A Configurable Dialogue Platform for ASORO Robots

Ridong Jiang, Yeow Kee Tan, Dilip Kumar Limbu, Tran Anh Tung and Haizhou Li,
Institute for Infocomm Research, 1 Fusionopolis Way, #21-01, Connexis, Singapore 138632
E-mail: {rjiang, yktan,dklimbu,tanhdung, hli }@i2r.a-star.edu.sg

***Abstract*—This paper is concerned with the architectural design and development of a spoken dialogue platform for robots. The platform adopts modular software architecture and event driven communication paradigm which makes speech enabled hardware devices and software components configurable and reusable. The platform is able to integrate heterogeneous dialogue components (such as speech recognizer, natural language understanding, speech synthesizer, etc.) through message-oriented middleware and a number of adaptors in a plug-and-play fashion. The dialogue system is empowered by a multi-purpose XML-based dialogue engine which is capable for pipeline information flow construction, programmable event mediation, multi-topic dialogue modeling and different types of knowledge representation. The proposed platform provides a generic framework for the easy and quick construction of robust, efficient and flexible spoken dialogue applications ranging from simple state-based dialogue prototype to complex frame-based and plan-based, mixed-initiative, multi-topic spoken dialogue applications. Most importantly, the system is cross platform, domain independent, service oriented, extensible through various plug-ins and capable of knowledge sharing within components. The platform has been deployed on various robots (robotic butler, smart home helper robot, receptionist, etc.) in A*STAR Social Robotics (ASORO) laboratory for the creation of different spoken dialogues. Results showed that components can be reused and configured for different robots without any programming effort while the development cycle was reduced significantly.**

I. INTRODUCTION

Spoken dialogue systems, as an intuitive, flexible, natural and inexpensive means of communication between users and machines, have obtained an increasingly widespread use in a wide variety of applications: voice-operated cell phones, car navigation systems, commercial information retrieval, gaming, education, healthcare and talking agents, etc [1].

In recent years, with the rapid advancement of social mobile robot and its wide application to various domains, spoken dialogue technology plays pivotal roles in human robot interface (HRI) and social robot intelligence. With more and more social robots are making their way into the world of our everyday life, for instance, bank tellers, tour guides, elder care nurses, home maids and receptionists [2~5], there is an increasing need on spoken dialogue applications which can be equipped on various robots for natural and intuitive human-machine communication. As we know, different robots come with different functionalities and their hardware configurations vary from one to another. For instance, some robots require close-talk microphone to record speech, while some other robots use microphone array for speech input. There are cases that a robot needs audio streaming as input for speech recognition. In the mean time, these robots are running on different software platforms, e.g. Microsoft Windows, Linux and Macintosh. All these facts pose a great challenge to us: How to design a spoken dialogue platform that works for as many robots as possible without repeating effort? More specifically, this in fact proposes requirements on spoken dialogue platform in the following aspects: a) Capable of integrating heterogeneous dialogue components locally or remotely; b) Capable of representing tasks, dialogue move and knowledge in different domains; c) Configurable, scalable and extensible when robot hardware changes, upgrades or robot transforms; d) Ease of use for users without programming background; e) Interoperability across platform.

The aforementioned challenges and our past experience working on different social robots motivate us to develop a component pluggable event driven spoken dialogue platform which targets for a broad spectrum of domains. A component can be any spoken dialogue module such as speech recognizer, natural language parser, speech synthesizer, etc. It can also be a hardware device which performs a specific function such as microphone, push-to-talk device. More generally, a component can be an algorithm (e.g. voice activity detection), a database connection, a graphic user interface or a remote agent. These components are integrated into the spoken dialogue framework by generic or tailored adaptors. Once a component is integrated, it provides services to the framework. In the mean time it can also share all the resources or services within current framework. To develop a new spoken dialogue application, what one needs to do is to assemble required components in a configuration file or dynamically load/unload the necessary components through dialogue script based on the current situation. The communication between different components can be done by triggering events which embed different messages for different purposes in a unified format. TCP/IP messages are also allowed and compatible when components are located on different machines. One message can be sent directly from one component to another one, it can also be relayed and routed in XML script through which message transforming, redirecting and handling are possible. Pipeline information flow and dialogue flow are managed by XML script through built-in finite state machine and/or frame-based slot filling.

In this paper, we start by reviewing some related spoken dialogue frameworks in section II. Then the system architecture is presented in section III. After that, the dialogue modeling supported by the platform is described in section IV. Section V presents a case study on how the platform can be used to configure the speech interface for a receptionist robot with minimum effort. Finally, we conclude our work on the configurable spoken dialogue platform and some possible future enhancement.

II. Related Work

Building a spoken dialogue application is a complex task. One has to deal with spoken dialogue related components, for instance, sound recording, voice activity detection (VAD), speech recognition, natural language understanding(NLU), natural language generation (NLG), text to speech (TTS), etc [1]. In the mean time, a dialogue manager is needed to manage the communication between these components, as well as task representation, domain knowledge representation, dialogue flow control, turn taking, confirmation, error-handling. For natural and intuitive communication, ambiguity, context, initiative, dialogue history, reasoning and planning must also be properly addressed. In addition, some other features such as system flexibility, extensibility, maintainability, cross-platform and ease of use must also be considered when building up practical spoken dialogue applications.

Much effort has been made by researchers from academic and industrial institutions. A notable project is American DARPA communicator project [6][7]. The Galaxy Communicator software infrastructure is a distributed, message-based, hub-and-spoke infrastructure optimized for constructing spoken dialogue systems. Each component is implemented as a separate process that connects to a traffic router - the Galaxy hub. The messages are sent through the hub, which forwards them to the appropriate destination. The routing logic is described via simple configuration script. Many spoken dialogue systems were developed on top of Galaxy II communicator, for instance, Olympus dialogue system [8], AT&T Spoken dialogue system [9]. Galaxy II based systems treat every component as a separate server. It needs the support of Python and environment setting. The hub script provides every limited programming capability. Another widely used dialogue system is information state update based approach. Trindikit [10] is a toolbox for building dialogue managers based on an information state and dialogue move engine. DIPPER [11] is implemented on top of Open Agent Architecture and it comes with its own dialogue management component which is similar to Trindikit. ISU-based systems require open agent architecture for communication and non-free dialect of the programming language Prolog for information state update and dialogue control [12]. VoiceXML is another widely used dialogue system for creating distributed voice applications that users can access from any telephone. However, VoiceXML lacks features on database access, programming logic control, modeling of returning dialogue flow, etc. [13].

The spoken dialogue platform presented in this paper employs loose component coupling, event-driven paradigm and service oriented architecture for component reuse. The component can be either a local plug-in or remote agent. Messages can be generated, mediated or transformed in programmable message centre. The dialogue system supports state-based and frame-based (or their mix) dialogue management techniques. The built-in dialogue topic engine enables multi-topic and topic changing management as well as support of different dialogue strategies: user-initiative, system-initiative and mixed-initiative. The dialogue system is driven by structured and extensible XML script which in fact becomes a powerful programming language supporting variables, array, list, string handling, expression, if/else, for loop, functions, timer, file I/O, database, etc. Most importantly, the dialogue platform can be enhanced by new plug-in development, for instance, plug-in for the support of artificial intelligence markup language (AIML) and ontology.

III. System Architecture

The dialogue platform is composed of a dialogue manager, a number of dialogue components, middleware interface to remote agent and graphic user interface, as shown in Fig. 1.

A. Message Centre

The message centre is an event-driven messaging engine for the effective message routing. Every message from different sources is passing through this hub. It provides a mechanism for the communication between dialogue components, as well as remote agents. All messages from different sources are represented in a unified form and can be dispatched and handled in the same way. A message may come from a local or remote dialogue component, XML script command or a user command issued by graphic user interface. A typical message usually includes time stamp, message source (module identifier), purpose of the message (command) and a series of parameters. This type of message is handled by message centre based on current situation. For instance, a recording buffer full event from microphone can be directed to a VAD module or multiple speech recognizers by event hub script programming. Besides the above basic elements of a message, some messages include event sink information, i.e. where this message will be going to. Following command is a message sent to microphone to request sound recording:

<post module="Microphone" command="start_mic"/>

Fig. 2 shows conceptual diagram of the message centre with the possible inputs and output. Functionally, the message centre is able to process all events in three ways as displayed in Fig.2. First is the direct message relaying. One message can be directly forwarded to one or multiple target agents with or without the change of parameters. Second is that the event is filtered by simple if/then rules, and then based on the filtering result the system decides how to forward or generate an event. The last is event handling by a piece of XML script program. This is the most complicated and intelligent event processing. The script program will base on the receiving event, the

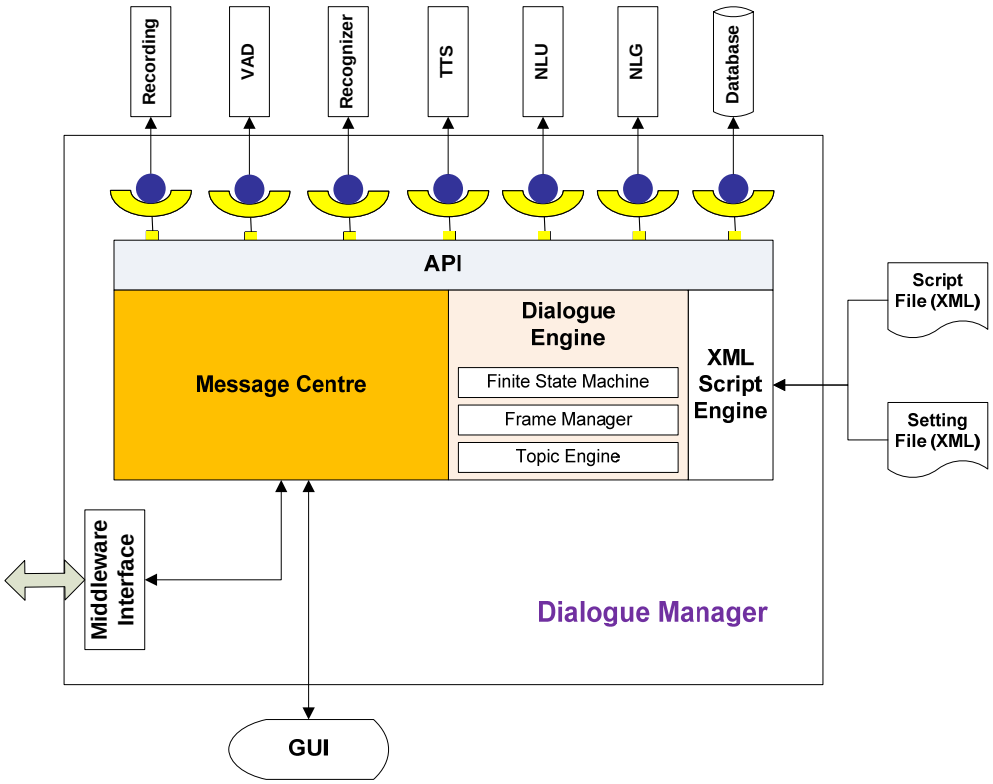

Fig.1 Conceptual diagram of message centre

current context and internal state of the robot to decide how to proceed with the current and new events.

The construction of pipeline information flow can be easily implemented by message centre programming in XML. This brings great flexibility for spoken dialogue development based on components. Following statement invokes an event in the context of speech recognition and sends recognition result to language understanding module for processing. "*_LPARAM1_*" is the first parameter sent from speech recognizer component which represents the text of recognition result.

<post module="NLU" command="DoNLU" param="_LPARAM1_"/>

B. Dialogue Engine

Dialogue engine is the core functional element in dialogue manager. It consists of finite state machine, frame manager and dialogue topic engine. Finite state machine is used for simple dialogue task representation and fast spoken dialogue prototyping. In one state, entry action, exit action, input action and transition action can be defined in XML script. In the mean time, within every state, all programming statements supported by the engine (assignment of variables, function calls, timers, etc) can be used. Frame manager is suitable for more complex dialogue management. In this case, dialogue is modeled as frames and filling of semantic slots. This avoids predefined states and transition network. Hence it is more flexible and natural for communication. Dialogue topic engine is designed for modeling and management of multiple topics. It registers and manages all the dialogue topics, actively initiates a conversation and asks questions related to a topic for more information so that all semantic slots are filled. The topic engine makes it possible for multiple topics conversation. Users can switch topics and continue a previous unfinished topic. The dialogue engine allows mixed use on all these elements. For instance, a dialogue task can be modeled as a number of frames and the frames are registered with topic engine for topic management. With this the dialogue system becomes a frame-based, mixed-initiative, multi-topic dialogue system. However, the system still allows a small subtask to be finished by a dialogue state such as confirmation. When this dialogue state is activated, it captures all events and tries to finish its task and then continues frame-based dialogue management.

The dialogue engine is fully driven by XML script such as the finite state representation, state transition network, structured topic representation, etc. The dialogue engine works seamlessly with the XML script engine which will be described with more details in the following subsection. Any XML script programming statement can be used in the description of tasks related to dialogue engine. For instance a normal function call is allowed in the field of semantic slot filling. Similarly, inside a function definition, state transition can be done if the transition condition is triggered for whatever reason.

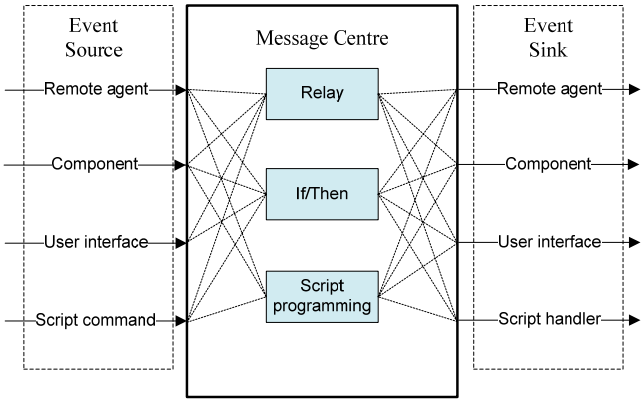

Fig.2 Conceptual diagram of message centre

C. XML Script Engine

Script engine is an execution component for dialogue engine. In addition, it is used for system configuration (XML setting file), event hub programming, pipelined information flow construction and XML script programming. In selecting a language for dialogue modeling, we realized that XML would be the most appropriate and promising language to use. In fact, XML is becoming a dominant scripting language for information storing, organization and business logic control because of its portability, extensibility, reusability and structured nature. In terms of programming capability, the script engine is becoming a full-fledged XML script engine which supports scoped variables, arrays, list, string handling, expression, if/else statement, arithmetic and logic operators, for loop, functions, timer, file I/O as well as database access.

D. API for Plug-in

If a dialogue component is integrated into the framework as a plug-in, API is provided for easy integration. Once a component is integrated, it is able to share all the resource provided by the framework and other plug-ins. For instance, access database through a database plug-in, sending a TCP/IP message to remote agent through the framework, etc. API is packed as various adaptors. Some adaptors are tailored for particular plug-ins. For example, a TTS adaptor is prepared for text-to-speech component. It defines standard interface for speech synthesizer. Currently, adaptors for TTS, VAD, microphone, speech recognition and natural language understanding are available for quick integration of respective dialogue components. We have successfully integrated Far talk and close talk microphones, Windows TTS (SAPI), Loquendo TTS, our in-house developed speech recognizer – Abacus, Dragon recognizer, robust language understanding, keyword spotting, etc. into the framework. All these integrated components can be reused in different projects by configuration in a XML setting file or by dynamic load/unload in XML script. A generic adaptor provides generic interface for all plug-ins. Followings are three important interfaces provided by the generic adaptor for component communication:

- OnMessage – This is the interface for a component to handle all coming messages from external sources. The message can come from a remote agent, other component in the framework, or the framework itself, which fires the message in XML script or through its graphic user interface.
- FireEvent – Interface to fire an event by the particular component. This event can be captured by event hub and handled in XML script, or relayed to one or many other components.
- SendMessage – This is the interface to communicate with remote agent through the platform. This creates a way for new plug-ins to directly communicate with remote agents or dialogue components. For instance, a dialogue plug-in can directly send a message to the robot to perform certain actions or even request information from a particular sensor.

The adaptors are developed with the concept of object oriented programming. The tailored adaptors inherit from the generic adaptor. A new plug-in will inherit from the generic adaptor or one tailored adaptor. New plug-in can be quickly developed with the support of generic adaptor “IPlugin” or other tailed adaptors because these adaptors provide interfaces and data structures to facilitate the communication between the new plug-in and the platform. Through these interfaces, the new plug-in provides services to other plug-ins and XML script. In the mean time the new plug-in also shares all the resources provided by other plug-ins. For instance, a keyword spotting plug-in is able to access database through the functions provided by a database plug-in.

E. Middleware Interface

Middleware interface provides a standard communication protocol to facilitate the information transmission between remote agent and the dialogue framework. Currently, Transmission Control Protocol (TCP) structured message is used by the system. TCP protocol ensures that every message for intercommunication can be sent to destination without any possibility of message loss. There are two types of messages: text and binary. Message type can be identified by a particular field in the message. In the mean time, every message also includes the following fields for clear and efficient communication: Time stamp, Component identifier, the command (purpose) of this message, and a list of parameters which is delimited by comma [14]. The communication middleware can be extended to support service oriented communication middleware such as SOAP and CORBA.

A dialogue component can also be developed as standalone remote agent which communicates with the dialogue platform through middleware interface. For instance, Nuance recognizer, Loquendo TTS, Microphone array and speech enhancement were integrated into the platform for different robots.

F. Graphic User Interface

The spoken dialogue platform provides a standard graphic user interface to show the current status of a human robot

interaction such as dialogue states, dialogue move, event details, value of variable, etc. The interface is configurable to certain extend based on the settings in a XML setting file. Every dialogue component integrated into the platform is also able to print out messages to this graphic user interface. Its basic functions can be summarized as follows:

- Display server information – The interface displays server information such as IP address and socket port number. The connection with remote server can be manually controlled through this interface.
- Display all messages and status of finite state machine (FSM) – There is a console to display all receiving messages, sending messages, the current state of FSM and its transition. In addition, the console can be set to display limited debug information such as to track the change of variable value.
- Tool for firing events – Through the interface, an event can be fired in the similar way of script programming. For instance, dynamically load/unload a plug-in, send a message to remote agent, invoke a command to a particular plug-in, etc.
- Reload XML script – One brilliant feature of the platform is that at any time, the XML script can be edited and reloaded on the fly without termination of the dialogue application. System is reset and the dialogue restarts from beginning based on the edited XML script. Through the graphic user interface, this can be done with the ease of one mouse click.

IV. DIALOGUE MODELING

Since a spoken dialogue system mediates the communication between human and an information system, first the information system must be modeled with proper structure and knowledge. The knowledge can be organized as common knowledge and domain knowledge. Domain knowledge is closely related to the specific dialogue task. In the mean time, dialogue task must be represented to facilitate the communication so that natural conversation and user's interaction goals can be achieved. Finally, the interaction management [15] which deals with the dialogue initiative and turn taking must also be addressed in the dialogue modeling.

The spoken dialogue platform for ASORO robots supports knowledge representation in the forms of XML script, database query, artificial intelligence markup language (AIML) and ontology. The dialogue task modeling and different dialogue strategies can be done through built-in state machine automata and frame-based multi-topic management. For process-oriented task, e.g. spoken dialogue for robots performing fetch-and-carry task, state machine automata is more suitable to represent the process. For information-oriented task, it will be quite clumsy to predefine the dialogue into different states, e.g. the information query about facilities in a building complex. The frame-based dialogue modeling can better address the problems related to information-oriented task. For very complex task which include both process-oriented and information-oriented actions, the proposed dialogue platform allows mixed representation of these two types of tasks as well as the dialogue management.

A. *State-based Task Representation*

This is the simplest task representation. Dialogue is modeled as a state transition network. First task is decomposed into predetermined steps or stages [16]. In every state, expected events are processed and grammars relating to this particular step are used. Then transitions are designed based on the dialogue control logic. The transition is triggered by certain events or after execution of a piece of XML script. The advantage of this task representation is easy to build, possesses better task success rate. However, it inhibits user's opportunity to take initiative, usually requires explicit confirmation and longer dialogue, hence it is not efficient and less natural.

In the modeling of state-based dialogue task, the platform allows following fields to be used in a dialogue state:

- *Entrance* – defines the first entry when the current state is started. Some initialization can be done in this field.
- *Exit* – opposite to the *Entrance*, this field defines the actions for leaving this state, for instance, calling some clearing functions when the dialogue exits from the current state.
- *Messages* – defines actions responding to different messages. Only when the state is in active state, the platform will direct messages to this state. This is the field handling various messages from agents or components.
- *Online* – defines message handler which responses to the event when a remote agent is online.
- *Offline* – defines message handler which responses to the event when a remote agent is offline.
- *functions* – a place holder where script functions are defined.

The transition action from current state to another state can be defined in any field as shown above. The transition statement can be embedded in a *function* or inside an *if* statement as shown below:

```
<if cond="_FrameDone==true">
  <transit state="Wait"/>
</if>
```

B. *Frame-based Task Representation and Multi-topic Management*

In the frame-based task representation, the dialogue is modeled as frames and filling of semantic slots. This representation is more flexible than state-based task representation because dialogue flow is not predefined. It allows mixed-initiative and more informative answers. It is more natural. The challenge for this representation is to model very complex dialogue which may include many dialogue

topics. Current dialogue platform provides a mechanism to effectively model a dialogue topic. Every topic encapsulates semantic slots, methods, constraints, facts and rules. In the mean time, confirmation and attributes-based knowledge query can be performed in a topic. Fig. 3 demonstrates how the platform transforms an utterance to a knowledge query.

First, the dialogue manager passes an utterance from speech recognition engine to natural language understanding (NLU) component. After analyzing the utterance, the NLU component posts a message to topic classifier with its parsing result. The topic classifier detects the topic and activates dialogue topic manager for matching and slot filling on the detected topic. Based on the slot filling results, the topic manager will dynamically form a knowledge query statement. The query statement then is sent to knowledge manager for acquiring of the related knowledge. The acquired knowledge will be posted back to the dialogue manager in the form of an event which may lead to a response to the user. Some of the process is demonstrated in case study in section V.

The knowledge can be built into a topic in XML script. A database plug-in and an AIML plug-in were developed to enable the platform for the access of database and knowledge defined in AIML language. We are also in the process to develop an agent which is capable of manipulating ontology. The communication between the ontology agent and the platform is through communication middleware.

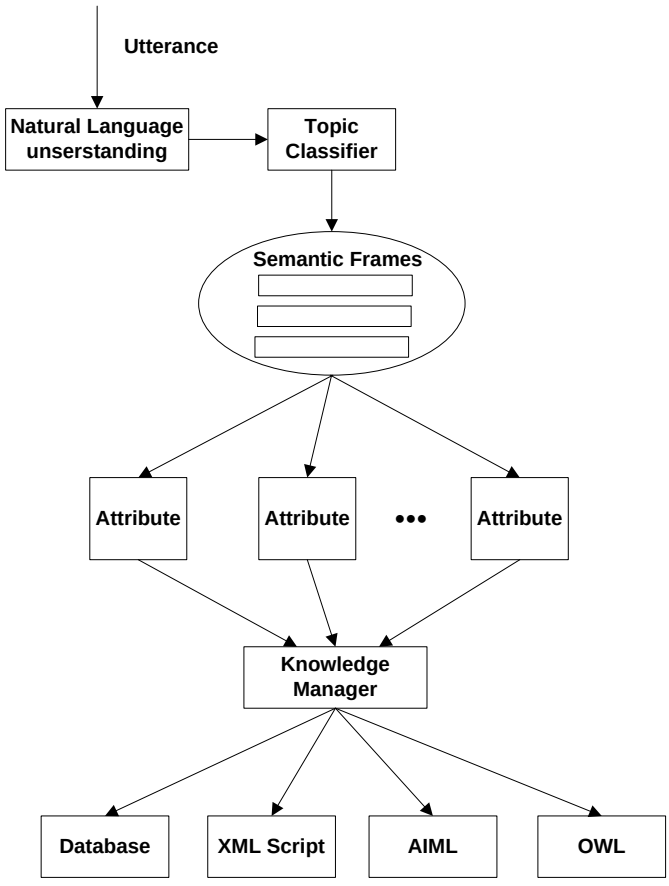

Fig.3 Working model of frame-based dialogue system

The platform allows multi-topic definition as well as multi-topic management for the modeling of complex dialogue applications. In the case study described in section V, the multiple tasks of a receptionist robot were defined in XML script. To enable mix-initiative dialogue strategy, topics must be registered with the topic manager. Once a topic is registered, the topic manger will actively check the status of every topic after an event for slot filling. If a dialogue topic is completed, consequent action will be taken, for instance registering the visitor to visitor database. Then the topic manager may actively change the communication to another registered topic and start the conversation of a new topic. If there is one uncompleted topic due to interruption, the system will give higher priority to the uncompleted topic and continue the conversion based on the status before interruption.

V. CASE STUDY

A. Receptionist Robot and Task Description

We have developed a receptionist robot based on this configurable spoken dialogue platform. The receptionist robot is a social robot designed to interact and serve visitors in office environment. This robot is equipped with speech recognition, sound localization, vision understanding and motion control for head, body and hands movement. The spoken dialogue designed for this receptionist robot is for receptionist task such as appointment, call transfer and taxi booking. In addition, the robot will be able to present information about building amenities within Fusionopolis and general inquiry about weather information and traffic information. Specifically, the robot must be able to response and talk about following topics:

- General greeting and bidding good bye
- Appointment: Visitor would like to meet someone or call someone. The robot will check the appointment database and notify the staff involved
- Facility enquiry: Visitor would like to check the availability of facilities or ask for more information about a specific facility
- Call transfer: Search the contact of a staff and transfer a phone call
- Taxi booking: Book a taxi and SMS the booked taxi plate number to visitor
- General enquiry: Visitor would like to check weather information and traffic information

B. Dialogue Configuration

As presented in the section of system architecture, dialogue components can be reused by just specifying the components in XML configuration file. In this design, all the components used are from existing components developed by our laboratory as shown in Fig. 4.

```
<Plugins>
  <Plugin>SR_WinTTS</Plugin>
  <Plugin>SR_SpeechEnhancement</Plugin>
  <Plugin>SR_Dragon</Plugin>
  <Plugin>SR_SLUR</Plugin>
  <Plugin>SR_NLG</Plugin>
  <Plugin>MySQLDatabase</Plugin>
</Plugins>
```

Fig.4 Components Setting for Receptionist Dialogue

In this setting, we use far talk microphone array to record speech, Dragon speech recognizer for speech recognition, our in-house developed spoken language understanding module to interpret the utterance, in-house developed NLG module which supports AIML to generate response, Windows SAPI is used for speech synthesis. In addition, a database component is used to access our staff database and facility database. A component can be replaced by any other component of the same type. For instance, if we change "SR_WinTTS" in Fig. 4 to "SR_Loquendo", then the TTS of current dialogue system is changed to Loquendo without any other changes involved. Most importantly, the pipeline information flow can also be constructed by a message handler as shown in Fig. 5.

As shown in Fig.5, once the message handler is registered, it starts to direct messages from one component to another component. First the microphone triggers the "*BufferFull*" event and sends a message to the message centre when its recording buffer is full. The message handler catches this message and posts a message to speech recognizer with the buffered audio stream. The speech recognizer then performs speech recognition and sends a message "*RecognitionResult*" to the message centre with its recognition result. Then the message centre directs this message to natural langue understanding component. The natural language understanding module will send back its result in message "*FrameInfo*" when it finishes the parsing. With the result from natural language understanding, the message centre passes the understanding result to dialogue manager to do topic classification and semantic frame filling. The dialogue manager will call natural language generation module based on the context and embedded knowledge. The natural language generation component then generates the response and posts its result to the message centre in its message "*Speak*". Once receiving the "*Speak*" message, the message centre will direct this message to speech synthesizer. Finally the system responds to the user in speech. The pipeline information flow is graphically shown in Fig. 6.

```
<handler name="CommonHandler">
  <messages>
    <component>
      <Record>
        <BufferFull>
          <post module="ASR" command="DoRecognition"/>
        </BufferFull>
      </Record>
      <ASR>
        <RecognitionResult>
          <var name="text" expr="_LPARAM1_"/>
          <call function="NameVariationMap">
            <var expr="text"/>
          </call>
          <post module="NLU" command="DoNLU" param="text"/>
        </RecognitionResult>
      </ASR>
      <NLU>
        <FrameInfo>
          <object name="nluRes" class="list"/>
          <getlist name="nluRes"/>
          <fire param="nluRes"/>
        </FrameInfo>
        <NoResult>
          <speak param="Sorry. I do not fully understand your talk…"/>
        </NoResult>
      </NLU>
      <NLG>
        <Speak>
          <speak param="_LPARAM1_"/>
        </Speak>
      </NLG>
    </component>
  </messages>
</handler>
```

Fig.5 Construction of pipeline information flow

C. *Database for Receptionist*

The receptionist functions rely very much on receptionist database. We built up a MySql database with following tables to support the receptionist tasks:

- Staff database – More than six hundred entries with staff ID, salutation, full name, designation, telephone number, email address, department, location, etc.
- Facility database – About fifty entries with facility ID, name, category, tags, unit number, description and contact number
- Meeting database – This is a database recording everyday's meeting information. It includes meeting time, venue, visitor information, etc.
- Visitor database – Database recording visitor's information such as visitor's name, salutation, contact number, email address, organization, etc.

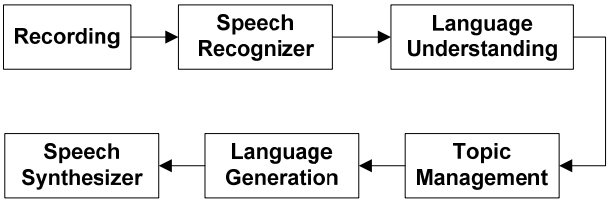

Fig.6 Diagram of pipeline information flow formed in Fig. 5

Currently, meeting database and visitor database have not been integrated yet. Staff database and facility database can be connected and queried in XML script by commands "*connectdb*" and "*querydb*". Following statements query staff names from staff database by current database connection:

<object name="staffList" class="list"/>
<querydb param="'select staff_name from rr_staff'" return="staffList"/>

In addition, spelling variation of names was done in script so that the dialogue system is still able to understand staff names when they are called in different sequences

D. *Task Representation*

A dialogue topic can be represented as state or frame in XML script. A frame consists of a number of slots. A slot includes several attributes which are designed for question generation, value constraint and action. Typically, a slot can include following attributes:

- name – Name of the slot. It cannot be repeated in the same frame
- type – A slot can be functional slot or informational slot. Functional slot must be filled up in order to complete the frame, while informational slot is optional
- expression – This attribute allows a slot to have a default value, which can be an expression
- question – Question for this sot. Dialogue manager always attempts to finish a topic. It will check every functional slot in current frame. If this slot is not filled, it will pose this question and try to seek the slot value from user

In addition, a slot can have following sub-nodes for constraints and actions:

- cardinality – Represent the constraints of current slot
- help – Similar to "question" attribute, this is more generic representation for questions for this slot. Many variations of questions can be used
- filled – Specify actions to be executed when this slot is filled up. Any action is allowed, for instance, sending a message to TTS for speech, call a function, query a database, transit to a new dialogue state, etc.

```
<slot name="FacilityName"
          type="Functional"
          expr="unknown"
          question="Which facility do you want to know more?" >
    <cardinality type="set">
      <value>Swimming pool</value>
      <value>Sky garden</value>
      <value>Auditorium</value>
      <value>Star home</value>
    </cardinality>
    <filled>
      <querydb param="'select * from facility where name=\''+this.FacilityName+'\''"
               return="dbRes"/>
      <post module="NLG" command="Generate" param="dbRes[6]"/>
    </filled>
  </slot>
```

Fig. 7 Sample slot representation on facility name

Fig. 7 shows an example on facility name slot which includes most of the attributes and sub-nodes stated above. The slot "FacilityName" is a functional slot which must be filled up for further conversation. Here we assume that the allowable facilities are swimming pool, sky garden, auditorium and star home. <filled> action can be defined in slot level as shown in Fig. 7. Alternatively, it can be defined in frame level. In the frame level, <filled> node is defined for the action when all its functional slots are filled. When one topic is completed, the dialogue topic engine will search all the registered topics and try to continue: a) a previously unfinished topic because of topic changing; b) a new topic if no any unfinished topic exists. This new topic will be set as current topic. The system will initiate the conversation if the user has not done so.

For the proper design of slots of a topic, it would be very helpful to analyze possible conversations and queries. Based on the analysis, categorized information can be used to decide what kind of slots should be defined. For instance, facility information query can be done by facility name, category, location, etc. Hence, facility name, category and location can be the slots of facility. However, the category, location, product type and action should be informational slots. These slots do not directly contribute to the fulfillment of the topic "Facility", they are attributes that can be used to help information query about facility. Fig. 8 shows conceptual frame deisgn of facility frame.

Facility enquiry can be based on one or multiple slots, for instance:

User: *Where is the swimming pool ? (name)*
Receptionist*: The swimming pool is on the twenty third floor,…*
User: *Is there any food court in Fusionopolis ? (category)*
Receptionist*: Yes, there is one food court named Food Chain food court at basement one.*
User: *Can I find any Coffee shop on the first floor? (product type, location)*
Receptionist:*Yes, Starbucks Coffee and Ya Kun Toastwich are on the first floor.*

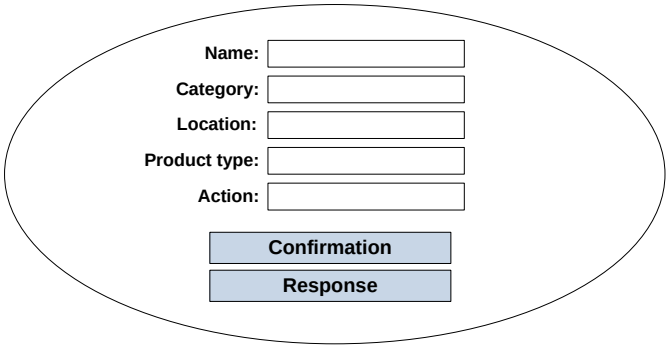

Fig. 8 Conceptual frame design for facility

We represent all the dialogue tasks in different dialogue frames and integrate the dialogue system with MySql database. A robust spoken language understanding component is used to transform speech recognitions results into interpretable representation of meaning. The representation can further be mapped into topics and semantic slots. Preliminary test was carried out on our social robotics test-bed, OLIVIA [17][18]. The receptionist is able to actively manage all its tasks and reply to the user based on the information from database and current dialogue status. Table 1 shows sample communication between user (U) and receptionist (R) on different topics.

VI. Conclusion and Future Work

This paper presents a configurable spoken dialogue platform for ASORO robots. The platform is able to integrate

TABLE I
SAMPLE CONVERSATION BETWEEN USER AND ROBOT

Communication	Comments
R1: *Good morning, how can I help you?*	System initiative
U1: *I am David from Comptec to meet Dr. Jeffery Ho.*	
U2: *By the way, is there any coffee shop on this seond floor?*	User initiative & change topic
R2: *No, however, there are two coffee shops on the first floor, namely, Starbucks Coffee and Ya Kun Toastwich.*	Query Database by category and location
U3: *How about food court?*	
R3: *There is one food court named Food Chain food court at basement one.*	Query Database by category
R4: *What's the time of your appointment?*	Change initiative & continue unfinished topic
U4: *10:30am.*	
R5: *Just a moment. I will call Dr. Jeffery Ho.*	Frame completed

heterogeneous dialogue components and engines through communication middleware or a number of adaptors in a plug-and-play fashion. The proposed technology employs object-oriented approach, loose component coupling, event driven paradigm, hub-and-spoke topology and service oriented architecture to facilitate the easy construction of robust and efficient spoken dialogue applications. The platform supports state-based and frame-based/plan-based dialogue techniques, user-initiative, system-initiative and mixed-initiative dialgoue strategies, multi-topic and topic changing management, confirmation, database query, etc. Compared with other XML-based dialgoue system, the current dialgoue framework empowered by a full-fledged XML script engine which supports scoped variables, arrays, list, string handling, expression, if/else statement, for loop, functions, timer, socket communication, file I/O, as well as pipeline information flow programming. In addition, the platform is domain independent, lightweight and does not require any other supporting software or environment setting. Most importantly, it can be enhanced by new plug-in development with API provided by the framework. Our practice on the receptionist robot showed that the development cycle for a new spoken dialogue system can be greatly reduced by its nature of configurability and component reusability as well as the powerful XML-based dialogue engine. The new dialogue system based on the platform is robust because users only need to deal with XML script for task modeling, dialogue move control and knowledge representation. No any programming effort is needed on the core dialogue framework.

For the future work, the communication middleware can be enhanced to support more protocols. In the mean time we also intent to develop more plug-ins to increase the pool of dialogue components. The inspiring functions of the components include robust information retrieval for robot, interface to make use of different types of knowledge base and ontology.

ACKNOWLEDGMENT

The research described in this paper is the result of collaborative effort of colleagues in the Lab of Agency for Science, Technology and Research (A*STAR) Social Robotics (ASORO) of Singapore and Mr. Yen-Lu Chow from WholeTree Technologies, their contributions are gratefully acknowledged.

REFERENCES

[1] MICHAEL F. MCTEAR, "Spoken Dialogue Technology: Enabling the Conversational User Interface", *ACM Computing Surveys*, Vol. 34, No. 1, March 2002, pp. 90–169.

[2] Terrence Fong, Illah Nourbakhsh, Kerstin Dautenhahn: A survey of socially interactive robots, *Robotics and Autonomous Systems* 42 (2003) 143–166.

[3] Enzo Mumolo, Massimiliano Nolich and Gianni Vercelli, An Interactive Receptionist Robot for Users with Vocal Pathologies, *Proceedings of the 2007 IEEE 10th International Conference on Rehabilitation Robotics*, June 12-15, Noordwijk, The Netherlands.

[4] Nisimura, R. Uchida, T. Lee, A. Saruwatari, H. Shikano, K. Matsumoto, Y., ASKA: receptionist robot with speech dialogue system, *Proceedings of the 2002 IEEE/RSJ, International Conference on Intelligent Robots and System*, Lausanne Seitzerland, 1314- 1319 vol.2.

[5] Illah R. Nourbakhsh, Clay Kunz, Thomas Willeke, The Mobot Museum Robot Installations: A Five Year Experiment, *International Conference on Intelligent Robots and Systems*, (pp. 3636 - 3641). Las Vegas, NV: IEEE, 2003.

[6] Galaxy Communicator Documentation: *http://communicator.sourceforge.net/sites/MITRE/distributions/GalaxyCommunicator/docs/manual/index.html*.

[7] Seneff, S., Hurley, E., Lau, R., Pao, C., Schmid, P. and Zue, V, "Galaxy-II: A Reference Architecture for Conversational System Development," *Proceedings of the 5th International Conference on Spoken Language Processing* (ICSLP '98), pp. 931-934, Sydney, Australia, December, 1998.

[8] Dan Bohus, Antoine Raux, Thomas K. Harris, Maxine Eskenazi, Alexander I. Rudnicky, Olympus: an open-source framework for conversational spoken language interface research. *Proceedings of HLT-NAACL 2007 workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technology (2007)*.

[9] Levin, E., Narayanan, S., Pieraccini, R., Biatov, K., Bocchieri, E., Di Fabbrizio, G., Eckert, W., Lee, S., Pokrovsky, A., Rahim, M., Ruscitti, P., and Walker, M The AT&T-DARPA Communicator mixed-initiative spoken dialog system, *Proceedings of the Sixth International Conference on Spoken Language Processing* (ICSLP'2000), Beijing, China, pp. 122-125, 2000.

[10] Staffan Larsson, David R Traum, Information state and dialogue management in the TRINDI dialogue move engine toolkit, *Natural Language Engineering*, Volum 6, Issue 3-4, 323 – 340, 2000.

[11] J. Bos, E. Klein, O. Lemon, T. Oka, "DIPPER: Description and Formalisation of an Information-State Update Dialogue System Architecture", *Proceedings of the Fourth SIGdial Workshop of Discourse and Dialogue*, 2003.

[12] Peter Ljunglöf, Trindikit.py : An open-source Python library for developing ISU-based dialogue systems. In *Proceedings IWSDS'09, 1st International Workshop on Spoken Dialogue Systems Technology Workshop*, Kloster Irsee, Germany, 2009.

[13] S.W. Hamerich, Y.H. Wang, V. Schubert, V. Schless and S. Igel, "XML-Based Dialogue Descriptions in the GEMINI Project". *Proceedings of the "Berliner XML-Tage 2003*, Germany, pp. 404-412.

[14] Ridong Jiang, Yeow Kee Tan, Haizhou Li, Chern Yuen Wong Anthony, Dilip Kumar Limbu, Development of Event-Driven Dialogue System for Social Mobile Robot, *Global Congress on Intelligent Systems (GCIS 2009)*, 117~121, Xiamen, China.

[15] David Schlangen, Modelling dialogue: Challenges and Approaches, *Künstliche Intelligenz. (the Journal of the Section on Artificial Intelligence of the German Association for Computer Science)*; 3/05; pp.23--28; July 2005

[16] Michael F. McTear. *Spoken Dialogue Technology*. Springer Verlag, London, Berlin, 2004.

[17] Por, L.T.C. Tay, A. Limbu, D.K., Olivia 2.0 @ TechFest 09: Receptionist robot impressed visitors with lively interactions, *Human-Robot Interaction* (HRI), 2010 5th ACM/IEEE International Conference, 351-351. Osaka, Japan.

[18] ASORO (A*STAR Social Robotics) OLIVIA Robot: *http://www.asoro.a-star.edu.sg/robots_olivia.html*