Enabling Spoken Dialogue Interaction About Team Activities

Laura M. Hiatt[1] and Lawrence Cavedon[2]

[1] Computer Science Dept., Carnegie Mellon University, USA

[2] CSLI, Stanford University, USA

Abstract. Spoken language dialogue is a powerful mode for human-robot interaction (HRI) in complex, dynamic environments. We describe extensions to an existing dialogue management system that enables activity-oriented interaction with multi-robot teams.

1 Introduction

There has been much research in the Multi-Agents Systems literature on frameworks for robustly managing multi-agent/robot teamwork. Some of the more succesful approaches involve agents explicitly representing team-level goals and intentions, and reasoning about their own actions in the context of the team activity (e.g. [1, 2]). There has also been much work on algorithms and techniques for automated task-allocation and -negotiation, multi-agent planning, task coordination, etc.

Conversely, our approach is to facilitate human involvment in the multi-agent/robot task environment. Rather than relying on automated management of multi-agent team issues, our system engages in conversation with a human operator to perform task-allocation and to resolve issues that may arise. This is not inconsistent with a partially-automated approach (and our system does implement some automatic task-reallocation on failure); however, for the purposes of our research, we delegate as much of the team-level decision-making to the human operator, and provide the framework for collaboration between human and agents via spoken-language dialogue. In general, we believe an appropriate model is one of *adjustable autonomy* [3], whereby multi-agent team processes are automated, but human-interaction is initiated when this is more appropriate. The spoken-language dialogue interface supports the human-robot interaction *within the context* of multi-agent teamwork when required.

1.1 Human-Robot Interaction (HRI) via Task-Oriented Dialogue

Previously at CSLI we have built spoken-language dialogue interfaces for human interaction with individual robots. The CSLI Dialogue Manager (CDM) is the core of a multi-domain dialogue system that has been applied to a number of applications, including control of a (simulated) robotic helicopter [4].

M. Hiatt L. and Cavedon L. (2005).
Enabling Spoken Dialogue Interaction About Team Activities.
In *Proceedings of the 1st International Workshop on Multi-Agent Robotic Systems*, pages 23-30
DOI: 10.5220/0001196400230030

Collaborative speech-based dialogue offers a powerful medium for interaction between humans and complex robots operating in dynamic, real-time environments, containing multiple concurrent activities and events which may succeed, fail, become cancelled or revised. An interface to a robot operating in such conditions must be interruptible, context-dependent, and amenable to multiple threads of conversation, and this has been one of our prime foci when building dialogue interface systems for human-robot interaction. *Task-Oriented Dialogue*, i.e. dialogue about tasks jointly performed by the human and robot, also requires the context of the task to be represented, to enable for robots to inform the human operator of progress on tasks, or to answer questions relating to task-performance or status (e.g. "Why are you doing that?").

These issues are exacerbated in multi-robot settings. The need for natural collaborative interfaces is increased, due to the increased cognitive load involved in interacting with multiple robots. Implementing dialogue interaction with multiple robots, however, introduces new technical challenges, such as: discussing the various stages of team-activity; minimizing the impact of information overload on the human operator; and managing multi-party conversation (e.g. recognizing which robot is being spoken to).[3]

1.2 Team Dialogue

The following short dialogue illustrates some of the functionality we have implemented. This is taken from a (simulated) multi-robotic helicopter domain and illustrates: (i) a team-formation sequence involving a robot acting as *team-leader* (*1-6*); (ii) individual robots reporting on their actions (7-9); (iii) a sub-dialogue directed at a specific robot that involves linking its individual sub-tasks to the overall team goal (10-15).

1. User: Organize a team to fight the fire at the tower.
2. Team Leader: OK. I will organize a team to fight the fire at the tower.
3. Team Leader: UAV1 will confirm that there is a fire at the tower, and then UAV1 and UAV2 will fight the fire.
4. Team Leader: Have I formed an acceptable team?
5. User: Have UAV2 confirm the fire instead.
6. Team Leader: OK. UAV2 will confirm that there is a fire at the tower, and then UAV1 and UAV2 will fight the fire.
7. UAV2: I am taking off and flying to the tower.
8. UAV2: I have flown to the tower. There is a fire at the tower.
9. UAV1: I am taking off and flying to the tower.
10. User: UAV2, what are you doing?
11. UAV2: I am flying to the lake.
12. User: Why?
13. UAV2: To pick up water.
14. User: Why?
15. UAV2: Because we are fighting the fire.

This paper describes initial work extending the CSLI Dialogue Manager (CDM) to allow it to support multi-agent task-oriented dialogues of the above form. The focus

[3] These and other issues for multi-party dialogue systems are discussed in [5].

of this paper is not so much on linguistic issues. Rather, we focus on describing the task-representation framework required to facilitate the type of interaction for human support of multi-agent teams via dialogue. Our approach is to explicitly model team activity, extending the single-agent model of activity implemented in the single-agent version of the CDM. The team-activity model is reminiscent of *team-plan* approaches to multi-agent teamwork (e.g. [1, 2]) and is used by the CDM to relate individual-robot activity to team-level tasks (e.g. as in *15* above). Other extensions to the CDM include supporting discussion about team issues (such as team-formation and task-allocation, as in 1-6 above), and allowing a mix of directly addressing the team (via the "team-leader"), individual agents (for task updates), or not using direct-addressing at all (and having the CDM determine the appropriate agent to respond, as in *13* and *15* above).

2 Dialogue System Architecture

The CSLI Dialogue System is designed around a component-based architecture, making use of existing systems for speech recognition, natural language parsing and generation, and speech synthesis. These and other components (e.g. interactive map GUI) are integrated using an event-driven loosely-coupled distributed architecture, allowing easy replacement of components by others of similar functionality.

Incoming speech utterances from the user are analyzed and classified as a type of *dialogue move*—i.e. as a *command*, *question*, *answer* to a question, etc.—and passed on to the Dialogue Manager (CDM), which performs a number of processes: resolving pronouns (e.g. "it") and object descriptions (e.g. "the red car") to actual objects in the robot's universe; resolving any ambiguous requests, or generating appropriate questions to do so (e.g. "Which tower do you want me to fly to?"); taking the appropriate action corresponding to the incoming dialogue move (e.g. a *command* may activate a new agent task); and generating any appropriate response (e.g. the answer to a question asked by the user).

The two central components of the CDM are the *Dialogue Move Tree (DMT)* and the *Activity Tree*. The DMT represents the history of the dialogue, in particular, the most recent salient dialogue context; this is used to interpret incoming utterances. The DMT is designed to support multi-threaded, multi-topic conversations, as required for conversations in highly dynamic complex domains (see [4, 6]). Our focus here is on the Activity Tree.

2.1 Task-Oriented Dialogue and the Activity Tree

The CSLI Dialogue Manager has been designed to support *joint-activities* which require collaboration between the human operator and an intelligent agent/robot. As the human operator works with the agent, it is critical that they maintain a shared conception of the state of the world, and in particular the status of the activities being performed.

To facilitate this, the Dialogue Manager incorporates a rich *Activity Model* to mediate between it and the agent/robot. The Activity Model consists of a language for writing activity *recipes* that support conversation about the activities the agent actually performs. These activity representations play a number of important roles within the

Dialogue Manager: identifying the properties of activities that must be specified by further dialogue (e.g. the *destination* of a "fly" command); disambiguating requests; and providing a framework for negotiating resource-conflict. We focus here on their use in the *Activity Tree* to model the status of activity-execution.

A recipe is a declarative description of a command (such as *fly*) and the slots to be filled by the arguments the command takes (such as *destination*). Hence, a recipe is reminiscent of a STRIPS planning operator, but is not used to control behaviour, only to represent the actions an agent can perform in order to support conversation about them. In this sense, the library of recipes is similar in concept to the use of plan libraries in [7–9].

A spoken command uttered by the human operator is analyzed and translated into a partially specified *Unresolved Activity*, which is added to the Activity Tree. An Unresolved Activity is a slot-based representation of the incoming utterance without fully interpreting the language-based descriptions. For example, a command such as "Go to the tower" generates an Unresolved Activity as follows:

action: `verb(go)`
arg: `pp(to, [the tower])`

where *verb* indicates the action word and *pp* indicates a phrase describing an argument to the action.[4] At this point, different possible activities could actually match the request: e.g. a robot may *walk* or *fly* to fulfill a "go" command, depending on its capabilities (defined via its Activity Model).

The system attempts to fully *resolve* the activity: mapping the verb to the appropriate Activity Model recipe,[5] finding objects that are referred to, or asking the human operator questions in order to gather further information if required. Once the activity has been fully resolved—i.e., once enough information has been gathered so the activity can be executed—it is sent to the agent to be executed. As it is executed, the agent updates the state of the activity, which is reflected in the activity's node on the Activity Tree, and the dialogue manager in turn is notified of these updates: see Figure 1.[6]

At any point, an activity can be in one of a number of states: `not_resolved` (partially described), `resolved` (fully described), `current`, `suspended`, `cancelled`, `done`, etc. The Activity Tree, then, mediates all communication between the dialogue manager and the agent, providing a layer of abstraction by which the dialogue manager can communicate with any task-oriented agent.

This architecture contrasts with a simple command-and-control approach in which the dialogue manager simply sends off commands to the agent and then forwards all reports sent to it by the agent to the human operator. First and foremost, the tree structure provides an explicit means of representing the relationship among activities being executed by the system. The decomposition allows the dialogue system to make intelligent

[4] PP denotes *prepositional phrase* but the linguistic category is not important here.
[5] Mapping rules to perform this are part of the Activity Model.
[6] Ideally, this process keeps the Activity Tree synchronized with the agent's own task execution, although some lag is possible.

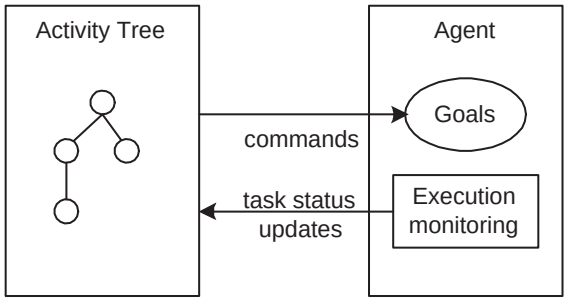

Fig. 1. Interaction between Activity Tree and agent

decisions about what changes in the world are conversationally appropriate, and which need not be mentioned.[7]

Finally, the Activity Tree provides a framework for answering queries about activity state ("What are you doing?"), revising a previous command or utterance ("Make that the tower instead"), and for explaining behavior ("Why are you doing that?"); see [10].

3 Multi-Agent Dialogue Management

Extending the CDM to handle multi-agent dialogue involved two architectural changes:

1. adding an *Agent Manager* to mediate between the CDM and multiple agents;
2. extending the Activity Tree to contain activities performed by multiple agents.

3.1 Agent Manager

The Agent Manager is responsible for selecting the most appropriate agent to handle an issued command, or to handle a query. Each dialogue-enabled agent registers itself with the Agent Manager by providing:

1. a unique name;
2. a description of its capabilities (i.e. an Activity Model);
3. an agent *DM proxy*, i.e., agent-specific code that interprets the slot-based Unresolved Activity representation of commands as constructed by the Dialogue Manager.

User utterances, particularly commands, are distributed to the appropriate device as follows: when possible, a user utterance is assumed to continue an existing thread of conversation, e.g. as in a follow-up question such as *12* or *14* in the example in Section

[7] In general, the expectation is that task decomposition and execution is handled by some external plan-execution engine residing on the agent. This requires some integration with the plan-executor itself—notifications of task status changes need to be sent to the Activity Tree.

1.2. If this is not the case—e.g. it is a new command—then the Agent Manager determines which agent should receive the utterance, based on capabilities. An Unresolved Activity representation of the command is sent to each registered agent proxy, which then attempts to interpret what the words and phrases in the Unresolved Activity mean, and whether they correspond to actions that the agent can perform (i.e. whether the action matches a recipe).

Each agent indicates whether it can definitely/possibly/not handle the command. If one of the agents that can handle the command is the *most conversationally salient* one (i.e. the agent to have most recently communicated with the user),[8] then this agent is selected to handle the command. For some commands, it is possible that multiple agents could handle it, even in different ways: e.g. a UAV or a ground robot could both handle "Go to the tower". In the absence of definitive selection criteria, the Agent Manager generates a disambiguation question for the user to answer, thus collaborating with the user in this decision-process:

User: Check if there is a fire at the tower.
DM: There are two agents that can currently perform this task. Should I assign this task to Robot1 or Robot2?
User: Robot1.
DM: OK.

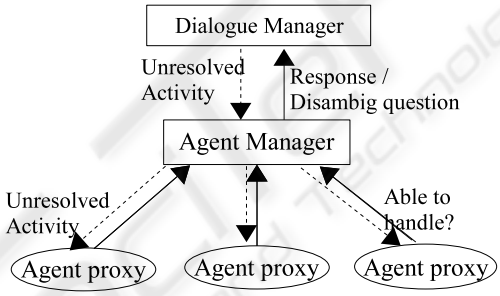

Fig. 2. Information flow during task assignment from utterance

3.2 Multi-Agent Team-Task Management

In order to facilitate task-oriented communication between the user and multiple agents, an integrated Activity Tree is maintained by the CDM, containing all tasks that any agent has handled. The Activity Tree is synchronized with each agent's task execution management, which allows the dialogue manager to provide generic feedback about the progress, success and failure of tasks belonging to any and all agents. Thus reports about task status, completion, failures, etc., are still generated from the Activity Tree.

[8] Salience is reset out after a specified time period.

Team tasks generally require multiple agents to execute them. Our approach involves assigning a complex task to a single agent, designated as being in the role *team-leader*. A *team-leader* agent's Activity Model contains recipes corresponding to tasks designed to be carried out by multiple agents. In effect, the multi-agent team task is assigned to the *team-leader* agent, which is then responsible for *delegating* sub-tasks to appropriate agents via the Agent Manager—i.e. if the *team-leader* agent's Activity Model contains a sub-task marked as to be performed by a different agent, then it sends a request to the Agent Manager which allocates an agent to handle the sub-task.

The basic Activity Model scripting language has been extended to accommodate the notion of task-delegation by allowing optional tags against tasks to specify which agent should handle a sub-task. The special tag *SOME-OTHER* is used to designate that a different agent (as opposed to the agent to which the task script belongs) should be assigned the subtask. The tag *ALL-AGENTS* can also be used to designate that all agents that support this activity should be assigned the subtask. If no appropriate agent is found, then the parent task fails.

Defining team behaviour via such team-activity recipes allows constraints to be set on team activity across multi-robot activity. For example, a *team search* may specify that different robots search different locations of the area to be searched. Further, the notion of task success can be explicitly tied to success by a specific team-member, all team-members, or any team-member. For example, a *team search* task can be defined to succeed if *any* robot involved in the task succeeds in finding the target object. Once the team goal is successful, all team participants are informed of this fact, leading them to terminate their own activity on any related sub-task (c.f. [1]).

In order to maintain agent-independent task execution, each agent only maintains knowledge about its own sub-tasks and their progress. Further, the agent responsible for the main task keeps a record of which agents were assigned to sub-tasks, but does not maintain any information on further sub-task decomposition. Only the CDM's Activity Tree maintains a complete team-activity representation, and hence can support dialogue on the complete team activity. Agents also maintain the relationship to the team-goals that spawned their individual tasks: this is used to implement a commitment to the team-goal, which is an important requirement for robust team-activity [1], and also allows individual agents to support dialogue on how their current activity relates to the team-level goals.

3.3 Dialogue About Team Formation

Explicit representation of team activity enables dialogue about the teamwork process itself, such as team formation. For example, the process of assigning agents to roles in a team Activity Model may be communicated to the user, who then has the option of confirming or modifying the team. This contrasts with the many techniques (e.g. auctions, Contract Net-style bidding) used for team formation and task allocation in the multi-agent literature, and particularly in role-filling in open agent societies [11]; again, our approach is that this is process is performed in collaboration with the user. The first part of the sample dialogue in Section 1.2 illustrates team formation. Team formation dialogue can also be seen as a rudimentary form of negotiation, whereby the *team-leader* agent proposes a team and the user accepts or revises it.

Similarly, the *team-leader* agent is able to reorganize a team if one of the members fails to complete its sub-task. Again, this is done in consultation with the user—the standard task-reporting functionality [6] of the CDM reports the task failure, and the *team-leader* agent reports on attempts to re-assign the sub-task to a different agent.

4 Discussion

We have described initial steps towards flexible dialogue with teams of robots, intermingled with dialogue with individual team-members. We believe the approach is important: managing cognitive load during interaction with multiple robots and their tasks requires the user to be able to shift level of discussion to the more abstract team-task level, rather than necessarily conversing at the individual level and managing all coordination and interactions between team members.

There are several directions to extend this work, both in terms of linguistic dialogue modeling and in terms of team-management. For example, we need to address more of the issues identified by Traum as problems in multi-party dialogue [5]. We also plan to support more elaborate negotiation dialogues, following [12]. We also need to address the information-overload issue that is exacerbated in the multi-robot setting; this could be partly alleviated by robots being aware of the complete dialogue context, including contributions by other robots.

References

1. Tambe, M.: Towards flexible teamwork. J. Artificial Intelligence Research **7** (1997)
2. Howden, N., Ronnquist, R., Hodgson, A., Lucas, A.: JACK intelligent agents—summary of an agent infrastructure. In: Int'l Conf. on Autonomous Agents, Montreal (2001)
3. Scerri, P., Pynadath, D., Tambe, M.: Toward adjustable autonomy for the real world. J. AI Research (2002)
4. Lemon, O., Gruenstein, A., Battle, A., Peters, S.: Multi-tasking and collaborative activities in dialogue systems. In: Proc. 3rd SIGdial W'shop on Discourse and Dialogue, Phil. (2002)
5. Traum, D.: Issues in multi-party dialogues. In Dignum, F., ed.: Advances in Agent Communication, LNAI 2922. Springer-Verlag (2004)
6. Lemon, O., Gruenstein, A., Peters, S.: Collaborative activities and multi-tasking in dialogue systems. Traitement Automatique des Langues (TAL) **43** (2002)
7. Allen, J.F., Schubert, L.K., Ferguson, G., Heeman, P., Hwang, C.H., Kato, T., Light, M., Martin, N.G., Miller, B.W., Poesio, M., Traum, D.R.: The TRAINS project: A case study in building a conversational planning agent. J. Experimental and Theoretical AI **7** (1995)
8. Fitzgerald, W., Firby, R.J.: The dynamic predictive memory architecture: integrating language with task execution. In: IEEE Symp. on Intelligence and Systems, Wash. DC (1998)
9. Rich, C., Sidner, C.L., Lesh, N.: COLLAGEN: Applying collaborative discourse theory to human-computer interaction. AI Magazine **22(4)** (2001)
10. Gruenstein, A., Cavedon, L.: Using an activity model to address issues in task-oriented dialogue interaction over extended periods. In: AAAI Spring Symposium on Interaction Between Humans and Autonomous Systems over Extended Periods, Stanford (2004)
11. Dastani, M., Dignum, V., Dignum, F.: Role-assignment in open agent societies. In: AAMAS-03, Melbourne (2003)
12. Traum, D., Rickel, J., Gratch, J., Marsella, S.: Negotiation over tasks in hybrid human-agent teams for simulation-based training. In: AAMAS-03, Melbourne (2003)