A Dialogue Architecture for Multimodal Control of Robots

Carl BURKE
The MITRE Corporation
7515 Colshire Drive
McLean VA 22102 USA
cburke@mitre.org

Lisa HARPER
The MITRE Corporation
7515 Colshire Drive
McLean VA 22102 USA
lisah@mitre.org

Dan LOEHR
The MITRE Corporation
7515 Colshire Drive
McLean VA 22102 USA
loehr@mitre.org

Abstract

Robots typically execute only pre-programmed, limited instructions. For humans to command teams of semi-autonomous robots in non-trivial, mobile, and dynamically changing tasks, the human-robot interface will need to include several aspects of human-human communication. These aspects include cooperatively detecting and resolving problems, making using of context, and maintaining contexts across multiple conversations. In this paper, we describe the architecture we are developing to support this dialogue system, based on the TRINDIKit framework.

Introduction

Robotics research has recently experienced a surge of interest due to a growing awareness that robots can work collaboratively with humans to perform tasks in situations unsafe for humans. The 1997 Mars Sojourner rover was tasked to act as a "mobile remote geologist" and conducted soil experiments in several different terrains (NASA 1997). Teleoperated robots assisted at the site of the World Trade Center in New York City after the September 11 attack. Robots were able to penetrate into areas of rubble debris in cavities too narrow and dangerous for humans and dogs (Kahney 2001). Finally, the US Government's Defense Advanced Research Projects Agency (DARPA) has invested substantial funding toward a vision in which robots will support future combat systems.

Despite this increased activity in robotics, relatively few advances have been made in the area of human-robot interaction. In a recent Robocup Rescue event, the best contenders in the competition relied upon teleoperation (joystick-style control) by human controllers (Eyler-Walker, p.c.). Though ultimately supervisory control of teams of semi-autonomous robots is a very promising avenue for future research in robot search and rescue, this technological approach does not yet reach the level of competence of teleoperation. Recently, NASA has been concerned with human-machine interaction are commanded by high-level commands rather than sequences of low level commands. A grapefruit-sized Personal Satellite Assistant (PSA) is being developed to operate aboard the Space Shuttle's flight deck. It will navigate using its own navigation sensors, wireless network connections, and propulsion components. Rayner et al. (2000a, 2000b) describe an architecture for a spoken interface with the PSA.

An alternative approach to human-robot interaction by Fong et al (2001) bridges teleoperation with "collaborative control". In this model, humans and robots act as peers exchanging information in dialogue to achieve goals. Instead of controlling the vehicular robot solely by direct (manual) control, the human specifies a set of waypoints that the robot can achieve on its own. One problem observed with waypoint driving is that robots may encounter obstacles for which its vision system is inadequate to assess. In such a circumstance, the robot can query the human about the nature of the obstacle and receive assistance.

In this paper we describe a dialogue architecture we are developing for a Personal Digital Assistant (PDA)-based dialogue interface to a robot, which we plan to extend toward a team-based search and rescue task. Currently, the PDA supports single user, single robot dialogue in a limited navigation and question-answer scenario for visitors to a technology trade show. Using touch gestures and speech, users may ask the robot to guide them to a particular booth, show images from remotely located robots, and answer questions about exhibits at the trade show.

Our primary research interest is the development of a dialogue system architecture robust enough to tolerate continuous operational use, flexible enough for porting to different domains and tasks, and able to support multiple, simultaneous conversations involving one or more humans and one or more cooperative robot entities. The dialogue management architecture we are developing is based on the TRINDIKit (Task oRiented Instructional Dialogue Toolkit) (TRINDI 2002) framework, although we have introduced a number of implementational changes in the reengineering of a TRINDIKit architecture.

1 Original Architecture

Our architecture was first assembled for development of a demonstration system called Mia, the MITRE Information Assistant. Mia is an information kiosk equipped with a touch screen and a microphone, and stocked with information about MITRE's internal research projects for use as a visitors' guide to a MITRE trade show.

Mia was built as a set of independent modules that communicated using SRI's Open Agent Architecture (OAA). The Graphical User Interface (GUI) was written in Tcl/Tk. The GUI handled push-to-talk for the speech recognizer, maintained a text menu of possible user utterances, showed a map of the overall trade show layout with the ability to zoom in on specific rooms, and displayed prerecorded output videos of the animated agent speaking and gesturing. Dialogue management was done with the TRINDIKit system.

TRINDIKit itself provides the basic infrastructure of a dialogue manager. It provides structured data types and the means to define an Information State (IS) from those types, a language for defining the modules of a Dialogue Move Engine (DME), and a language for controlling the application of individual modules to the job of dialogue management. With all TRINDIKit provides, it does not implement a theory of dialogue. For that we used the GoDiS (Gothenburg Dialogue System) (Larsson et al 2000) system, which implements the Questions Under Discussion model in TRINDIKit. We were able to adapt existing GoDiS dialogues to our kiosk domain in a very short time.

In order to integrate TRINDIKit into the kiosk using the OAA, we used TRINDIKit's concurrent mode, which incorporates support for use of the OAA. While this seemed to be a natural choice, and allowed more natural definition of module interactions, it also raised several problems, as discussed below.

1.1 Speed

TRINDIKit in concurrent mode ran very slowly, on a 750 MHz Pentium2 with 384 MB RAM running WindowsNT and no other processes. We believe using the OAA for data transport caused the delays, as a large number of messages were exchanged. Lewin et al (2000:45) report that running GoDiS with TRINDIKit on the OAA yielded a 2-second user utterance to system utterance time, compared to a 0.5 second time when using TRINDIKit's internal agent environment (which is not available for use with non-prolog components). Although modules run independently in concurrent mode, updates to IS were still transmitted to each module individually. Updates were sent whether they were used by that module or not, and all other processing waited until that module finished its work.

1.2 Data Consistency

TRINDIKit does not exercise good controls over asynchronous modifications to IS. At one point we had to build artificial delays into our system to work around these limitations. The dialogue manager we built for Mia was based on GoDiS, which requires very structured turn-taking. In several cases, however, the interactions with the user flowed better if these responses were automatic. Processing was sufficiently slow that our GUI's automatic acknowledgement often arrived and was processed before TRINDIKit was finished cleaning up from the previous utterance. As a result, it was possible to change the IS twice before the DME could respond to one change, and the system lost track of the dialogue state. Consistency of data needs to be assured throughout the design of the system.

1.3 Inconsistent Semantics

We encountered situations where constructs of the GoDiS plan language were interpreted differently depending on the depth of the plan. With the proliferation of small languages im-

plemented by different sets of macros, it was difficult to track down bugs in the rules and conversation scripts. This was made more difficult by the nature of Prolog. Clauses that fail do not normally generate any error messages, because failure is a normal aspect of program execution. Unfortunately, database bugs and misspelled names often caused unexpected failures, causing the system to generate either no response or a response that looked reasonable but was in fact incorrect. We feel it's necessary to provide explicit notification of certain kinds of failure, such as failure to find a named variable, failure to find a matching value in a table, and so on.

1.4 Lack of Multimodal Support

Neither TRINDIKit nor GoDiS provides any direct support for multimodal processing. The primary interface driving the development of these systems was language; there is no separation of events by source, no temporal tagging of input events, and no provision for assessing temporal relationships between different inputs.

2 Revised Architecture

We are moving ahead with the design for a dialogue manager for robot control. From our experience with the dialogue manager in Mia, we're convinced of the advantages of a kit-based approach. We feel that TRINDIKit was a good first cut at it, and hope that our efforts will lead to a second, somewhat better iteration.

2.1 Distributed Information State

We've chosen to model all of our module interactions as if they were asynchronous. This provides the cleanest separation of modules, and the cleanest conceptual integration with the asynchronous requirements of robot control. Our approach to solving this problem is to define an explicit interface definition language, which will be used to define every module's interface with the outside world. We explicitly include the information state structure in this interface definition, perhaps as a module in itself. Since TRINDIKit does not include a separate language for specifying module interfaces, we are designing our own. This language is analogous to CORBA Interface Definition Language, but with less concern for the physical implementation.

There are a large number of protocols and infrastructures that have been developed to support communications between agents, each with characteristics optimized for particular tasks or emphasizing desired functionality. We intend to support small standard set of operations that have wide applicability across programming languages and communication protocols.

2.2 Controlled Extensibility

Our interface specifications will need to be translated into specific computer languages before they can be executed. The translation will vary depending on the underlying protocol used to communicate between modules. While we want to support the widest possible audience, we don't want to get bogged down in the construction of translators for every possible set of implementation language and protocol. Our approach is to exploit an existing standard set of translation software, namely XML and XSLT processors such as Xalan. We are specifying a dialect of XML for modules interface definitions, and a small set of templates for realizing interfaces with specific combinations of programming language and protocol. Additional templates can be written as necessary to extend the kit to other languages and protocols without requiring modification of the kit itself.

The same approach extends to the specifications of DME rules, module synchronization and control, and the definition of new "languages" for the kit. We have defined what well-formed formulas look like in our kit's scripting language: what names look like, the types of expressions that are possible, how expressions and statements are aggregated to form programs. What is left unspecified is the exact sequences of expressions that form statements in any particular script language. Those are specified using templates analogous to XML DTDs, which gives us the flexibility to define new constructs as needed.

2.3 Rule Engine

The DME rules in TRINDIKit have strong similarities to rules in expert systems. We plan to implement these rules in both a sequential form, equivalent to the current TRINDIKit, and in an expert system form which may be more efficient. We expect that there will be differences in operating characteristics between those

two styles, and we want to identify and quantify those differences.

One issue we must address in our design is unification. While logic variables are natural for modeling discourse given the history of the field, most languages typically used to implement robot software do not support it directly. Our kit must ensure that sound unification procedures are provided for every language it supports, so that semantics are common across all realizations of a script. We must also provide for backtracking or iteration through the set of alternatives in a straightforward way.

2.4 Control and Synchronization

Our primary focus is multimodal communication, potentially multiparty as well. We are extending TRINDIKit's triggers to include support for consideration of temporal relationships between events, both within and across modes.

2.5 Integrated Environment

An ideal toolkit would have an integrated set of tools for designing, testing, and debugging dialogues. We would like to support static and dynamic analysis of dialogues, recording and playback of dialogues, graphical dialogue design tools, a "validation suite" of tests to support extension of the toolkit to new programming languages and agent protocols, and above all, the ability to plug-in as-yet-undefined capabilities.

3 Future Work

Significant effort has been devoted to defining our mutable language capability. This capability provides both a reasonable transition path from TRINDIKit scripts and a means for specifying module interfaces and information state structure using a common XML representation.

Our intent is to provide support for several different transport mechanisms to explore the limitations of our approach. To date, we have completed an initial interface definition specification and have developed templates to realize those interfaces with the OAA. DARPA's Galaxy Communicator is the second transport mechanism we will be considering. Time and resources permitting, we will examine some additional transports with differing characteristics, such as CORBA, Java Remote Method Invocation (RMI), or Linda.

We have devoted considerable time to up-front consideration of scripting languages, portable code generation, and module communications, and are now beginning the task of implementing our versions of the TRINDIKit scripting languages. Our target realization for these scripts is a combination of Java code and expert systems that can be executed within a Java program.

We plan to port and formally evaluate our dialogue toolkit within three domains (question-answering, automated tutoring, and multimodal robot control). Our dialogue toolkit will be openly available, as well as sample implementations for each of these domains.

Conclusion

We have described our evolving architecture (based on the TRINDIKit framework) for a flexible dialogue system capable of robust, multimodal, multiparty control of robots.

References

Fong T., C. Thorpe, and C. Baur (2002), Robot as Partner: Vehicle Teleoperation with Collaborative Control, Workshop on Multi-Robot Systems NRL, Washington, D.C.

Kahney, Leander (2001) *Robots Scour WTC Wreckage*. Wired Magazine, http://www.wired.com/news/print/0,1294,46930,00.html

Larsson, Staffan, Robin Cooper, Stina Ericsson (2000) *System Description of GoDis*. Third Workshop in Human-Computer Conversation, Bellagio, Italy.

Lewin, I., Rupp, C., Hieronymus, J., Milward, D, Larsson S., and Berman, A. (2000) 'SIRIDUS Project Deliverable D6.1, System Architecture and Interface Report (Baseline), URL (May 2002): http://www.ling.gu.se/projekt/siridus/Publications/publications.html.

NASA (1997) *Past Missions – Mars Pathfinder*. NASA, http://www.jpl.nasa.gov/missions/past/marspathfinder.html

Rayner, M., B.A. Hockey, and F. James. (2000a) *Turning Speech into Scripts*. AAAI Spring Symposium on Natural Dialogues with Practical Robotic Devices.

Rayner, M., B.A. Hockey, and F. James (2000b) *A compact architecture for dialogue management based on scripts and meta-outputs*. ANLP.

TRINDI (2002) . http://www.ling.gu.se/projekt/trindi.