A Distributed Spoken User Interface Based on Open Agent Architecture (OAA)

Ying Cheng, Anurag Gupta and Raymond Lee
Motorola Labs, Motorola Australian Research Center, Sydney Australia
(ycheng, agupta, rlee)@arc.corp.mot.com

Abstract
The paper describes a spoken user interface that can be distributed over a network and that allows the user to have voice access to various functions such as user authentication, command controls in web browser and personal or public information retrieval. Integration of the user interface system is accomplished by using the Open Agent Architecture. A detailed description of the distributed speech recognizer used in the interface is also provided.

1 Introduction
In the course of the spoken language processing research, different components (e.g. speech recognizers, Text-To-Speech and so on so forth) have been developed and it is often the case that these components need to be quickly integrated to build a spoken dialog systems for a number of applications. It is imperative, in such a situation, to have an architecture that allows us to easily integrate and configure different components into spoken dialog systems of various applications.

There is an added complexity brought on by the fact that the various components can be developed independently by different research groups, coded in different programming languages and targeted at different platforms. Furthermore, as the computing and communication world becomes more networked there is the possibility that the required components will be available and distributed on a number of machines. Thus, in addition to the issues mentioned previously, we also face problems associated with distributed systems.

The MIT Galaxy architecture [4] addresses some of the above issues. In this paper, we discuss our experiences in using the Open Agent Architecture (OAA) [3] to integrate a spoken dialog user interface that comprises a number of components coded in different programming languages and distributed over a network. The spoken user interface is designed to perform secure voice access, voice command/control and voice-based information retrieval.

With the distributed system in mind, we have also implemented a client/server based speech recognizer comprising, (i) a client that performs feature extraction on the input speech signal and (ii) a server that decodes the feature sequences generated from the client into the recognized word sequences. The server and clients can run on different machines. The details of the recognizer will be discussed in Section 3 of the paper.

The rest of the paper is organized as follows. In the Section 2, the motivation for the system design and a description of the system are given. The details of the distributed speech recognition components are discussed in Section 3. An analysis of the experiences gained during the design of the system is provided in Section 4. In Section 5 a short conclusion and possible extensions to the work is briefly described.

2 System Description

2.1 Problems and Motivations
Over the years, we have built a number of spoken language processing components independently. We often endure time-consuming and painful experiences in integrating the existing components in order to build proof-of-concept demonstrations.

By using existing resources, such as small speaker dependent recognizer, speaker verification, and distributed speaker independent recognizer, we want to build a spoken user interface that provides various services seamlessly. For instance, the voice authentication should be automatically fired when one attempts to access personal information.

Our motivations for the work presented in this paper include:

- building a spoken user interface that provides seamless integrated services;
• building a distributed speech recognizer that can exploit the advantages of a networked world;
• experimentation with the existing architectures with the aim of discovering a solution to the issues described earlier.

In our experiments, we chose OAA as the core of our architecture.

2.2 Brief Description of OAA

There are three main elements in an OAA:
1. Facilitator agents that serve as controller, communication router and knowledge base server.
2. Application agents that register with a facilitator agent to provide and/or request services.
3. The Inter-agent Communication Language (ICL) that is shared by all agents regardless of what programming language, platforms and machines a given agent is associated with.

It is noteworthy that the requests from application agents are expressed in terms of services to be required and not in terms of how to perform or what module is to perform the service. This feature enables dynamic resource allocation over a dynamic network where new programs are launched and old ones are terminated dynamically.

2.3 System Architecture

Figure 1 illustrates the architecture of the spoken user interface system based on OAA.

The system comprises an OAA facilitator and six autonomous agents that can perform different functions. The services provided by each agent will be discussed in the next section.

2.4 Functions of the User Interface

The user interface provides first of all, voice access to personal stock portfolio. Since this is a personal and confidential information, a voice password has to be set up first. We developed a speaker verification engine to train the on-user-defined password and then to perform the voice authentication when a user starts a stock portfolio session. A speaker dependent speech recognizer of small footprint is used to recognize queries for stock information. In a real application, this may reside on individual wireless handheld devices. Currently the responses from machine are communicated to the user via speech that is generated by a TTS engine developed in Motorola Labs.

The user interface also allows voice access to public information that does not require speaker authentication. For instance, one can speak to browse web sites or to access work phone numbers of colleagues or to place a phone call. A distributed speech recognizer is used to perform these tasks and machine responses (except the web display) are communicated via voice.

2.5 Agents

An application agent consists of two parts. One part performs the autonomous task and the other communicates with the facilitator. The libraries provided by SRI can be used to build the latter part on any existing components in C, C++, Java or some other programming languages.

In the following, we briefly describe the functions of each application agent.

Facilitator agent: Other application agents register themselves with the facilitator. The facilitator receives requests from application agents and delegates them to appropriate agents and then pass back the results to the agents that originally issue the request. If there are more than one agent that provides the same service, a competition scheme can be set up such that only one of the results is chosen.

Small vocabulary recognizer agent: This is a speaker dependent recognizer with small footprint that can be potentially implemented into devices with limited computation power, for instance handheld devices. The agent provides the following services:

![Architecture of Spoken User Interface](image-url)
• Client Connection to provide recognition agent details i.e. IP addresses of the agent, and the port where speech data could be passed.
• Training function: to be used during a service set up.
• Recognition: When a recognition request message arrives, the agent performs the recognition task on the speech obtained at the specified port.

Speaker verification agent: The speaker verification engine agent creates an instance of the engine and then provides training and speaker verification capabilities. For training, three utterances of the user-defined password and the User ID are required. If successful, a model of the user is created and stored. During verification, the audio data along with the User ID is passed to the agent and it then returns a positive or negative verification. The various functionality provided by the agent are:

- **Client Connection**: Providing recognition agent details i.e. IP addresses of the agent, and the port where speech data could be passed, as well as an identity number to the client
- **Training**: Takes in speech data at the specified port and a User ID and trains a model. It returns whether it was successful in creating the client model.
- **Verification**: Takes in speech data at the specified port and a User ID and verifies using the stored model. It then returns the result of verification.
- **Client Disconnection**: General management routine to keep track of multiple clients.

Motorola TTS agent: This agent creates an instance of the SAPI-compliant Motorola TTS Engine and registers itself with the facilitator. Upon the arrival of the message from the facilitator, it parses out the prompt text from the message and produces speech output on either a sound card or over a telephone channel depending upon the configuration of the system.

Distributed ASR server agent: It takes the speech feature input from the client and outputs the recognized string back to client.

Distributed ASR client: When receiving a request from the facilitator, it takes the audio input from speech input channel (either microphone or telephone, based on the system configuration). Then it computes speech features and passes them to recognition server that sends back the results.

Dialog Manager agent: This is a state-based dialog manager. Dialog is carried out as the active dialog states fire requests to the facilitator, process the responses and then either fire up next dialog state or claim the end of a dialog session.

The six application agents are implemented respectively in C, C++ and Java.

## 3 Distributed Speech Recognition Components

The speech recognition component is implemented in the Java programming language as it currently seems to be the only language that provides integral support for distributed applications over a network. We adopted a typical client/server architecture as shown in Figure 2. The speech recognizer comprises of two major modules:

1. A client that performs feature extraction on the input speech signal and;
2. A server that decodes the feature sequences generated from the client into the recognized word sequence.

The network communication between the client and the server is performed by the remote method invocation (RMI) mechanism supported in Java.

![Figure 2: Client/Server architecture for the speech recognition component.](image)

### 3.1 Speech Recognition Client

The speech recognition client accepts speech signal as input and converts it into the front-end features. The speech signal can be streamed from either a microphone on a computer or from the telephone. Twelve (12) MFCCs [2] and an energy term are extracted for each frame of data and transmitted to the server.
3.2 Speech Recognition Server

The server is a speaker independent decoder based on tied-state multiple mixture continuous density Hidden Markov Model (HMM). A time-synchronous one-pass search method [5] is employed. Finite state networks (FSN) are used to provide the grammar constraints of the decoder. A user-specified FSN is first compiled into a word-graph. The word arcs are then replaced with the pronunciations of words, to generate the search network. Both absolute pruning and rank based pruning strategies [1] are implemented to improve the speed performance.

The recognition result can be the recognized word sequence which are transmitted back directly to the client, or it can be the output of further post-processing of the words. For example, the result can be the data extracted from a database situated on the network based on the recognized words.

The acoustic models used in the system described here are trained from the Macrophone database; only the monophones are used. The front-end features consists of 12 MFCCs and the energy term together with their first and second derivatives, giving a total of 39 features. Note the first and second derivatives of the MFCCs are generated at the server using the parameters it received from the client. Each monophone model has a left-right HMM topology with no state skip. Ten (10) Gaussian mixtures are used for each HMM state.

4 Discussions

From our experience of building the above system, we found OAA provides a reasonable architecture that facilitates the integration of the existing components. It is very convenient to add a hook to any existing component to become an OAA agent. Also, a particular component can be easily replaced by another one without affecting the functionality of the integrated system as long as the new component performs the same function as previously registered with the facilitator. We have also experimented with distributed computing by launching agents on different machines. This does not pose any problems unless large amount of data exchange is required between agents on different machines.

Dynamic resource allocation is certainly a very good feature of OAA. However, there are no standard semantics that defines unique mappings between a solvable (i.e. a service that an agent provides) and what the solvable performs. Thus, the agent that requests the service and the one that provides the service need to have a common pre-defined semantics. We believe this shortcoming greatly constrains the scope of dynamic resource allocation.

5 Conclusion

In this paper, we have discussed our experiences in the quest for a solution to flexible component configuration and distributed computing. We mainly focused on speech modality as input and output. In the future, we will extend our research to cover multi-modal input/output. Also, we will study more closely, the issues involved in developing a seamless user interface when switching between different services and modalities.

We note that our efforts have not fully explored the capabilities provided by OAA and suspect that as the system becomes more complex and more components are involved an extensive use of OAA will be required.

Given the dynamics of real network traffic we will further explore the potential of our distributed speech recognizer in various networking conditions.

Reference
