Dialogue Interaction with the DARPA Communicator Infrastructure: The Development of Useful Software

Samuel Bayer
The MITRE Corporation
202 Burlington Rd.
Bedford, MA 01730
sam@mitre.org

Christine Doran
The MITRE Corporation
202 Burlington Rd.
Bedford, MA 01730
cdoran@mitre.org

Bryan George
The MITRE Corporation
11493 Sunset Hills Rd.
Reston, VA 20190
bgeorge@mitre.org

ABSTRACT

To support engaging human users in robust, mixed-initiative speech dialogue interactions which reach beyond current capabilities in dialogue systems, the DARPA Communicator program [1] is funding the development of a distributed message-passing infrastructure for dialogue systems which all Communicator participants are using. In this presentation, we describe the features of and requirements for a genuinely useful software infrastructure for this purpose.

Keywords

Spoken dialogue, speech interfaces

1. INTRODUCTION

Over the last five years, three technological advances have cooperated to push speech-enabled dialogue systems back into the limelight: the availability of robust real-time speech recognition tools, the explosion of Internet-accessible information sources, and the proliferation of mobile information access devices such as cell phones. However, the systems being fielded, and the standards arising from these efforts, represent only a limited set of capabilities for robust voice-enabled interaction with knowledge sources. The most prominent indication of these limitations is the fact that these systems are overwhelmingly system-directed; the system asks a question, and the user responds. While this type of interactions sidesteps a number of problems in speech recognition and dialogue tracking, it is overwhelmingly likely that these restrictions are not manageable in the long term.

The DARPA Communicator program [1] is exploring how to engage human users in robust, mixed-initiative speech dialogue interactions which reach beyond current capabilities in dialogue systems. To support this exploration, the Communicator program has funded the development of a distributed message-passing infrastructure for dialogue systems which all Communicator participants are using. In this presentation, we describe the features of and requirements for a genuinely useful software infrastructure for this purpose.

2. BUILDING USEFUL SOFTWARE

The Galaxy Communicator software infrastructure (GCSI) is an elaboration and extension of MIT's Galaxy-II distributed infrastructure for dialogue interaction [3]. The fact that all program participants are required to use the GCSI imposes a somewhat more severe set of requirements on the infrastructure than usual, and these requirements range far beyond the straightforward considerations of functionality.

- **Flexibility**: the infrastructure should be flexible enough to encompass the range of interaction strategies that the various Communicator sites might experiment with
- **Obtainability**: the infrastructure should be easy to get and to install
- **Learnability**: the infrastructure should be easy to learn to use
- **Embeddability**: the infrastructure should be easy to embed into other software programs
- **Maintenance**: the infrastructure should be supported and maintained for the Communicator program
- **Leverage**: the infrastructure should support longer-term program and research goals for distributed dialogue systems

3. FLEXIBILITY

The GCSI is a distributed hub-and-spoke architecture based on message-passing. The hub of the GCSI incorporates a scripting mechanism that allows the programmer to take control of the message traffic by implementing "hub programs" in a simple scripting language. The benefits of this sort of infrastructure are considerable in the context of exploring different interaction and control strategies for dialogue. For example:

- Because the infrastructure is based on message-passing instead of APIs, there's no need for the hub to have any compile-time knowledge of the functional properties of the servers it communicates with (in contrast to, for instance, a CORBA infrastructure).

- Because the hub scripting allows the programmer to alter the flow of control of messages, it's possible to integrate servers with a variety of implicit interaction paradigms (e.g., synchronous vs. asynchronous) without modifying the servers themselves
- Because the hub scripting allows the programmer to alter the flow of control of messages, it's possible to insert simple tools and filters to convert data among formats without modifying the servers themselves.
- Because the hub scripting language fires rules based on aspects of the hub state, it's easy to write programs which modify the message flow of control in real time.

4. OBTAINABILITY

We believe that the simplest licensing and distribution model for software like the GCSI is an open source model. With the appropriate open source licensing properties, there are no barriers to freely distributing and redistributing the GCSI, or to distributing dialogue systems created using the GCSI, or to building commercial products based on it. The GCSI is distributed under a modified version of the MIT X Consortium license, and we are reasonably certain that the license simplifies all these tasks. In particular, two Communicator sites are planning to distribute their entire dialogue systems as open source, which would not be possible without appropriate licensing of the GCSI.

It's also important to address the level of complexity of installing the software once it's obtained. Research software is notoriously hard to install, and it's far more useful to ensure that the software can be used straightforwardly on a small number of common platforms and operating systems than to try to make it run on as many platforms as possible. We've targeted the three platforms which the program participants were developing on: Windows NT, Intel Linux, and Sparc Solaris. The GCSI is known to work or to have worked on other configurations (HP-UX and SGI IRIX, for instance), but these configurations are not supported in any meaningful way. The open source model can help here, too: if someone wants to port the infrastructure to a BSD OS, for instance, they have all the source (and will hopefully contribute their modifications to the open source code base).

5. LEARNABILITY

Once the software is installed, it's important to know where to start and how to proceed. We have offered a series of two-day intensive training courses on the Communicator infrastructure which have been attended by the majority of Communicator participants. In addition, the GCSI comes with extensive documentation and examples, including a toy end-to-end dialogue system example which illustrates one possible configuration of Communicator-compliant servers. Our goal is to ensure that it's possible to learn to use the Communicator infrastructure from the documentation alone, and at least two sites have succeeded in creating dialogue systems using the GCSI in a short period of time without attending our training course.

6. EMBEDDABILITY

The GCSI includes libraries and templates to create Communicator-compliant servers in C, Java, Python, and Allegro Common Lisp. However, it's not enough to provide a software library; this library has to be well-behaved in a number of ways. In particular, if the GCSI is to be used in conjunction with CORBA or various windowing systems, it must be possible to embed the GCSI server libraries into other main loops, and to control all the features of the GCSI without controlling the toplevel flow of control. To enable this goal, the GCSI is based on a straightforward event-based programming model, which is used to implement the default Communicator server main loop, as well as the implementation of the Python and Allegro server libraries. The GCSI is distributed with a number of examples illustrating this embedding.

7. MAINTENANCE

Finally, GCSI consumers must be able to rely on getting help when something goes wrong, and expect that design and implementation problems will be rectified and that desired complex behaviors will be supported. The importance of responsiveness and flexibility in maintenance is one of the reasons we prefer the GCSI for Communicator instead of a third-party tool such as SRI's Open Agent Architecture [2], which the Communicator program does not control the development of.

In addition to maintaining a bug queue for the GCSI, we have addressed successively more complicated infrastructure requirements in successive releases of the GCSI. For instance, in the most recent release (3.0), we addressed infrastructure support for asynchronous delegation strategies being explored by the Communicator effort at MIT and issues relating to consumption of audio input by multiple recognizers.

8. LEVERAGE

Ultimately, we hope that the GCSI, together with open-source servers such as recognizers, parsers, synthesizers and dialogue modules provided by application developers, will foster a vigorous explosion of work in speech-enabled dialogue systems. For example:

- The programming-language-independent nature of the GCSI message-passing paradigm allows the Communicator program to develop implementation-independent service standards for recognition, synthesis, and other better-understood resources.
- The freely available nature of the GCSI allows application developers to contribute dialogue system modules which are already configured to work with other components.
- The availability of an "environment" for dialogue system development will support the development of an open source "toolkit" of state-of-the art, freely available modules. A number of Communicator sites are already releasing such modules.
- A common infrastructure will contribute to the elaboration of "best practice" in dialogue system development.

There are certainly a number of emerging and existing alternatives to the GCSI for dialogue system development (SRI's Open Agent Architecture, for instance). However, we believe that the combination of a software package like the GCSI and the critical mass generated by its use in the DARPA Communicator program presents a unique opportunity for progress in this area.

The GCSI is available under an open source license at http://fofoca.mitre.org/download.

9. ACKNOWLEDGMENTS

This work was funded by the DARPA Communicator program under contract number DAAB07-99-C201.

10. REFERENCES

[1] http://www.darpa.mil/ito/research/com/index.html.

[2] D. L. Martin, A. J. Cheyer, and D. B. Moran. The open agent architecture: A framework for building distributed software systems. Applied Artificial Intelligence, vol. 13, pp. 91--128, January-March 1999.

[3] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue. Galaxy-II: A Reference Architecture for Conversational System Development. Proc. ICSLP 98, Sydney, Australia, November 1998.