A Distributed Prototyping Environment for Human Language Technology

Theban Stanley, Julie Baca, Miao Liu and Joseph Picone

Center for Advanced Vehicular Systems
Mississippi State University
{stanley, baca, liu, picone}@cavs.msstate.edu

Abstract – *The DARPA Communicator program has fuelled the design and development of impressive human language technology applications. Its distributed framework has offered numerous benefits to the research community, including reduced prototype development time, sharing of components across sites, and provision of a standard evaluation platform. It has also enabled development of client-server applications with complex inter-process communication between modules. However, this latter feature, though beneficial, introduces complexities which reduce overall system robustness to failure. In addition, the ability to handle multiple users and multiple applications from a common interface is not innately supported. In this paper, we describe our enhancements to the original Communicator architecture to address robustness issues and to support a multiple multi-user application capability. These enhancements have been evaluated using a series of experiments and they have shown a 7.2% improvement in the robustness of the system. These enhancements are available in our public domain toolkit.*

Keywords – *DARPA Communicator, multi-user/application environment, state machine architecture, handshaking.*

1. INTRODUCTION

Early human language technology systems were designed in a monolithic fashion. As these systems became more complex, this design became untenable. In its place, the concept of distributed processing evolved wherein the monolithic structure was decomposed into number of functional components that could interact through a common protocol [1]. This distributed framework was readily accepted by the research community and has been a cornerstone for the advancement in cutting edge human language technology prototype systems.

Once distributed systems became the widely accepted implementation of human language technologies, there arose a need for a common architecture that would support reusability and compatibility between the modules developed at different sites. Many projects were conducted with the goal of creating a common open source platform for HLT. Some of the notable projects included Advanced Language Engineering Platform (ALEP) [2], General Architecture for TEXT Engineering (GATE) [2] and TIPSTER project [2] started in the mid 1990s. Most of these projects were initiated to produce a common architecture for text based applications such as information retrieval, information extraction and automatic text summarization which would improve document processing efficiency and cost effectiveness.

The Communicator program was funded by DARPA for the purpose of creating open source architecture for spoken language applications. It was one of the first architectures to provide a conversational and multi-modal interface for human language technologies [2]. The Communicator architecture was designed using the MIT Galaxy II system [3]. The wide availability of Communicator compatible components, such as speech recognition and dialog management made it valuable to speech researchers. Its success is evident from the wide variety of applications that were developed using the Communicator architecture which include navigation systems [4], weather information systems [5] and travel planning systems [6].

Most of the above mentioned architectures have been predominantly research-oriented. Widespread commercialization of HLT has led to web-based technology platforms. Some of the successful technology includes Nuance's SpeechObjects [7] which is based on VoiceXML. Philip's SpeechMania [8] is an online architecture based on High-level Dialogue Definition Language (HDDL) which is Philip's special purpose programming language [8].

In our laboratory, we used the DARPA Communicator architecture to design and develop a human language system consisting of four applications, namely, the Speech analysis, Speech recognition, Speaker verification and Dialog system application. Figure 1 shows the intial Dialog system prototype. The plug-and-play capability of the Communicator architecture is well-known for reducing prototype development time by enabling sharing of components across sites, allowing research groups to specialize in specific technologies and share others. It also provides a standard platform for evaluation of systems developed by different laboratories. Within this platform, multiple servers communicate through a common protocol programmed in the "hub". Figure 1 illustrates the use of this hub and spoke architecture for our Dialog system. The servers include the speech recognition module [4], database and dialog management modules [4], developed in our lab and the natural language parser and generation modules [9] from the Center for Spoken Language Processing, University of Colorado.

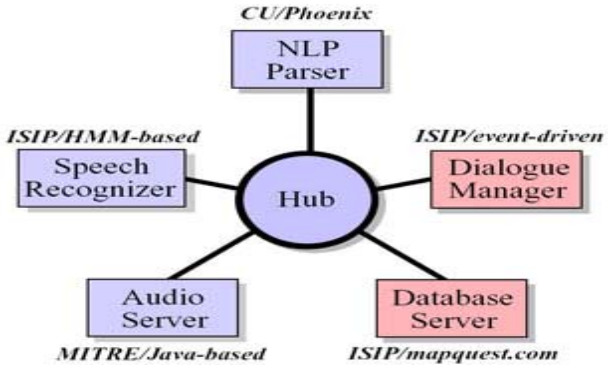

Fig. 1. An overview of DARPA Communicator-based dialog system architecture.

The features noted above proved invaluable in reducing our initial development time. However we also encountered certain vulnerabilities in the architecture during this phase and the need for additional capabilities in the subsequent expansion of our system to include multiple applications. This paper describes design enhancements made to the original Communicator architecture to address these needs, including automated support of multiple multi-user applications through a common interface, improvements on robustness to failure, and enhanced debugging. Finally, we present measurements of system performance improvements and plans for future development.

2. ORIGINAL ARCHITECTURE

During the initial design phase, we experienced communication deadlocks among servers and memory management issues that were difficult to debug. Basic logging mechanisms were provided to address some of these issues, but certain desirable features were not available, such as automated server startup, error-detection and correction. We anticipated such issues would grow in number and complexity as we added multiple multi-user applications.

As an example, the user interface for our system ran as a client program on a laptop with the computational servers running on a workstation. The original architecture serviced multiple users, but required manual server startup, including the port allocation to avoid port conflicts. Further, it required manual detection and correction of server errors by restarting them from the workstation. In either case, startup or error detection, the laptop and workstation may not be in close proximity. Clearly, one solution to the latter problem was to enhance the system robustness to failure. We describe our efforts to enhance this capability later in this section. However, no such solution will remove all errors and their potential grows as the number of applications and users increases. It is important, therefore, to also provide graceful error management. To address these issues, we developed a module to automate server startup as well as server error detection and correction.

Supporting multiple applications also required a common interface that allows the user to choose from the applications and coordinates inter-process communication with each application server and process. We designed and integrated this enhanced functionality with the server management module as well.

With respect to robustness, the Communicator architecture provides a basic structure called a "frame" for communication among servers and processes [10]. This structure implicitly allows a strict "handshaking" protocol, but does not require or provide an implementation of such a protocol. We found that implementing and enforcing such a protocol became critical for system robustness as the number and complexity of our applications grew. We also developed debugging tools with corresponding diagnostics and visual displays specific to this protocol.

3. ARCHITECTURAL ENHANCEMENTS

Our first and most critical need concerned automating server startup, error detection and correction. Secondly, we required a common interface to allow users to select among applications. In addition, the need for robustness to error and improved debugging capabilities were heightened with multiple applications.

3.1. Automated Server Management

Automated server management became critical with the addition of multiple applications. Though the Communicator process monitor provides an interface to start and terminate servers, it requires manual monitoring. To address this issue, we designed the Process Manager module that automatically starts and controls all server processes in the prototype system architecture. Figure 2 shows an overview of the multi user architecture for multiple applications.

When a user starts a new application, the client program requests the Process Manager to start the respective servers and the hub. The Process Manager performs this startup task by invoking a Java Process Object. The Java Process Object enables the Process Manager module to control all server

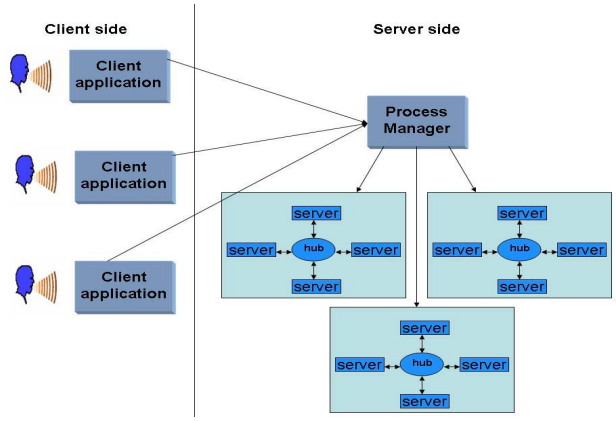

Fig. 2. The Process Manager module controls multiple applications and servers.

processes. The Process Manager module can create a process, wait on a process, perform input/output on the process and even check the exit status of the process. If a server process fails for any reason, the Process Manager detects the failure and sends a message to the client side forcing the user to restart the demo. In a multi-user environment, port allocation also needs special attention. The Process Manager allocates port numbers and ensures no two servers are assigned the same port.

3.2. Common Application Interface

Support for multiple applications required providing a common interface from which users could select an application of interest. We designed our Demo Selector module to provide the desired interface and coordinate with the Process Manager module to start the required servers.

The Demo Selector interface displays a single screen with icons for each of the four applications. Once the user selects an application, the Demo Selector loads and displays the user interface needed for the specific application. Figure 3 shows the Demo Selector interface for the four applications, superimposed with the user interface for the Speech Analysis application, after it has been selected. The client program sends a Communicator frame with a key-value pair containing the name of the application that was selected. Upon receiving the message in this frame, the Process Manager starts the required servers. The Demo Selector also has a network configuration menu as referenced in Figure 3 that allows the user to set the IP address of the server machine and port through which the client program communicates with the process manager.

3.3. Improvements on System Robustness

Improving system robustness to failure was a primary focus of our enhancements. As the foundation of our redesign strategy, we targeted a simple application, Speech Analysis. Our approach entailed using the implicit capabilities of the Communicator to enhance reliability of inter-process communication between clients and servers. This section describes how we implemented a state machine architecture to support a basic handshaking protocol between the client and servers using frames.

Figure 4 shows the state machine architecture and basic handshaking supported between the Speech Analysis client and the Signal Detector server. We used a simple handshaking protocol with signals and acknowledgements, each implemented as Communicator frames sent via the hub. The states and handshaking protocol support three major interaction phases between client and server, 1) preparing for data transfer; 2) data transfer it self, and 3) end of data transfer. For phase 1, the client begins in the Initialization state, during which it establishes connection with the hub. It then transitions to the Audio_Ready state and sends an audio_ready signal to the Signal Detector server to prepare it

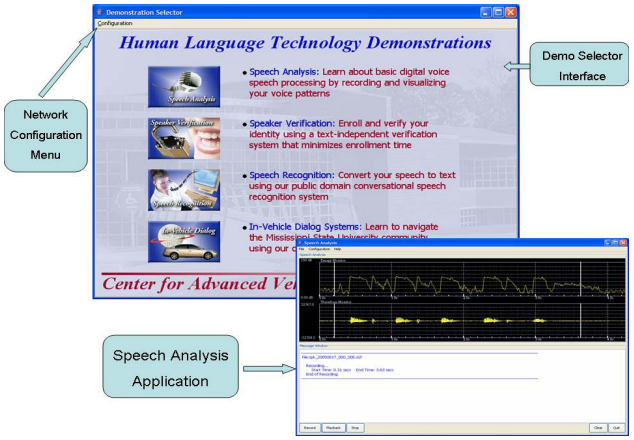

Fig. 3. Demo Selector and Speech Analysis user interface

for audio data transfer. The client then waits for an acknowledgement of the audio_ready signal from the Signal Detector server, and once received, it transitions to the Audio_Ready_Ack state.

Phase 2, data transfer, begins when the client then transitions to the Data_Transfer state and sends packets of audio data in Communicator frames to the server. For each frame of data sent, the client waits for an acknowledgement from the server, which checks each for validity. If the server receives a frame that is invalid, it does not send an acknowledgement signal, but generates an error message, written to a log file. The client will not send further data until it receives an acknowledgement. If data transfer completes successfully, the Signal Detector server detects endpoints and passes the endpointed data to the client. The client then sends an end of utterance signal to the Signal Detector server and waits for an acknowledgement. On receiving the end-of-utterance signal, the Signal Detector server sends an acknowledgement signal to the client and resets itself to the initial state. The handshaking protocol described in this example is implemented for all applications and has eliminated server failures and deadlocks due to

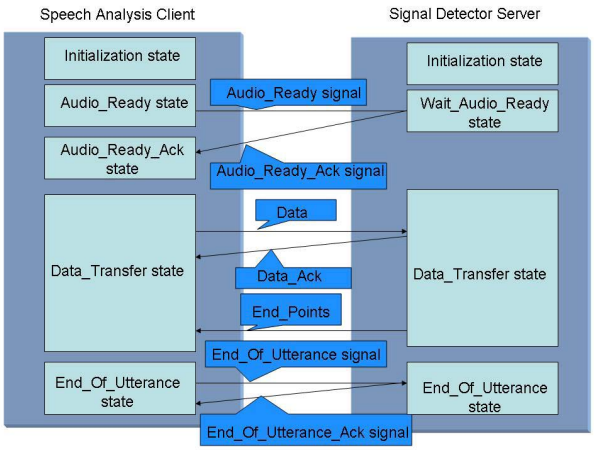

Fig. 4. Handshaking between the Speech Analysis client and the Signal Detector server

communication errors.

4. PERFORMANCE IMPROVEMENTS

The following section discusses performance improvements achieved by analyzing the enhancements quantitatively and qualitatively.

4.1. Quantitative Analysis

A series of experiments are discussed which provide the necessary data in measuring the robustness improvements achieved due to the enhancements.

4.1.1. Experiment 1. Table 1shows the performance data for 386 queries spanning through five different query types. This data was gathered early in our development efforts, prior to our enhancements. Out of the 386 queries, 96.92% "passed" while 3 % "failed" due to a server error or a deadlock. These 386 queries were re-run after the architectural enhancements, and we were successfully able to reduce these failures related to the robustness of the system. One limitation of the experiment is that it tested the system against baselines established using the text mode i.e., Natural Language Processing modules. Though necessary to test against these established baselines first, these are not sufficient results to fully measure overall robustness improvements.

4.1.2. Experiment 2. The second experiment consists of one non-native, male speaker, who had prior experience using the system, performing a set of tasks which were randomly selected from a task pool. In this experiment, a task consisted of one or more interaction of the user with the system. An example of a task is "Use speech mode in the Dialog system to query the distance between two places". The pool had a total of 38 tasks out of which 8 tasks were hypothesized to fail for the original architecture under certain system specifications. The other 30 tasks were hypothesized to be successfully completed in both the architectures and covered a wide variety of usage categories as shown in Figure 5. A random generator was used to pick 30 trials from the task pool and the user performed these trails. In this experiment,

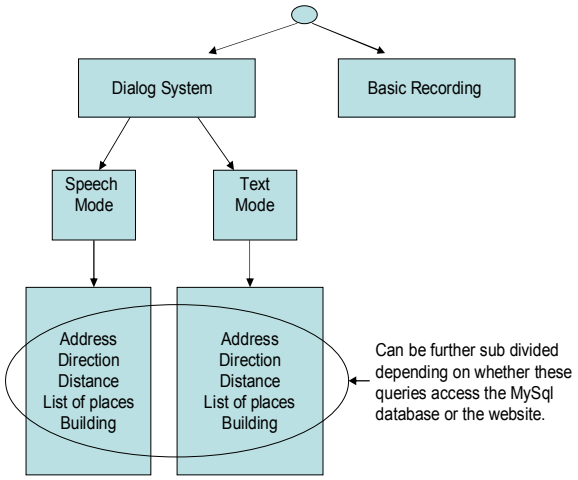

Fig. 5. Categorization tree of the scenarios.

all the 30 trials were successfully completed under the enhanced architecture and only 24 trials passed the original architecture. This experiment confirmed that these tasks did fail in the original architecture which was not tested before. Though a random number generator removes a level of bias, it is not based on observed system usage, and, as such, does not necessarily capture typical usage patterns.

4.1.3. Experiment 3. In this experiment, five, first-time users performed tasks pertaining to 24 usage scenarios. A scenario is a general situation under which the user is asked to use the system. A scenario may require performing one or more tasks. For example, consider a user is planning a vacation to the city of her choice. She needs to decide on a travel itinerary by using the system. This is a general scenario and the user may choose to accomplish this by performing several tasks using the system. Each task may require a single query or multiple queries to accomplish. This is referred to as an interaction which is defined as one response from the system to accomplish a specific task. Among the five users, there were three males and two females. These included one native speaker and four non-native speakers. The usage scenarios required performing such tasks as recording for varying time

Table 1. Performance data for a dialog application

		Before Enhancements			After Enhancements		
Queries	# of utterances	Passed (%)	Failed (%)		Passed (%)	Failed (%)	
			Server Errors	Deadlocks		Server Errors	Deadlocks
Address	98	100	0.00	0.00	100.00	0.00	0.00
Direction	219	95.43	2.28	2.28	100.00	0.00	0.00
Distance	23	91.31	8.70	0.00	100.00	0.00	0.00
List of places	36	100	0.00	0.00	100.00	0.00	0.00
Building	10	100	0.00	0.00	100.00	0.00	0.00
TOTAL	386	96.92	1.80	1.29	100.00	0.00	0.00

Table 2. Three categories of experimental data.

Category	Number of interactions successfully attempted by User 1		Number of interactions successfully attempted by User 2		Number of interactions successfully attempted by User 3		Number of interactions successfully attempted by User 4		Number of interactions successfully attempted by User 5	
	Enhanced	Original	Enhanced	Original	Enhanced	Original	Enhanced	Original	Enhanced	Original
Basic recording	9	8	8	6	7	6	12	8	9	8
Dialog system: Speech mode	8	7	9	9	9	4	12	3	9	0
Dialog system: Text mode	5	8	9	9	7	0	8	0	8	0
Total number of interactions	**22**	**23**	**26**	**24**	**23**	**10**	**32**	**11**	**26**	**8**
The number of interactions that passed the enhanced architecture: 129					**The number of interactions that passed the original architecture: 76**					

durations and querying for information. The scenarios were carefully drafted not to prompt the user for a specific query. Before participating in the experiment, each user was presented with a set of instructions and was allowed to take a 10-minute practice session to get familiar with the functionality of the system. Once the practice session was over, the user engaged in the usage scenarios for the experiment, using both architectures, with a time limit of 30 minutes for each. The user performed the tasks, first on the enhanced architecture followed by the original architecture to prevent any robustness improvement trend that may occur due to the user's familiarity with the system. The user was asked to cease testing if there was a system failure or he/she exceeded the allotted time of 30 minutes.

As shown in Table 2, a total of 129 interactions were successfully completed using the enhanced architecture and only 76 interactions could be performed successfully using the original architecture. Table 2 illustrates that "Dialog system: Speech mode" category shows an evident improvement in robustness compared to the other two categories. This is expected, as the "Dialog system: Speech mode" is the most complex task in the HLT system. Inference cannot be made from the data obtained from this experiment as they may overstate the actual improvement in robustness because the user was asked to abort the experiment following a system failure which prevented her from performing the subsequent tasks. To obtain a more focused measure of robustness improvement, further experimentation was needed to target the "Dialog system: Speech mode" category.

4.1.4. Experiment 4. This experiment was designed to target the "Dialog system: Speech mode" category to measure the robustness improvement on the most complex tasks in the HLT system. The limitations of Experiment 3 were addressed by asking the user to request a system restart in the event of a system failure and to continue testing for the full 30 minutes.

Five users were asked to perform 9 usage scenarios restricted to the "Dialog system: Speech mode" category. Among the five users, there were four males and one female. The user pool consisted of one native speaker and four non-native speakers. Each user was provided with a series of scenarios originating from the user's visit to Starkville from her city of residence. The users were initially presented with a set of instructions and were allowed to take a 10-minute practice session to get familiar with the functionality of the system. The user performed tasks from 9 different scenarios with a maximum time duration of 30 minutes per session on each architecture.

As shown in Table 3, 55 interactions were completed successfully using the enhanced architecture while only 51 interactions were completed successfully using the original architecture. The results show a 7.2% improvement in the robustness on the most complex set of tasks defined for the HLT system.

4.2 Qualitative Analysis

Most of the enhancements to the DARPA architecture did not exist in the original architecture and were developed out

Users	Number of interactions that passed successfully the Enhanced architecture	Number of interactions that passed successfully the Original architecture
User 1	10	**8**
User 2	10	11
User 3	**9***	9
User 4	**16***	**13**
User 5	10	**10**
Total number of interactions that passed successfully	55	51
* indicates that a server error was experienced but the enhancements prevented a system failure.		

Table 3. The number of interactions that passed the original and the enhanced architecture in experiment 4.

of necessity for better error handling and debugging capabilities. The newer modules that have been built into the enhanced architecture have added powerful capabilities which were not innately supported by the original architecture. The two main qualitative enhancements are the Process Manager module and better debugging capabilities. The Process Manager module provides automated server management and built-in capability to handle multi user applications. The multi user and multiple application capability were not available in the original architecture.

Therefore, these two enhancements cannot be evaluated against a baseline. Nonetheless, these capabilities clearly extend the complexity of applications that can be deployed, and thereby, the fundamental research issues that can be investigated using this architecture. The enhanced architecture has provided better logging of communication which along with the State machine architecture and basic handshaking capabilities has provided a more efficient debugging paradigm for the system. The enhancements related to better debugging capabilities were achieved through rigorous design meetings and reviews. The debug window which is a part of the user interface provides an excellent tool for the user to debug the system when she has no access to the log files on the server side. The user interface was designed by a team that included experts in human-computer interaction and graphic design.

Further, both the debugging and user interface enhancements were reviewed and evaluated by two categories of users respectively, 1) software developers programming this technology and 2) principle investigators who presented these technologies to research sponsors. Figure 6 illustrates the use of the debug window. In this particular case, the client program had not received an acknowledgement from the server for the data frame it sent. The user can use the debug window and browse through the Communicator messages to reconstruct the exact scenario that led to the failure. Thus the debug window provides a better debugging interface for the user which never existed in the original architecture.

Fig. 6. A debug window showing an audio data transfer error

5. CONCLUSIONS

The DARPA Communicator architecture significantly advanced human language technology and, has played a critical role in the design and development of human language technology applications in our laboratory. In developing these applications, we have addressed vulnerabilities in this architecture through several important enhancements, including automated server startup, error detection and correction, support for multiple multi-user applications, increased system robustness to failure, and improved debugging capabilities. Quantitative analysis has shown a 7.2% improvement in the robustness of the system.

Further experimentation includes allowing the system to respond to user queries continuously for prolonged time periods under carefully controlled conditions so that meaningful data are collected from these experiments. The experimental design reported here will serve as basis for prolonged studies. We also plan to enhance the Process Manager to create and manage server processes on different host machines to increase the computational power available for applications. This capability will enable us to run applications at significantly greater speed on our supercomputer clusters.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. IIS-0414450. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

REFERENCES

[1] K. Hacioglu and B. Pellom., "A Distributed Architecture for Robust Automatic Speech Recognition," Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, pp. 1234-1234, Hong Kong, April 2003.

[2] F. Olsson, ``A requirement analysis for an open set of human language technology tasks." In Proceedings of Workshop on Portability Issues in Human Language Technologies, Las Palmas, Spain, June.

[3] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue, "Galaxy-II: A reference architecture for conversational system development," in these Proceedings, Sydney, Australia, 1998.

[4] J. Baca, J. Zheng, H. Gao, and J. Picone, "Dialog Systems for Automotive Environments," in Proc. European Conf. on Speech Comm. and Tech., Geneva, Switzerland, pp. 1929-1932, Sep. 2003.

[5] V. Zue, et al, "JUPITER: A telephone-based conversational interface for weather information," IEEE Trans. on Speech and Audio Processing, vol. 8, no. 1, January 2000.

[6] A. Rudnicky, et al., "Creating natural dialogs in the Carnegie Mellon Communicator system," Proc. Eurospeech, pp. 1531--1534, 1999.

[7] "Nuance SpeechObjects," http://www.w3.org/TR/ speechobjects/.

[8] "Philips SpeechMania," http://www.nist.gov/speech/publications/darpa98/html /ww20/ww20.htm.

[9] Ward, W. and Pellom, B., "The CU Communicator System," in Proc. IEEE Automatic Speech Recognition and Understanding Workshop, Keystone, Colorado, USA. pp. 1234-1234, December 1999.

[10] "Galaxy Communicator," SourceForge, 2003 http://sourceforge.net/projects/communicator.