ISIS: A Multi-Modal, Trilingual, Distributed Spoken Dialog System developed with CORBA, Java, XML and KQML

Helen Meng[1], P. C. Ching[2], Yee Fong Wong [1] and Cheong Chat Chan[1]
[1]Human-Computer Communications Laboratory, [2]Digital Signal Processing Laboratory,
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong SAR, China
{hmmeng@se.cuhk.edu.hk}

Abstract

ISIS (Intelligent Speech for Information Systems) is a trilingual spoken dialog system in the stocks domain. It supports the three languages commonly used in Hong Kong (Cantonese, Putonghua and English), and serves as a test-bed for our research in various speech and language technologies. ISIS also features combined interaction and delegation dialogs, and automatic assimilation of newly listed stock names into the system's knowledge base. This paper focuses on the architecture and multi-modality of ISIS. We use the CORBA middleware to implement a distributed system that is interoperable across platforms. We also describe the incorporation of KQML (Knowledge Query and Manipulation Language) software agents in ISIS to handle delegation dialogs. The latest enhancement supports multi-modal and mixed-modal input which suit the natural affordances of certain interactions in order to improve usability. Input modalities include speaking, typing or mouse-clicking. Output media include synthesized speech, text, tables and graphics.

1. Introduction

Over the past two years our group has been developing a trilingual, distributed spoken dialog system known as ISIS (Intelligent Speech for Information Systems). The ISIS knowledge base is restricted to the stocks domain, and the system serves as a test-bed for our research and development in speech and language component technologies, which includes trilingual speech recognition, biliteral natural language understanding, trilingual speech generation, speaker authentication and dialog management that combines online interaction with offline delegation dialogs. The latest enhancement allows the handling of multi-modal and mixed-modal input.

Details regarding the components can be found in [1,2]. These component technologies run on different platforms. Integrating them into an end-to-end system presents a challenge in distributed system architecture design and development. Previous work in this area includes [3,4]. For ISIS, we have chosen a unique blend of number of Internet technologies with attractive properties – (i) CORBA offers location transparency, interoperability and scalability; (ii) Java offers an object infrastructure with platform independence, and Java Applets are used to achieve multimodality; (iii) XML offers clear language semantics for communication between objects; and (iv) KQML supports implementation of software agents for offline delegation dialogs.

A reference implementation of ISIS that includes software, documentation and an online demo is available from our website (http://www.se.cuhk.edu.hk/~isis/download).

2. The ISIS Architecture – CORBA

CORBA (Common Object Request Broker Architecture) is a middleware with specifications produced by OMG (Object Management Group). It provides ease and flexibility for distributing components. CORBA provides the ORB (Object Request Broker) that handles communication between objects, including object location, request routing and result returning.

Figure 1 illustrates the ISIS client/server architecture, which includes six server objects, another server object encapsulating KQML agents, together with the client object. Some are implemented in Java or C on UNIX; others in Visual C++ on Windows NT. These server objects extend the stubs/skeletons (i.e. the glue to the ORB from the client/server) to the core speech and language engines. The objects can communicate with each other via the intranet or the Internet using IIOP (Internet InterORB Protocol).

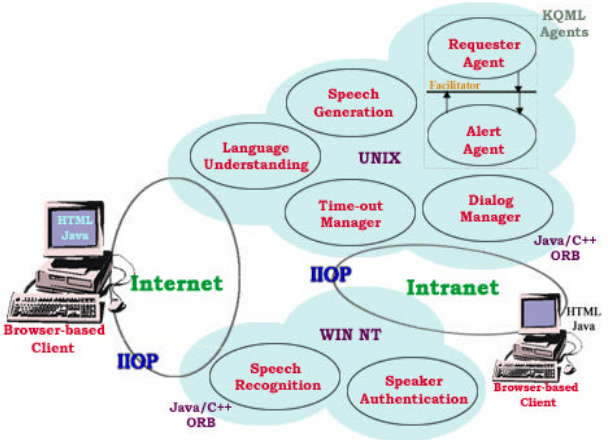

Figure 1. The ISIS Architecture.

2.1 Location Transparency

Consider the case of a socket-based implementation – the server needs to start a listener at a given port, e.g.,[1]

```
ServerSocket serverSocket = new ServerSocket (4410);
```

and then establishes a connection with the client to receive input. Similarly, the client needs to start a connection with the server at a specified host and port, e.g.,

```
Socket socket = new socket (" pc3.se.cuhk.edu.hk" , 4410);
```

and then establishes a connection with the server to send input. Hence the implementation needs to explicitly manage the host/port for every server in the system.

Contrastively, CORBA offers the desirable feature of *location transparency* to ISIS. No host/port information is needed explicitly. Only the names of the server objects need to be known for two-way (receiving/sending) communication (see Figure 2 for an example).

[1] Examples of Java code fragments.

```
//CORBA-based implementation of a server (e.g. NLU)
public class corbaserverUnderstanding{…
    //activate the server manager named corbaserverUnderstanding
    recognizerManagerPOA.activate_object_with_id
    ("corbaserverUnderstanding".getBytes(),
    recognizerManagerServant);
}
```

```
//CORBA-based implementation of the client
public class corbaclient{…
    //locate an NLU server manager named corbaserverUnderstanding
   Server.ServerManager manager =
    Server.ServerManagerHelper.bind(orb,
    "/server_agent_poa",
    "corbaserverUnderstanding".getBytes());
…
   //sends a string to the NLU server
    cse.send2corbaserver("testing"); ..}
}
```

Figure 2. Java code fragments from a CORBA-based implementation of the natural language understanding (NLU) server and the client. Only the name of the server needs to be known for communication. There is no need to manage the corresponding host or port information.

2.2 Interoperability

CORBA enables distributed object applications to interoperate across platform through the network, by providing the IDL (Interface Definition Language) to communicate with different programming languages running on multiple operating systems. Hence, we can abstract away from hardware characteristics. IDL is a declarative language and can be used to define modules, interfaces, data structures, etc. Figure 3 shows an IDL of the module named ISIS.

```
module ISIS {
    interface Recognizer {  (1)
      void send2recognizer(in string str, in bytearray byte);};
    interface RecognizerManager {  (2)
       void registerClientHandler(in Client ch);};
…}
```

Figure 3. Example of the ISIS IDL showing the Recognizer server object. send2recognizer has two arguments, one corresponds to the XML message and the other to the .wav file.

Since the Recognizer server object is implemented in C++, the IDL in Figure 3 is compiled by idl2cpp into an object class also named "ISIS". The Recognizer interface (see label #1 in Figure 3) provides a single member function for receiving incoming messages. The RecognizerManager (see label #2 in Figure 3) creates a recognizer object instance for each specific client session and exchanges it with the client object reference. Compilation produces isis_c.hh and isis_c.cpp, which are internal definitions and *client stub routines* for the Recognizer and RecognizerManager classes to build client applications. Compilation also produces isis_s.hh and isis_s.cpp, which are the internal definitions and *server skeleton routines*. Hence the Recognizer server object can function as a client or server.

For other server objects implemented in Java, we compile the IDL by idl2java, and the subsequent processes remain similar to those described above.

2.3 Scalability

CORBA also offers a scalable architecture, where a new class can be added to the system simply by adding its corresponding interface definition to the IDL, followed by recompilation. For example, to augment ISIS with speaker verification, we may add the following to the ISIS IDL in Figure 3:

```
interface SpeakerVerification {..};
interface SpeakerVerificationManager {..};
```

2.4 Quality of Service

CORBA also offers QoS(Quality of Service) which defines and manages the connection between the client and server objects. If the client encounters a communication problem with a server object, which may be caused by the termination of a server or server reboot, a rebind attempt will be made automatically if invocation is retried.

3. Delegation to KQML Software Agents

ISIS supports asynchronous human-computer interaction in terms of offline delegation. The system can launch a software agent to monitor the dynamic financial feed on the user's behalf. When the user's pre-specified condition is met, the software agent sends an alert message back to the user. The software agents are implemented in KQML (Knowledge Query Manipulation Language) [5]. KQML is both a message-format language and a message-handling protocol for agent-to-agent communication. It provides run-time information exchange and knowledge sharing among agents. The ISIS implementation uses JKQML that is entirely in Java. The integration between KQML and CORBA is consistent with the methods described in the previous section.

There are three KQML software agents in ISIS – the Requester Agent, Facilitator and Alert Agent. If a user's requested transaction (e.g. "Buy three lots of HSBC at eighty nine dollars please") cannot go through due to a mismatch between the requested and market prices, ISIS will trigger the offline delegation procedures. First, a non-blocking XML message is sent from the dialog manager server to the Requester Agent (see Figure 4).

```
<DIALOG_MANAGER>
   <TRIGGER LANG =English STATUS =launchAgent>
   <USER_ID>005</ USER_ID>     <ACTION>buy</ ACTION>
   <MARKET>95.0</ MARKET>     <LOTS>3</ LOTS >
   <STOCK>0005.HK</ STOCK>    <SHARE>---</ SHARE>
   <PRICE >89</ PRICE >
   <TIME_STAMP >Aug_02_2001_14:00:38</TIME_STAMP >
   </TRIGGER>
</DIALOG_MANAGER>
```

Figure 4. An example XML message sent by the dialog manager server to the Requester Agent.

The Requester Agent receives this XML message, decodes it and transmits a corresponding KQML message (see Figure 5) to the Facilitator. The Facilitator is a software substrate for agent-to-agent communication, as it maintains a registry of all agents. In Figure 5, ASK-ALL is a performative (speech act) to request for service from all agents. The :SENDER and :RECEIVER fields constitute the communication layer. The :LANGUAGE field specifies the format of the :CONTENT parameter. The :ONTOLOGY field specifies the set of term definitions used in the :CONTENT parameter.

The Facilitator receives the KQML message, interprets it and stores the request in a database of similar requests. The Alert Agent keeps track of these requests and monitors the real-time financial information feed accordingly. If the pre-specified condition is met (i.e. HSBC's market price hits 89 dollars per share), the Alert Agent will send a KQML message (see Figure 6) through the Facilitator to alert the Requester Agent. The performative TELL is the expected response to ASK-ALL. Hence the Facilitator knows to re-route this KQML message back to the Requester Agent.

```
(ASK-ALL
: SENDER requester agent        : RECEIVER interpreter
: REPLY-WITH messageID
: LANGUAGE xml                  : ONTOLOGY insertion
: CONTENT (theaction: buy  theuser_id: 005
           theprice: 89  thetimestamp: Aug 02 2001 14:00:38
           theric: 0005.HK  thelot: 3  theshare:---))
```

Figure 5. An example KQML message sent by the Requester Agent to the Facilitator.

```
(TELL
: SENDER alert agent        : RECEIVER interpreter
: REPLY-WITH messageID
: LANGUAGE xml              : ONTOLOGY insertion
: CONTENT (theaction: buy  theuser_id: 005  themarket: 89
           theprice: 89  thetimestamp: Aug 02 2001 14:00:38
           theric: 0005.HK  thelot: 3  theshare:---
           theresponse: alert!))
```

Figure 6. An example KQML message sent by the Alert Agent to the Facilitator.

In the final step, the Requester Agent returns a KQML alert message (see Figure 7) to the dialog manager server.

```
<KQML_AGENT>
   <MARKET>89</MARKET>
   <TIME_STAMP>Aug_02_2001_15:37:02</TIME_STAMP>
   ..(other parameters identical to those in Figure 4)
</KQML_AGENT>
```

Figure 7. An example XML message sent by the Requester Agent to the dialog manager server.

4. Multi-modality in ISIS

The ISIS client is implemented as a Java applet, and can be viewed by an applet viewer or a web browser with Java plugin by providing a URL. The client incorporates support for multi-modal interactions. More specifically, the system can accept user input in the form of speech, typed text or mouse clicks. The system also presents output to the user in the form of synthesized speech as well as textual, graphical and tabular displays. Multi-modal I/O involves nine java classes, such as:

ClientApplet.java: starts and maintains the connection with the speech and language server objects;

ClientImpl.java: message passing via JPanelISIS.java (see below) for textual input, captures input mouse clicks and displays text/graphical outputs;

RecordPlayback.java: records input speech and plays back the synthesized speech using Java Sound API;

EndPointDetection.java: uses Java Sound API, and detects the start and end points of the speech input based on energy, zero-crossing rate and periodicity;

JPanelISIS.java: captures user input via text (by JText), and radio button selections (by JRadioButton).

4.1 Scenario incorporating multi-modal inputs

In the following we will traverse an example scenario to illustrate multi-modal interactions in ISIS. In this scenario, ISIS "learns" about the newly listed company, ARTEL Solutions Group Holdings Ltd (abbreviated as "ARTEL"), and incorporates the new listing into the ISIS knowledge base.

Our scenario begins with the user's spoken query,[2] "Do you have the real-time quotes of ARTEL?" Figure 8 is a screenshot of our client. Here, the radio buttons corresponding to language selection and input modality show that the user has chosen to input via spoken English. Selected radio buttons are captured by JRadioButton in JPanelISIS.java prior to every user request. Hence the user can switch freely in between languages and input modalities while maintaining a coherent conversation with ISIS.

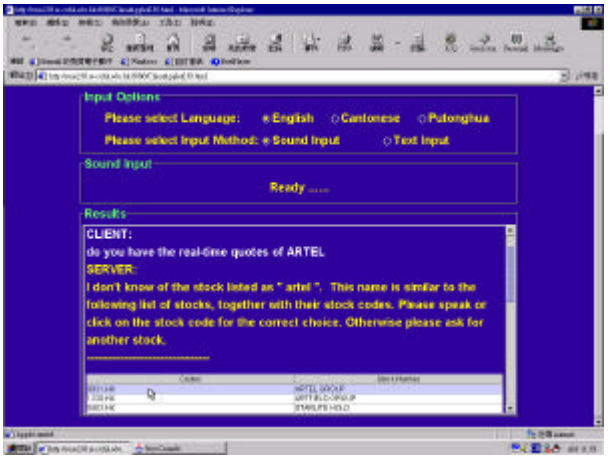

Figure 8. Screen shot of the client presenting information in response to the user's spoken input "do you have the real-time quotes of ARTEL?"

Next, the input speech recorded by the client is sent to the English recognizer, which returns the (correct) transcription in XML (see Figure 9).

```
<RECOGNIZER>
  < ENGLISH >
      do you have the real time quotes of ARTEL
  </ ENGLISH >
</ RECOGNIZER >
```

Figure 9. XML string sent from the recognizer to the client.

The client then displays this transcription as output text onscreen (also shown in Figure 8) by calling a Java class JTextPaneHelper.java that extends JTextPane.

Additionally, the speech generation server returns *two* XML responses to the client (see Figure 10). For the first XML message, the client displays the textual part via JTextPaneHelper, and the list of possible RIC names as a table via JTable. The textual message states that "ARTEL" is uknown to the ISIS system, and presents a table of possible stock names that may correspond to "ARTEL". Upon receiving the second XML message, the client retrieves the

[2] We are only handling out-of-vocabulary (OOV) words starting at the NLU level. Recognition of OOV will be addressed at a later stage.

synthesized speech wave from the specified URL, and plays it for the user by invoking the class RecordPlayback.java.

First XML message:

```
<RESPONSE_GENERATION>
   <TXT>I don't know of the stock listed as ARTEL ./TXT>
   <RIC_NAME >Code-StockName | 0931.HK-ARTEL GROUP |
    1229.HK-ARTFIELD GROUP |.</RIC_NAME >
</RESPONSE_GENERATION >
```

Second XML message:

```
<RESPONSE_GENERATION>
      <SYNTH_FILE>http:// (URL) </SYNTH_FILE>
</RESPONSE_GENERATION>
```

Figure 10. Two XML strings sent from the response generation server to the client.

The synthesized speech asks the user to select the stock code corresponding to "ARTEL" either by speaking or clicking. These multi-modal interactions derive synergy to suit the natural affordances of the application to enhance usability of ISIS. As shown by the arrow in Figure 8, user clicked on the stock code "0931.HK". The client class ClientImpl.java extends ListSelectionListener (a class provided by JDK1.2). ListSelectionListener can capture the mouse action for row selection in JTable and obtain a row index from the JTable object. Contents from this row is retrieved by the client and sent to the natural language understanding (NLU) server. The NLU server extends its knowledge base to include "ARTEL" by adding two grammar rules: STOCK_NAME → ARTEL and STOCK_CODE → 0931.

4.2 Mixed-modal input

We have also begun to implement support for mixed-modal input in ISIS. Figure 11 shows a screen shot with the system's response to the user query "Show my account information." ISIS provides a table of all transactions. A possible *mixed-modal* input at this point may involve *both speech and mouse clicks*, e.g. "Show the transaction details of these two stocks <click1> <click2>."

To handle such a request, ClientImpl.java starts a thread to measure the duration of the spoken input plus three seconds beyond the detected endpoint. The recorded speech is sent to the recognizer as usual, and all mouse clicks captured within the entire duration are registered. The client extracts the row contents corresponding the mouse clicks, appends the contents to the recognizer's transcription, and produces an *integrated* XML message (see Figure 12 for an example) based on the mixed-modal input. The message is sent to the NLU server.

```
<CLIENT>
     <ENGLISH>Show me the transaction details of these two
stocks.  zero zero six six .HK.  one one two two .HK.
     </ENGLISH>
</CLIENT>
```

Figure 12. Example of an integrated XML message incorporating mixed-modal input information.

5. Conclusions and Future Work

This paper presents the design and development of ISIS, a trilingual spoken dialog system for the stocks domain. We describe the use of a suite of Internet technologies to develop a multi-modal, distributed system. The Internet technologies include:

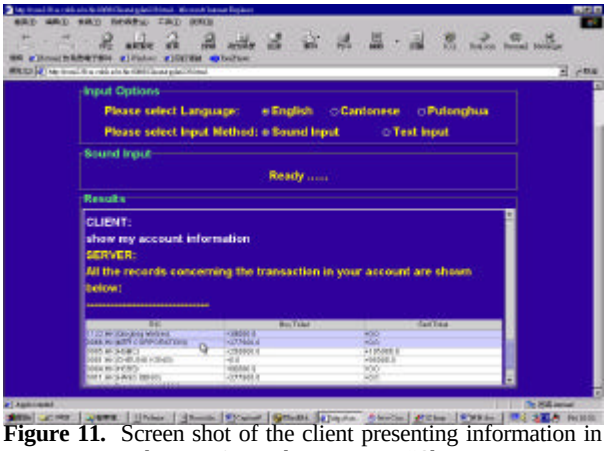

Figure 11. Screen shot of the client presenting information in response to the user's spoken input, "Show my account information."

(i) CORBA, which offers location transparency, interoperability and scalability; (ii) Java, which offers an object infrastructure with platform independence; (iii) XML, which offers clear language semantics for communication between objects; and (iv) KQML, which supports implementation of software agents for offline delegation dialogs. We also describe system implementation that supports multi-modal and mixed-modal input, which suit the natural affordances of certain domain-specific interactions and enhance the usability of ISIS. A possible future direction is to migrate ISIS towards a Web services model, e.g. migrating the ORB towards UDDI and IDL towards WSDL/SOAP, to increase accessibility to a wide range of users.

6. Acknowledgments

This research is supported by the Joint Center for Intelligence Engineering between Peking University and The Chinese University of Hong Kong. We thank the past and present members of the Human-Computer Communications Laboratory and Digital Signal Processing Laboratory of CUHK, as well as the National Key Laboratory for Machine Perception of PKU. In particular, we thank Professor Hui-Sheng Chi, Professor Ke Chen, Professor Tan Lee and Ms. Lan Wang for their participation in this project. We are grateful to Reuters Hong Kong for donating their satellite feed to support our research.

7. REFERENCES

[1] Meng H., et al., "ISIS: A Multilingual Spoken Dialog System developed with CORBA and KQML agents," Proceedings of ICSLP, 2000.

[2] Meng, H. et al., "ISIS: A Learning System with Combined Interaction and Delegation Dialogs" Proceedings of Eurospeech, September 2001.

[3] Cohen, P. et al., "An Open Agent Architecture," in AAAI Spring Symposium, March 1994.

[4] Bayer S. et al., "Exploring Speech-Enabled Dialog with the GALAXY Communicator Infrastructure," Proceedings of HLT Conference, 2001.

[5] Finin, T. et al, "KQML – A Language and Protocol for Knowledge and Information Exchange," Technical Report CS-94-02, Univ. of Maryland, UMBC .