, since it assumes DIARC is making progress on the task. In the latter two cases, ICARUS abandons the previously selected skill path, even if otherwise applicable, and attempts to find another, since it knows DIARC has either completed it successfully or failed to do so. This feedback lets ICARUS retain high-level oversight of the robot's behavior while taking advantage of DIARC's efforts at lower levels.

Our final claim is that *the integration of* ICARUS *and* DIARC *just described is sufficient to execute the same broad range of tasks that the command language and the command translator support*. More specifically, the integrated system enables teleoreactive control of robots which carry out extended activities that involve complex combinations of conditions and sequencing. In the next section, we report demonstrations of this ability with a physical robot in realistic scenarios. Again, we will not argue that our approach is the only response to this challenge, but we believe it is a viable one that makes a clear contribution to the literatures on human-robot interaction and cognitive systems.

[3]This approach to hierarchical yet reactive control incorporates ideas from both Ingrand, Georgeff, and Rao's (1992) PRS and Nilsson's (1994) teleoreactive programs.

Experimental Demonstrations

To ensure that our approach to human-robot interaction has the intended capabilities, we have tested it on the Pioneer P3DXE platform in the environment described earlier. During the development phase, we used a 2D simulator to debug the system, making simplifying assumptions to ease the process, but the physical world poses additional operational issues. Only by demonstrating our system on an actual robot can we be sure that our technical approach is robust in the face of these issues. Here we consider one run in detail and then present the highlights from additional tests.

Our first demonstration involved the English command that we presented in the beginning:

First Until you find open door D
Holds you go down the hallway
Next you go through D.

As explained previously, the first step in processing this task description is to pass it through a compiler that transforms it into a list-structure representation. As background knowledge, this process depends on the presence of basic concepts for domain-specific conceptual predicates like *open-door* and domain actions like *go-through*. The second step, which we also discussed earlier, involves translating the list-structure encoding into a set of hierarchical ICARUS skills that refer to the conceptual predicates in their conditions and effects. Finally, the system generates a top-level intention, *(skill-2 ?d you)*, which it asks ICARUS to execute.

Given this intention and its supporting structures, the system invokes ICARUS to make conceptual inferences and execute relevant skills to carry out the intention, relying on `DIARC` to provide it with percepts and to carry out low-level activities like moving through a door and going down a hall. For this task, ICARUS initially executes skills that move the robot down the hallway, calling on `DIARC` to handle the necessary motion commands. This continues for several ICARUS cycles until the robot reaches a door, leading to satisfaction of the 'until' condition and to completion of the first subskill. ICARUS then calls on another translator-generated skill to move the robot through the door, again invoking `DIARC` to handle details of environmental execution.

Our second demonstration involved an English instruction of similar complexity that illustrates sequential behavior:

> Go straight through the door in front of you and turn left, then go down the hallway and turn right.

Using our constrained English syntax, we rephrased this statement as

If you are a robot
Then First If D is a door in front of you
Then you go through D
Next you turn left
Next you go down the hallway
Next you turn right.

Conversion of this command to the list-structure notation and translation into ICARUS structures produces the two skills

```
((skill-4 ?you)
 :conditions ((robot ?you))
 :subskills  ((skill-3 ?d ?you) (turn-left ?you)
              (go-down-hallway ?you)
              (turn-right ?you)))
((skill-3 ?d ?you)
 :conditions ((door-in-front ?d ?you))
 :subskills  ((go-through ?you ?d)))
```

In addition, the system generates the top-level intention *(skill-4 you)*, which it passes to ICARUS for execution.

In this run, ICARUS infers that a door is directly in front of the robot and calls `DIARC` to maneuver through it. Once the latter reports success, ICARUS again invokes `DIARC`, first to turn the robot to the left and then to move it down the hallway. Some time later, when `DIARC` completes these activities, ICARUS tells it to turn the robot right. Finishing this behavior means the entire task is done, at which point the system halts and awaits further instructions.

A final task that illustrates the conditional execution of behavior involves the command:

> Go through the first doorway after the blue box.

This time, we rephrased the instruction in our constrained syntax to be

If you are a robot and
B is a box and B is blue and
D is the door immediately after B
Then you go through D.

In this case, the conversion and translation processes generated only the single ICARUS skill

```
((skill-5 ?you ?d ?b)
 :conditions ((robot ?you) (box ?b) (blue ?b)
              (door-immediately-after ?d ?b))
 :subskills  ((go-through ?you ?d)))
```

along with the intention *(skill-5 you ?d ?b)*, which the system calls on the architecture to execute. Here ICARUS must notice a blue box that `DIARC` detects for it, infer which door comes immediately after the box, and call on `DIARC` to maneuver the robot to that door and through it. Afterward, the agent does not go through any more doors without further commands, since it has completed the specified task.

Although these example traces do not cover the entire range of instructions that can arise in our setting, they provide clear evidence that our system operates as planned, converting English commands into list structures, translating this internal notation into hierarchical skills, and using one of these skills to state a specific intention. The system then calls on ICARUS to execute the intention, which in turn invokes `DIARC` to control the robot. The demonstrations provide support for the three claims stated earlier: that our command language

covers a wide range of robot behaviors, that our translation mechanism converts these commands into legal ICARUS skills, and that the hybrid ICARUS-DIARC system executes the specified behavior in a realistic setting. The integrated character of our approach makes it difficult to evaluate these claims separately, but the successful runs with a physical robot suggest their viability. Naturally, we plan to carry out additional tests on other complex tasks in this domain, but the results to date have been encouraging.

Related Research

There has been considerable work in the human-robot interaction community (Goodrich & Schultz, 2007) on coordinating teams of humans and robots, examining both high and low levels of interaction, and ranging from one-on-one coordination to managing large robot teams. For example, Kennedy et al. (2008) advocate using detailed mental simulation of a single human teammate's reasoning and choices in order to predict his actions and thus avoid wasted effort. At the other extreme, Lewis et al. (2006) have examined ways that a human operator can teleoperate large robot teams without encountering cognitive overload.

Natural language interaction is widely viewed as the most promising communication medium for human-robot teams, and several projects have explored integration of natural language capabilities in robot systems. For instance, Rybski et al. (2008) describe an approach that lets a human teammate teach a robot new action scripts using natural language descriptions, although the scripts are fairly simple and descriptions must conform to a highly structured subset of language that mirrors the procedural semantics of action execution. Another approach, reported by Brenner (2007), uses knowledge about the robot's capabilities to aid language understanding. This method relies on a correspondence between words and terms that appear in models of actions' conditions and effects. This connection aids determination of parts of speech and, more generally, understanding of commands in the domain.

There has been extensive research on formal representation languages for expressing the complex activities that arise in many planning and executon tasks. For example, Baier and McIlraith (2006) present a heuristic approach to planning that converts goals stated in a variant of linear temporal logic into nondeterministic finite automata over which planning techniques can operate. In other work, Fainekos et al. (2009) describe a motion-planning system that converts high-level behavior descriptions (e.g., a sequence of desired actions) expressed within linear temporal logic into robot programs. Similarly, Grosskreutz and Lakemeyer's (2000) cc-GOLOG extends the logic programming language GOLOG to robotic tasks by supporting the statement of complex hierarchical procedures. However, none of these systems actually control a physical robot; the resulting plans have been tested only in simulation or compared to "known" solutions.

Finally, there have been successful efforts to control robots using other agent architectures, such as SOAR and ACT-R. Laird et al.'s (1991) Robo-SOAR interacted with the external environment via a camera and robotic arm to achieve simple block manipulation goals. Although the system could accept advice when unable to solve a problem itself, this was restricted to suggesting a particular task operator and was not provided in natural language. ACT-R/E enables control of mobile robots by integrating modules that interact with sensors (e.g., visual and aural), reason about the world (e.g., localization), and carry out actions (e.g., locomotion and speech production). Kennedy et al.'s (2008) work with ACT-R/E is similar to our integrated approach, but it does not translate natural language instructions into a formal notation before further processing or construct an explicit skill hierarchy.

Hence, although our work incorporates ideas from each of these research projects, there have been no reported efforts on human-robot coordination that combine presenting commands in constrained natural language, translating them into a formal notation, and executing them via an agent architecture like ICARUS. This integrated approach offers clear advantages for human-robot interaction, since both the high-level goal language and the architecture's internal structures for beliefs, goals, and intentions remain close to natural language, which humans use effectively in coordination. Thus, our work makes clear contact with previous results but combines them in new ways to provide novel capabilities for human-robot coordination.

Concluding Remarks

In this paper, we reported a novel approach to human-robot interaction that supports communication of high-level but complex tasks and their robust execution. Our approach relied on three distinct but integrated components. The first specifies instructions in a constrained version of English, relying on pairs of key words that allowed direct conversion into an internal list-structure notation. The second translates these list structures into a set of hierarchical skills that can be interpreted by ICARUS, an agent architecture that supports conceptual inference and teleoreactive execution. The third specifies an intention based on the command, expands the skills in a top-down manner, and passes information to DIARC, a robotic architecture, to handle the details of controlling the physical device, which returns results to ICARUS for use on future decision cycles. We demonstrated this integrated system on a Pioneer robot in a multi-room setting on tasks that could arise in search-and-rescue operations.

The encouraging results from these experiments suggest that we should continue pursuing this approach to human-robot interaction, but it is also clear that our initial system would benefit from a variety of extensions and studies. One natural avenue for future work involves adding knowledge to DIARC and ICARUS that would let them recognize a broader range of objects and

act appropriately in response to them. We should also examine empirically whether humans can use our command language effectively on realistic search-and-rescue tasks. It seems possible that they will find the syntax overly constraining, in which case we should explore more flexible approaches to processing robot instructions stated in natural language (e.g., Dzifcak et al., 2009; Kress-gazit et al., 2008).

In the longer term, we should take advantage of ICARUS' ability to solve novel problems (Langley et al., 2009), so that our integrated system can handle complex tasks even when it lacks domain-specific skills for their components. Moreover, the architecture's capacity for acquiring such skills from successful problem solving should support learning of high-level commands, which would let humans simplify their instructions to robots and allow even more efficient communication. A more radical extension would let the agent draw inferences, based on a human's commands and questions, about the latter's beliefs, goals, and intentions, which in turn would further reduce the need for detailed commands. Taken together, these extensions would let the ICARUS-DIARC combination play an even fuller role as a member of a human-robot team.

Acknowledgements

The research reported in this paper was funded in part by MURI Grant No. N00014-07-1-1049 from the Office of Naval Research. The views and conclusions contained herein are the authors' and should not be interpreted as representing the official policies or endorsements, either expressed on implied, of ONR or the U. S. Government. We thank Wende Frost, Ravi Gummadi, and Anupam Khulbe for their earlier contributions to the project.

References

Baier, J. A., & McIlraith, S. A. (2006). Planning with first-order temporally extended goals using heuristic search. *Proceedings of the Twenty-First National Conference on Artificial Intelligence* (pp. 788–795). Boston: AAAI Press.

Brenner, M. (2007). Situation-aware interpretation, planning and execution of user commands by autonomous robots. *Proceedings of the Sixteenth IEEE International Symposium on Robot and Human Interactive Communication*. Jeju Island, Korea: IEEE Press.

Dzifcak, J., Scheutz, M., Baral, C., & Schermerhorn, P. (2009). What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. *Proceedings of the 2009 International Conference on Robotics and Automation*. Kobe, Japan.

Fainekos, G. E., Girard, A., Kress-Gazit, H., & Pappas, G. J. (2009). Temporal logic motion planning for dynamic mobile robots, *Automatica*, *45*, 343–352.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated planning*. San Francisco: Morgan Kaufmann.

Goodrich, M. A., & Schultz, A. C. (2007). Human-robot interaction: A survey, *Foundations and Trends in Human-Computer Interaction*, *1*, 203–275.

Grosskreutz, H., & Lakemeyer, G. (2000). cc-GOLOG: Towards more realistic logic-based robot controllers. *Proceedings of the Seventeenth National Conference on Artificial Intelligence*. Austin, TX: AAAI Press.

Ingrand, F. F., Georgeff, M. P., & Rao, A. S. (1992). An architecture for real-time reasoning and system control. *IEEE Expert*, *7*, 34–44.

Kennedy, W. G., Bugajska, M. D., Adams, W., Schultz, A. C., & Trafton, J. G. (2008). Incorporating mental simulation for a more effective robotic teammate. *Proceedings of the Twenty-Third Conference on Artificial Intelligence*. Chicago: AAAI Press.

Kress-Gazit, H., Fainekos, G. E., & Pappas, G. J. (2008). Translating structured English to robot controllers, *Advanced Robotics*, *22*, 1343–1359.

Laird, J. E., Yager, E. S., Hucka, M., & Truck, C. M. (1991). Robo-SOAR: An integration of external interaction, planning, and learning using SOAR. *Robotics and Autonomous Systems*, *11*, 113–129.

Langley, P., Choi, D., & Rogers, S. (2009). Acquisition of hierarchical reactive skills in a unified cognitive architecture, *Cognitive Systems Research*, *10*, 316–332, 2009.

Langley, P., Laird, J. E., & Rogers, S. (2009). Cognitive architectures: Research issues and challenges, *Cognitive Systems Research*, *10*, 141–160.

Lewis, M., Polvichai, J., Sycara, K., & Scerri, P. (2006). Scaling-up human control for large scale systems. In N. J. Cooke, H. Pringle, H. Pedersen, & O. Connor (Eds.), *Human Factors of Remotely Operated Vehicles*. New York: Elsevier.

Nilsson, N. (1994). Teleoreactive programs for agent control. *Journal of Artificial Intelligence Research*, *1*, 139–158.

Rybski, P. E., Stolarz, J., Yoon, K., & Veloso, M. (2008). Using dialog and human observations to dictate tasks to a learning robot assistant. *Journal of Intelligent Service Robots*, *1*, 159–167,

Scheutz, M. (2006). ADE - Steps towards a distributed development and runtime environment for complex robotic agent architectures, *Applied Artificial Intelligence*, *20*, 275–304.

Scheutz, M., & Andronache, V. (2004). Architectural mechanisms for dynamic changes of behavior selection strategies in behavior-based systems, *IEEE Transactions of System, Man, and Cybernetics Part B*, *34*, 2377–2395.