2016 Conference of The Oriental Chapter of International Committee
for Coordination and Standardization of Speech Databases and Assessment Technique (O-COCOSDA)
26-28 October 2016, Bali, Indonesia

# Spoken Dialog System Framework Supporting Multiple Concurrent Sessions

Muhammad Qasim, Sarmad Hussain
Centre for Language Engineering, Al-Khwarizmi Institute of Computer Sciences
UET, Lahore, Pakistan
firstname.lastname@kics.edu.pk

Tania Habib
Computer Science and Engineering Department
UET, Lahore, Pakistan
Tania.habib@uet.edu.pk

Shafiq Ur Rahman
Department of Information Technology
UCP, Lahore, Pakistan
shafiq.rahman@ucp.edu.pk

***Abstract*— Spoken dialog systems are becoming increasingly popular to provide information to people. Different frameworks are available that provide the necessary infrastructure to develop a dialog system for any domain and language. The support for multiple concurrent sessions is a major requirement of real-world dialog systems. This paper proposes a framework that supports multiple concurrent sessions for developing dialog systems. The framework uses Galaxy's hub and spoke architecture for communication and Olympus' RavenClaw dialog manager to make use of its domain independent dialog engine. The paper describes the design of a Session Manager module that handles multiple concurrent sessions. The proposed framework has been implemented and tested on a real-world spoken dialog system which can handle ten concurrent sessions. The proposed framework performs well for multiple sessions and may be used to develop commercial scale dialog systems.**

***Keywords—spoken dialog system; multiple concurrent sessions; galaxy; ravenclaw dialog manager***

## I. INTRODUCTION

Information technologies are helpful in providing people the access to information but most of these technologies are only available to literate and people with access to internet. This is especially the case for developing countries where a significant proportion of population is illiterate or has no internet facilities. Thus, telephone based spoken dialog systems are feasible choice to provide information access to people in their local language.

Spoken dialog system provides means of interaction between humans and computers through speech interface. Spoken dialog systems are used to assist, help and provide desired information to the users using a spoken language. The applications of spoken dialog systems range from basic tasks like weather information and travel reservation to advanced systems like SIRI by Apple Inc. and Google Now by Google.

In order to develop a spoken dialog system, multiple components are required such as speech recognition, spoken language understanding and natural language generation. Before the development of these systems, a framework to manage and provide communication protocols for all components is required to be designed and developed. In this paper, we propose a framework for spoken dialog system which provides support for multiple concurrent sessions.

The rest of the paper is structured as follows: Section 2 describes different types of spoken dialog system and architectures being used. Section 3 presents our proposed framework for spoken dialog systems. Section 4 describes the integration of this framework with a telephone channel to develop a phone based spoken dialog system. Section 5 discusses a spoken dialog system based on our proposed framework. Finally, Section 6 concludes our discussion.

## II. LITERATURE REVIEW

Spoken dialog systems can be categorized into finite-state, frame based and agent based systems depending on their dialog control strategy. The finite-state systems take user through a number of pre-determined states limiting vocabulary size by restricting the user response. The frame based system ask questions from the user and the system extracts desired information from the user's response to perform a task such as flight inquiry but dialog flow is not pre-determined rather it depends on the user input. An agent based system allows advanced form of communication between system and user. These systems are mixed initiative where users can take control of the dialog flow.

One of the earliest spoken dialog system developed was ESPIRIT SUNDIAL [1], a finite-state system, which was designed to provide spoken language interfaces to computer databases through telephone. The architecture of the SUNDIAL project is that of a distributed database which targets to provide a close coupling among modules. Jupiter [2] is a frame based system that provides weather information. Mercury [3], Intelligent Tutoring Spoken dialogue system (ITSPOKE) [4] and Air Travel Information Service [5] are examples of agent based systems.

Different architectures have been used to develop spoken dialog systems. Open Agent Architecture (OAA) [6] is a distributed computing model that provides support for building applications requiring multiple software agents. The communication between agents is achieved using Inter-agent Communication Language (ICL). An agent is an ICL complaint software process that provides its services to be used by other

978-1-5090-3516-8/16/$31.00 ©2016 IEEE

agents. The agents may reside on a single system or distributed in a networked system. OAA is an extensible framework where individual agents can be added or replaced at runtime. Many systems have been built using OOA including CommandTalk [7] is a speech interface that provides commanders the ability to control simulated forces using spoken English commands. It is used at Marine Corps Air Ground Combat Center Twenty nine Palms, California, USA. Air Travel Information Service (ATIS) [5] is another OOA based spoken dialog system for accessing flight schedule information.

MCUBE is another architecture used in spoken dialog systems that enables message passing either from one agent to another or from one agent to multiple agents. MCUBE is a Java based framework where messages are encoded in XML. Multi-modal Access To City Help (MATCH) [8] is built using MCUBE framework to access restaurants information in New York city.

Intelligent Speech for Information Systems (ISIS) [9] uses Common Object Request Broker Architecture (COBRA) and Knowledge Query and Manipulation Language (KQML). COBRA provides support for communication between softwares written in different programming languages running on multiple systems through Interface Definition Language (IDL). KQML is a message passing protocol that provides support for knowledge sharing and information exchange among agents.

Galaxy [10] framework developed by MIT is most widely used in spoken dialog systems. It is a hub and spike model where a central hub controls all the other components and most of the information exchange occurs though the hub. Galaxy Communicator is an optimized and extended form of Galaxy framework. It's open-source, scalable and plug-and-play nature makes it very feasible for developing spoken dialog systems. A script based dialog management is available in MIT Galaxy architecture which has a lot of limitations like difficulty in navigation in case of complex tasks. Galaxy Communicator uses an agenda based dialog management strategy which uses two data structures: a dynamic product and an agenda [11]. The product consists of handlers which encapsulates processing related to a dialog level while agenda contains task related topics. The order of handlers governs the association of a user input with a product node.

Olympus [12] is a Galaxy Communicator based spoken dialog framework developed at Carnegie Mellon University (CMU). The major focus in the design of Olympus was to develop an open, transparent, modular and flexible architecture. Let's Go! Bus Information System [13], Language-Based Retrieval of Repair Information (LARRI) [14] for maintenance and repair activities of aircraft mechanics and TeamTalk [15] are some of the systems developed using Olympus framework. RavenClaw dialog manager [16], an improved version of Agenda dialog manager, is employed for dialog management in Olympus. RavenClaw is a task-independent dialog engine which means that core functionalities like error handling, timing and turn-taking are reusable. This decoupling between the task-dependent and task-independent components reduces the effort of developing dialog manager for a new domain because of reusability. Currently, Olympus framework does not handle multiple sessions which is requirement of most of the real world applications.

Olympus Peer to Peer (P2P) [17] deals with handling multiple sessions by Olympus. The key idea is to have multiple instances of Olympus running for each session or client while some of the resources like language model and grammar is shared between users for knowledge update. There are Olympus Peer nodes which can run certain number of concurrent Olympus instances and there are Olympus Super Peers which have many Olympus Peers connected to them. When a user or client wants to start a new session, it contacts the Olympus Peer and it initiates the new session. If that Olympus Peer have run of its limit, it contacts the Olympus Super Peer and then re-directs the client to some other Olympus Peer which has free sessions available. As only few resources are shared and most of the components have to run multiple instances to support multiple sessions, this may not be feasible for many applications.

The study [18] deals with multi-party interaction rather than standalone multiple sessions. In such scenario, system has to interact with multiple users simultaneously and keep track of the dialog going on with each user. This is done by having a separate dialog manager running for each user. It also introduces a new module, conversation manager, which controls which dialog manager will have floor, i.e. which user will be conversed with, at a certain time and it assigns high priority to certain sessions.

We propose a framework for spoken dialog systems that extends the features of Galaxy Communicator by using RavenClaw as its dialog manager and supports multiple concurrent sessions. Our proposed framework follows similar philosophy as in [18] but in different scenario as our end goal is to have multiple concurrent independent sessions with each having equal priority and same average response time.

## III. Proposed Framework

We have selected Galaxy Communicator framework as our base system and combined it with Olympus' RavenClaw dialog manager. As discussed earlier, Galaxy Communicator supports multiple concurrent sessions through the use of session identifiers. The dialog manager of Galaxy Communicator is not efficient as whole dialog manager has to be reengineered for a new task. But RavenClaw requires less effort to develop a dialog manager for a new task due to its decoupling of task-dependent and task-independent functionalities. Therefore, we decided to get the best of both by combining Galaxy Communicator and RavenClaw to utilize the scalability of Galaxy Communicator and easy to develop nature of RavenClaw dialog manager. In order to integrate RavenClaw into Galaxy Communicator, an interface is required to allow communication between Galaxy hub and RavenClaw. The following subsections discuss brief introduction of RavenClaw dialog manager highlighting the lack of multiple sessions handling in RavenClaw and design of Session Manager to handle multiple sessions.

### A. *RavenClaw dialog manager*

RavenClaw is a task-oriented dialog manager for managing the flow of dialogues in the spoken dialog systems. It isolates the domain-independent and domain-specific parts of the system and the developer only needs to be concerned with the domain specific part. This is accomplished by having a two-tier hierarchy, with the Dialog Task Specification as the upper layer and the Dialog Engine as the lower layer.

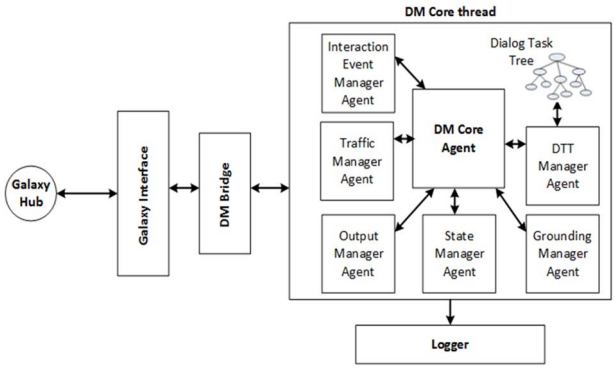


Figure 1: RavenClaw dialog manager.

Figure 1 shows architecture diagram of RavenClaw dialog manager. The diagram shows the four major components of the RavenClaw; Dialog Management (DM) Core thread, DM Bridge, Galaxy Interface and Logger. Each of the 4 components are spawned as separate threads. The Galaxy Interface provides the routines for interacting RavenClaw with the other Galaxy modules whereas DM Bridge acts as a mediator between the Galaxy Interface thread and the DM Core thread. Each incoming frame from the Galaxy hub is passed to DM Bridge which then passes it to DM Core thread and vice versa for the opposite direction.

At the start of a dialog session, the DM Bridge component creates the DM Core thread and initializes all the sub-components of the DM Core. Likewise, when the dialog session ends, the DM Bridge terminates the DM Core and closes the DM Core thread. RavenClaw uses extensive logging of each individual sub-component to give the end user a detailed insight into what is going on during a dialog session. The Logger is started on the initiation of a dialog session during which it receives the logging directory and log file name, and is stopped when a dialog session ends. The communication between different components is achieved by posting messages on threads. Following sub sections discuss the details of DM Core thread and Dialog Task Tree.

*1) Dialog Task Tree:* The dialog flow in RavenClaw is written in the form of a tree using RavenClaw Task Specification Language (RTSL). The RTSL provides various macros which are used to declare and define the nodes within a tree, perform various operations on each node, and to control the flow of the dialog. RTSL is built on C++ and certain macros allow to execute C++ code. The traversal pf Dialog Task Tree starts from left most leaf node and proceeds in left to right fashion. Figure 2 shows a Dialog Task Tree for a weather information system.

The nodes in the Task tree fall into two major categories: agents and agencies. Agencies are usually logical blocks that sub-divide different portions of the dialog based on the major tasks to be performed during a particular stage in the dialog. Every agency has sub-nodes, which can be another agency, or an agent.

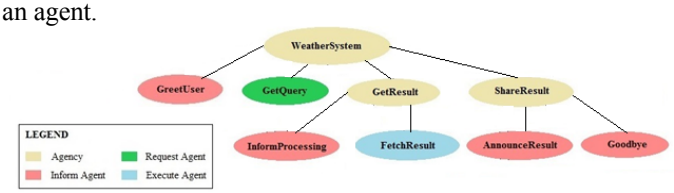


*Figure 2: Sample Dialog Task Tree.*

The WeatherSystem, GetResult, and ShareResult nodes in Figure 2 are the examples of an agency. The agents on the other hand are leaf nodes. The task performed by an agent is logically atomic and cannot be broken dwn any further. RTSL provides four kinds of agents: inform, request, execute and expect. The inform agents simply generate a system message when executed. The request agents generate a system message which is always a question, and then take the user's input. The execute agents are used for executing a back-end (database) query and then performing any necessary processing on the fetched result. Expect agents are used for generating system messages which aren't necessarily questions, but they can result in a user response. Generally expect agents are used in interactive dialogs where the caller can shift the course of a dialog. The GetQuery node is a request agent, the FetchResult node an execute agent and all other leaf nodes are inform agents. For each new application or project, the developer need only to design and implement a new Dialog Task Specification Tree.

*Dialog Management Core thread:* The Dialog Management Core thread of RavenClaw is created when the Galaxy framework sends the Galaxy Interface a begin_session command to initiate a dialog session. The DM Core thread has seven sub-components: Dialog Task Tree (DTT) Manager agent, State Manager agent, Interaction Event Manager agent, Traffic Manager agent, Grounding Manager agent, Output Manager agent and DM Core agent. The DM Core is the main core component that manages the execution of a dialog session. It makes use of the other agents to execute a particular portion of the Dialog Task Tree. The DTT Manager Agent is used to load and perform operations on the Dialog Task Tree. The DM Core loads the task tree and executes each node from left to right. During the execution, all system prompts are processed and the DM sends the output to the Galaxy framework via the Output Manager Agent. The Output Manager Agent also maintains a history of all the prompts synthesized during a dialog session. The calls to the database are processed through the Traffic Manager Agent. The result frame received from the database is also received via the Traffic Manager Agent and is then passed to the DM Core which then proceeds with the execution of the dialog task tree.

For system prompts that lead to an input, the Interaction Event Manager Agent receives the decoded input from the Hub, and sends it to the DM Core for processing and binding to the variables (called concepts in RavenClaw). Apart from capturing the inputs, all events received during the various stages of the dialog to update the state of the dialog manager are received via the Interaction Event Manager Agent as well. On receiving each event, the DM Core updates the dialog state and notifies the State Manager Agent. Various events include system_utterance_start, system_utterance_end, dialog_state_change and user_utterance_end. The system_utterance_start event denotes that the system response generator has received the prompt message to synthesize and the system_utterance_end event informs the DM Core about end of system utterance. The dialog_state_change event is sent after the system_utterance_end event before taking the user's input. The DM Core then loads the concepts for updating based on user's input.

The DM Core also sends state information to the State Manager Agent which generates a broadcast of the dialog manager's state and sends it to the Hub. After the user's input has been recorded and decoded, the decoded result is sent to the dialog manager in the user_utterance_end event. Then, the DM Core enters the concept binding phase i.e. the received input is bound to the concept that had been loaded after the dialog_state_change event. The DM Core makes use of the error recognition and handling techniques during concept binding. For this part, the Grounding Manager Agent is used. Based on the grounding policies defined in the RavenClaw configuration files, the Grounding Manager Agent checks whether there is a need to apply any error handling technique. Should there be any such need the appropriate grounding agent is pushed onto the DM Core's execution stack dynamically for execution.

### *B. Multiple Sessions in RavenClaw*

RavenClaw cannot handle multiple sessions due to the fact that at each point of time, RavenClaw must keep a record of the current dialog state and all the associated state variables. A number of global variables are also used for smooth communication like the thread IDs of DM Core and DM bridge, and the last frame sent by the DM. The logger module just takes the path of log file as argument and the name of log file is fixed, which also indicates that the developers of RavenClaw didn't have handling of multiple sessions in mind when designing the system. To add multiple-session handling functionality in DM, the only plausible solution seems to be to have a separate instance of DM running for each session. But for that, we need to add an intermediate module in-between Galaxy framework and the RavenClaw dialog manager which will keep track of which DM session is handling a specific Galaxy session. We have called this module Session Manager and its architecture and working is discussed in the next section.

*1) Session Manager:* Session Manager acts as a bridge between the Galaxy framework and the RavenClaw dialog manager. It starts and kills all instances of dialog manager, passes the frames (messages) from Galaxy framework to the respective dialog managers and starts and ends the session in dialog manager. The interfacing of Session Manager with dialog managers and Galaxy Hub is shown in Figure 3.

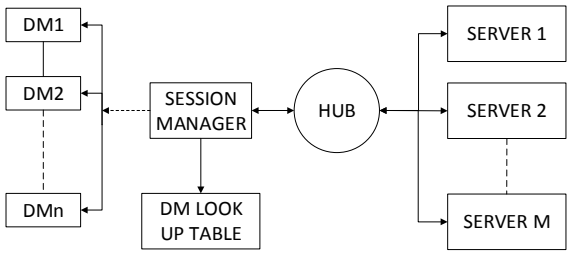

Figure 3: Session Manager interfaced with dialog managers and Galaxy Hub.

When Session Manager first starts, it reads the configuration files of Dialog Manager (DM), populates the initial dialog manager table and starts up DM instances. After performing these initial tasks, it manages the creation and termination of DMs. The details of each of these functionalities are discussed below:

- Read Configurations**:** The configuration files of Dialog Managers have some key information regarding the connections between dialog manager and the Galaxy hub. They list the IP address of the Galaxy hub and the port numbers at which dialog managers will contact hub, maximum number of dialog manager instances that can be run concurrently, number of dialog manager instances to start up-front before any execution starts and the number of instances of dialog manager that should be up and free at all times. The number of instances of DM that should be up and free at all times is important as it would save us from loading a new dialog manager instance each time a new session starts which might incur delay in initial response to the user.
- Populate DM Table: The information read by the Session Manager from the configuration files is stored in a table which is updated whenever a session starts or ends. Initially all the started dialog managers have their status marked as "FREE". Each time a new Galaxy session starts, a dialog manager is locked with that session.
- Start DM instances: Session Manager starts some DM instances initially so that whenever a DM session is required it can immediately be assigned without having to wait for any delay to start a DM instance.
- DMs Management**:** Once the Session Manager has done the above mentioned tasks, it starts to manage the DMs. Upon initiation of a new Galaxy session, Session Manager binds some DM instance with that session, marks it as "BUSY" and maintains the information in DM table. If a message has to be sent to DM, Session Manager looks up the id of DM instance associated with that session and passes that information to Hub to route the message to that DM. When an existing session is closed, the Session Manager removes

the DM entry from the DM table and marks the DM as "FREE".

### C. Galaxy Interface

Galaxy Interface is responsible for interfacing RavenClaw with the Galaxy framework. Originally, the design of the interface was such that Galaxy Hub acted as the client and the DM as the server, which is default mode for all Galaxy modules. This caused a problem in the case of multiple sessions as the Hub needs to connect with all the modules in advance before it starts any communication with them. This kills the purpose of introducing a new module to manage multiple instances of DM if we have to keep all of them running at all time. The only way to circumvent that problem was to reverse the client-server model for Hub and DM and make Hub a client and DM a server so that Hub can open ports at the start of its execution for DM to make connections and whenever a new DM instance starts, it connects with the hub at the specified port and when the session ends, it closes the connection.

## IV. Spoken Dialog System

This section discusses the design and development of a telephone based spoken dialog system. In order to develop a new dialog system, the framework is reusable and only some domain specific components are required to be developed. The architecture diagram of the complete system is given in Figure 4. The architecture diagram shows two machines: 1) Dialog Server and 2) Telephone Server. The telephone server runs CentOS while dialog server runs on Windows OS, hence, two separate machines are required.

### A. Telephone Server

The telephone server provides an interface between the dialog server and the PRI telephone line. The telephone server runs Trixbox CE[1], a VOIP system based on the Asterisk[2] Private Branch Exchange (PBX), which includes CentOS, MySQL and other necessary tools for a telephone server. Asterisk is a language for programming PBX type telephone servers. The PRI E1 line terminates at Digium's TE 110p[3] gateway card connected to telephone server. The telephone server plays the system response sent by dialog server to the caller and records the caller's response and sends it to dialog server. The telephone server uses socket connections to communicate with the dialog server.

### B. Dialog Server

Dialog server provides the major functionality of spoken dialog system. It runs our proposed framework which includes the Galaxy Hub and RavenClaw Dialog Managers (DMs) along with Session Manager to handle multiple calls. Galaxy Hub communicates with the telephone server through a Waiting module using socket connection. Waiting module also keeps track of the data coming from the telephone server. Some other modules shown in Figure 4 include Interaction Manager, Interactive Voice Response (IVR), Automatic Speech Recognizer (ASR) and Backend Server. Interaction manager acts as bridge between the Dialog Manager and rest of the modules. DM sends a frame within the frame, which is parsed by the Interaction Manager and sent to Galaxy Hub which forwards it to IVR. IVR module generates the system response depending upon the message sent by DM. ASR decodes user input and sends the decoded result to the DM. The Backend module fetches the information related to the query and sends it to IVR which generated the system response.

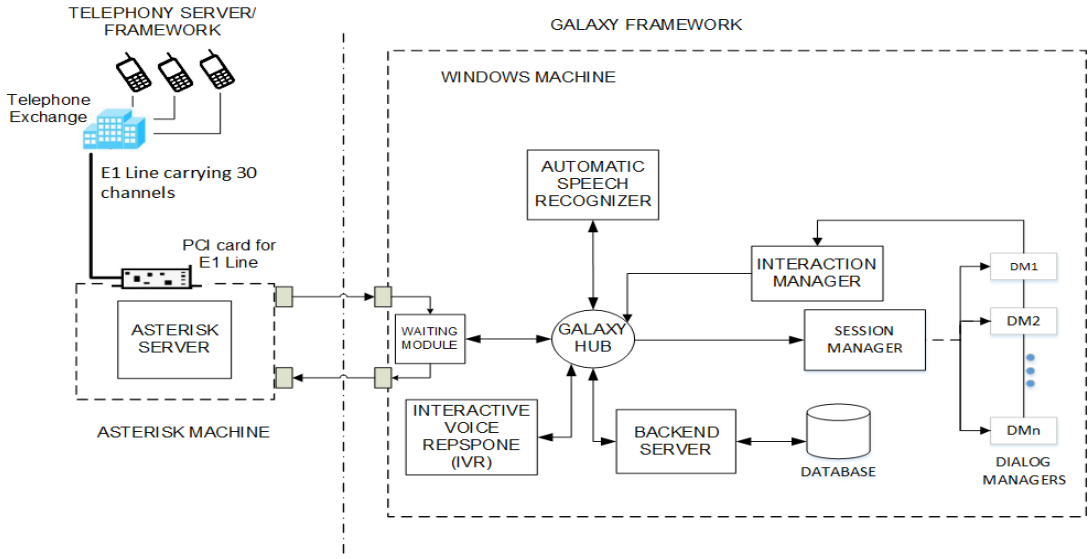


Figure 4: Architecture diagram of spoken dialog system with multiple call handling capability

[1] http://trixbox.org/
[2] http://www.asterisk.org/
[3] http://www.digiumcards.com/te110p.html

## V. Weather Information System

The proposed framework has been implemented and used in Weather Information spoken dialog system that provides weather information of 139 districts of Pakistan[4]. System is running on a PRI E1 line and currently ten concurrent calls are supported. It has been tested by placing ten concurrent calls. The testing session comprised of ten participants who called the system continuously for a duration of thirty minutes. The system performance during the testing period was normal as expected without any anomalies or breakdown in the system.

## VI. Conclusion

In this paper, we have discussed a framework for spoken dialog systems that supports easy to modify dialog manager and handles multiple concurrent sessions. Using this framework, spoken dialog systems for any language and domain can be constructed with quite ease. The framework has been tested to be working satisfactorily for ten concurrent sessions. It needs to be further tested for greater number of concurrent sessions to study any possible delays that may be introduced in the system. Multiple dialog systems may be deployed in such scenarios to avoid any such delay.

## Acknowledgment

This work has been conducted through the project, Enabling Information Access for Mobile based Urdu Dialogue Systems and Screen Readers supported through a research grant from National ICT RnD Fund, Pakistan.

## References

[1] J. Peckham, "Speech understanding and dialouge over the telephone: an overview of the ESPRIT SUNDIAL," in Human Language Technology workshop on Speech and Natural Language, Stroudsburg, PA, USA, 1991.

[2] V. Zue, S. Senef, J. R. Glass, J. Polifroni, C. Pao, T. J. Hazen and L. Hetherington, "JUPITER: a telephone-based conversational interface," IEEE Transactions on Speech and Audio Processing, vol. 8, no. 1, pp. 85-96, 2000.

[3] S. Seneff and J. Polifroni, "Dialogue Management in the Mercury Flight Reservation System," in Satellite Dialogue Workshop of the Applied Natural Language Processing - North American Chapter of the Association for Computational Linguistics Meeting, Seattle, Washington, USA, 2000.

[4] D. J. Litman and S. Silliman, "ITSPOKE: an intelligent tutoring spoken dialogue system," in North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Demonstrations 2004, Stroudsburg, PA, USA, 2004.

[5] H. Bratt, J. Dowding and K. Hunicke-Smith, "The SRI telephone ATIS system," in Spoken Language System Technology Workshop, Austin, Texas, 1995.

[6] D. L. Martin, A. J. Cheyer and D. B. Moran, "The Open Agent Architecture: A framework for building distributed software systems," Applied Artificial Intelligence: An International Journal, vol. 13, no. 1-2, pp. 91-128, 1999.

[7] R. Moore, J. Dowding, H. Bratt, J. M. Gawron, Y. Gorfu and A. Cheyer, "CommandTalk: A Spoken-Language Interface for Battlefield Simulations," in Fifth Conference on Applied Natural Language Processing, Washington, DC, USA, 1997.

[8] M. Johnston, S. Bangalore, G. Vasireddy, A. Stent, P. Ehlen, M. Walker, S. Whittaker and P. Maloor, "MATCH: An architecture for multimodal dialogue systems," in 40th Annual Meeting on Association for Computational Linguistics, Philadephia, Pennsylvania, USA, 2002.

[9] H. M. Meng and e. al., "ISIS: A multilingual spoken dialog system developed with CORBA and KQML agents," in INTERSPEECH, Beijing, China, 2000.

[10] A. Raux, B. Langner, Bohus, B. D., A. W. and M. Eskenazi, "Let's go public! taking a spoken dialog system to the real world," in Interspeech, Lisbon, Portugal, 2005.

[11] D. Bohus and A. I. Rudnicky, "LARRI: A language-based maintenance and repair assistant," in Spoken multimodal human-computer dialogue in mobile environments, Springer Netherlands, 2005, pp. 203-218).

[12] T. K. Harris, S. Banerjee, A. Rudnicky, J. Sison, K. Bodine and A. Black, "A Research Platform for Multi-Agent Dialogue Dynamics," in IEEE International Workshop on Robotics and Human Interactive Communications, 2004.

[13] D. Bohus and A. I. Rudnicky, "The RavenClaw dialog management framework: architecture and systems," Computer Speech and Language, vol. 23, no. 3, pp. 332-361, July 2009.

[14] V. Catania, R. Di Natale, A. R. Intilisano and A. Longo, "A Distributed Multi-Session Dialog Manager with a Dynamic Grammar Parser," International Journal of Information Technology & Computer Science ( IJITCS ) , vol. 8, no. 2, pp. 1-9, 2013.

[15] A. Pappu, M. Sun, S. Sridharan and A. Rudnicky, "Situated multiparty interaction between humans and agents," in 15th International Conference on Human-Computer Interaction: Interaction modalities and techniques, Las Vegas, NV, USA, 2013.

[16] D. Goddeau, E. Brill, J. R. Glass, C. Pao, M. Phillips, J. Polifroni, S. Seneff and V. Zue, "Galaxy: A human-language interface to on-line travel information," in Third International Conference on Spoken Language Processing, Yokohama, Japan, 1994.

[17] A. Rudnicky and W. Xu, "An agenda-based dialog management architecture for spoken language systems," in IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), Keystone, Colorado, U.S.A, 1999.

[18] D. Bohus, A. Raux, T. K. Harris, M. Eskenazi and E. I. Rudnicky, "Olympus: an open-source framework for conversational spoken language interface research," in HLT-NAACL, Rochester, New York, USA, 2007. K. H. Davis, R. Biddulph and S. Balashek, "Automatic Recognition of Spoken Digits," Journal of the Acoustical Society of America, vol. 24, no. 6, pp. 627-642, 1952.

[4] The weather information system is online at UAN +925111638638.