# Implementation of Simultaneous Coordinated Multimodality for Mobile Terminals

**Narada Dilp Warakagoda**
Telenor R&D, Snarøyveien 30
1331, Fornebu
Norway
narada-dilp.warakagoda@telenor.com

**Jan Eikeset Knudsen**
Telenor R&D, Snarøyveien 30
1331, Fornebu
Norway
Jan-eikeset.knudsen@telenor.com

**Anders Smeby Lium**,
Norwegian University of Science and Technology,
Norway
lium@stud.ntnu.no

**Abstract**

In this paper we will describe a platform for speech centric multimodal dialog applications which supports simultaneous coordinated multimodality. The system consists of a light-weight client usually running on a mobile device and a server placed in the network. The client offers a HTML/Javascript based graphical interface and a voice interface to the user. Both the client and server are described with their architecture, functionality and behavior. In particular we will show how the system has been constructed to support the simultaneous coordinated multimodality that allows a maximum freedom for the user.

## 1 Introduction

Speech centric multimodality is often proposed as a solution for the usability problems of the traditional interfaces to the information and communication systems of ever increasing complexity. The source of most of the advantages of multimodality over speech only or traditional graphics/pointing only systems is the possibility that the strengths of one modality can compensate for the weaknesses of another modality. For example, speech can be used to construct highly complex inputs easily, something which is much more difficult with pointing on menus or a soft keyboard. On the other hand, speech is less suitable for presenting long descriptive information to the user, whereas graphics is more appropriate for such tasks. But the most interesting possibility here is to combine different modalities together to form a single composite input. W3C has tried to classify these different types of multimodality in a systematic way (Hickey, 2000). According to this, there are three main classes of multimodality:

*Sequential* :- One modality at a time is available

*Simultaneous Uncoordinated*:- Several modalities are available but only one modality is interpreted in a given dialog state.

*Simultaneous Coordinated*:- Several modalities are available and they are interpreted together in a given dialog state.

Obviously the third class of multimodality above is the most advanced type and it gives the user the maximum freedom. The famous article titled “put that

there" first introduced this kind of multimodality (Bolt, 1980). Even though the technology has advanced significantly since then, the level of ambition for the nature and context of such applications has also increased proportionately. For example, nowadays there is a huge interest in multimodality in connection with small terminals such as PDAs which communicate with servers over relatively high capacity wireless networks. This distributed nature is one of the sources of challenges in building mobile multimodal systems. In particular, predictable and unpredictable time delays in data transmission and processing constitute an important issue in implementation of real-time simultaneous coordinated multimodal systems. In this paper, we will describe the implementation details of a distributed multimodal platform for small mobile terminals. We are particularly concerned of the realization of the functionality of simultaneous coordinated multimodality, both in modality fusion and dialog management. The implementation, to a large extent, is based on commercially or otherwise available and more or less standard technology. Further, we have tried to maintain a clean separation between the platform and the applications the platform will host. This makes the platform open for new applications. In addition, the platform has been designed in such a way that different types of on-line and off-line adaptations are possible. Especially, adaptation to the nature or capabilities of the terminal and the network has been given a careful consideration.

This paper is organized as follows. In section 2 the overall architecture is presented. Details of the modules that handle the voice and graphics modalities are described in section 3. Multimodal integration and dialog management is the subject of section 4. Finally in section 5 we make some concluding remarks.

## 2 Architecture

The system presented in this paper is an improved version of the MUST system (Almeida et al., 2002). This is based on the Galaxy communicator (Galaxy Communicator, n.d.) and thus has a hub-spoke type architecture as shown in Figure 1. As seen from this figure, the server part of the system consists of five separate modules which can communicate with one another through the central facilitator module "hub". The hub and inter-module communication facility are provided by the Galaxy communicator infrastructure. Other modules in the server, except for the voice server which is written in C++, are custom made using Java. The communication among the modules are based on messages, each of which consists of a set of attribute-value pairs. Note that all the server side modules can be run even on a moderately powerful PC. In our experiments we used a PC with a Pentium II, 450 MHz processor and 128 MB RAM. The client part of the system consists of two main components. These components are the counterparts of the server-modules handling voice and GUI modalities. The client is usually run on a PDA having Windows CE/PocketPC 2002 operating system. However, it can also be run on a usual portable or stationary PC with Windows 2000 operating system.

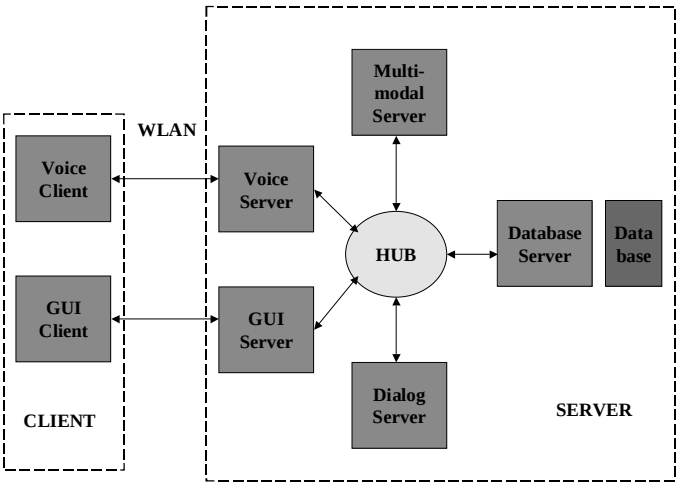


Figure 1: Architecture of the system.

The information flow through the system in a typical user query which contains both voice and pointing inputs is shown in Figure 2. As seen from this figure, the client collects the pointing and voice signals and transfers to the server over the wireless connection, which in our case is a wireless local area network (WLAN) based on the IEEE 802.11b protocol. The GUI server and voice server collect these signals respectively. While the GUI server annotates the pointing signal in a suitable way, voice server performs a speech recognition operation to extract the concepts carried by the speech signal.

Then both the voice server and GUI server pass the concept values further to the multimodal server. The purpose of the multimodal server is to combine the information coming from the voice server and the GUI server. It uses a simple timer mechanism to bundle all the information that arrive in during a predefined time window. In that sense it does not perform a complete multimodal integration, but only a partial integration based on temporal information.

The combined information are passed to the dialog manager which actually completes the multimodal integration process and interprets it to perform the necessary action depending on the current dialog state. In a typical situation the appropriate action would be to contact the database server with a request for certain information. The database server is merely a wrapper to a SQL-compatible database. Therefore, its task is to convert the request from dialog manager to a well formed SQL-query, perform the database-lookup and send back the result back to the dialog manager. Note that the database server returns information in XML format, and this includes only the names of the involved image files. The actual image files are stored in the GUI server.

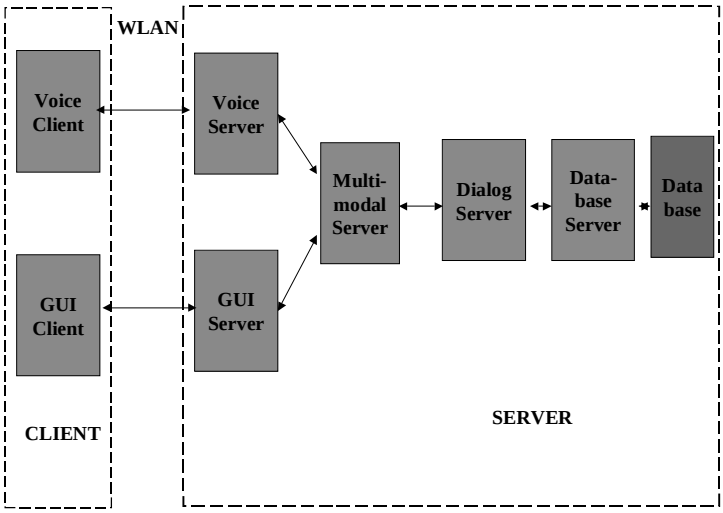


Figure 2: Information flow through the system

Dialog server processes the result from the database server, in order to make it more human understandable, and send further to the multimodal server. In this case, the multimodal server performs a simple "fission" operation by splitting the feedback from the dialog server into a voice part and a GUI part. These parts are then sent further to the voice client and the GUI-client, where these are presented to the user.

# 3 Voice and GUI Subsystems

## 3.1 Voice subsystem

The voice subsystem consists of voice-client running on the mobile terminal and the voice server running on the server side. The client-server separation of the voice subsystem makes it possible to be less dependent on the computational capability of the mobile terminal.

Since the terminals we mainly target our service for, such as PDAs, do not usually possess a high computational capacity, the voice client has been made very simple. From the input side, it simply reads the audio input device of the terminal and copies the speech data to the IP socket connection between the voice client and the voice server. From the output side, it just performs exactly the opposite operation. Even though this strategy works most of the time especially with lightly loaded network connections, it is not optimal by any means. One possible improvement is to transfer voice activity detection (VAD) and feature extraction to the client from server, as proposed in distributed speech recognition (DSR homepage, n.d.). However, at this time this is not possible because we use Philips ASR engine (SpeechPearl homepage, n.d.) which is a closed system and

not compliant with the DSR-standard. Another improvement which we could have tried was the transfer of time-stamping to the client. This is very important for multimodal integration, especially when the network introduces unpredictable delays to the speech signal.

Voice server, on the other hand, contains all the heavy components; namely the automatic speech recognition (ASR) engine, text-to-speech (TTS) engine, in addition to the VAD and the handler of socket connection with the voice client. As mentioned earlier the ASR engine is based on the Philips SpeechPearl, and the TTS can make use of any Microsoft SAPI-4.0 compliant system (SAPI homepage, n.d.). Note that each of the four components mentioned above run in its own thread, making asynchronous operation possible and contributing highly to the real time operation.

The operation of the voice server is as follows. When an input signal is received through the socket connection it is tested by VAD on a frame-by-frame basis. If speech activity is detected, then the signal is fed to the ASR engine, otherwise the signal is discarded. The ASR-engine operates in the so called "open grammar" mode, meaning that the ultimate result of the ASR operation consists of a set of concepts, their values and corresponding confidence scores. For example, a speech signal corresponding to "I would like to travel from here to Lysaker at two o'clock in the evening" would result in a structure like:

$$\begin{bmatrix} ACTION & travel & 600 \\ TO\_STATION & Lysa\,\text{ker} & 700 \\ DEPARTURE\_TIME & 1400 & 900 \end{bmatrix}$$

The voice server then packs this structure in a Galaxy-compliant message and sends further to the multi-modal server. On the other hand, when a message from the multimodal server is received, the voice server passes the intended text string to the TTS-engine to generate a speech waveform, which in turn is copied to the socket connection with the client.

### 3.2 GUI subsystem

The GUI subsystem consists of the the GUI-client on the mobile terminal and the GUI-server on the application server. Figure 3 shows the GUI subsystem.

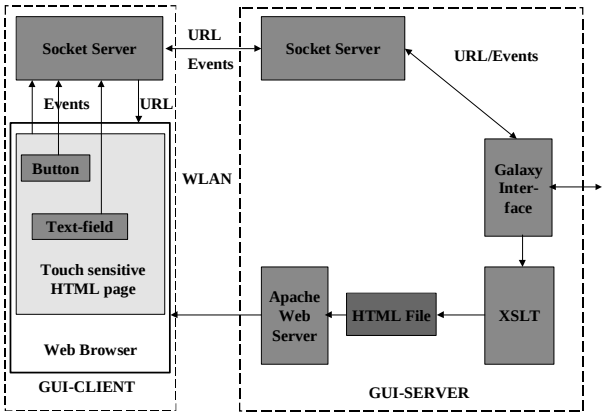


Figure 3: GUI-subsystem

The GUI-client consists of a Web Browser (Internet Explorer) and an IP socket server. The web browser uses specially developed ActiveX controls to construct usual GUI-components such as buttons, text areas and lists etc. or to make the whole web page a hot-spot sensitive to pointing. Therefore it is possible to programmatically collect the events generated in those components and divert them to the socket server. The socket server sends the collected events to the GUI server. It can also receive the URL of the web page to be displayed from the GUI server. Once the URL is received, the actual web page is fetched from the web server component of the GUI-server using the HTTP protocol in the usual way.

The GUI server consists of four main components. First component of this module is the interface to the Galaxy compliant messages. When such a messages arrives, this interface receives it and passes the definition of the graphical output in a device independent XML format to the extensible style sheet transformer (XSLT). At the same time, the Galaxy interface generates a suitable URL and passes it to the socket server which sends it to the GUI client. The XSLT in the meantime converts the XML page to a HTML file and saves in accordance with the generated URL, in the document area of the web server. Then the GUI client fetches this as described earlier and sends back user generated events, such as tapping on a button, selecting an item of a list etc.. The socket server collects them and sends further to the rest of the system via the Galaxy interface.

## 4 Multimodal Integration and Dialog Management

In literature there are several different strategies for multimodal integration (Johnston et al., 1997; Oviatt et al. 2000; Wu et al. 1999). But initially we have chosen a simple integration process which is distributed over two modules; dialog server and multimodal server.

### 4.1 Multimodal Server

The integration algorithm implemented in the multimodal server is specifically targeted for inputs that contain a maximum of two pen inputs and a simultaneous speech input. For example, saying "I will travel from here to there" and at the same time pointing at two locations on a displayed map will generate such an input. The integration algorithm in a given dialog state is as follows. As soon as an input from the voice server or the GUI server arrives at the multimodal server, it starts a timer set to a predetermined time. Then it waits until the timer expires, unless a total of two GUI tapping inputs and a voice input arrive before that, in which case it exits waiting immediately. At the end, the multimodal server bundles the information collected during the waiting period, and sends the result to the dialog server. Note that this is only a temporal integration and no consideration is given to the semantics of the inputs.

### 4.2 Dialog Server

The dialog manager server consists of four main components that were implemented as separate classes: Context Manager, User input processor, System response generator and XML processor

*Context manager* is the basic framework of the whole module. It is a finite state machine that contains several states possibly having physical interpretations. Each of these states defines a particular context in the dialog. That is the reason why this state machine is called context manager. When the user generates an event, a state transition can occur. The route of the transition is dependent upon both the current user input $I_t$ and the current state $S_t$ . Each state transition will trigger an action set such as looking up a database to generate an output $O_t$. A chain of such state transitions defines the dialog-flow.

*User input processor* basically operates on a table which has two columns; one for the concepts and the other for the corresponding values of the concepts. The concept table is filled using the values coming from the multimodal server after its temporal integration. During the filling operation, input ambiguities are solved, in this way completing a late fusion. Once filled, the concept table defines the current input $I_t$: If the values in the concept table are $I_t(1)$, $I_t(2)$,…….$I_t(n)$, then the N-tuple ($I_t(1)$, $I_t(2)$, $I_t(n)$) is the current input $I_t$. The number of different inputs can be prohibitively large, even if the length of the concept table (M) and the number of values a given concept can take (K) is moderate. In our case we have reduced the number of inputs by employing a many-to-one mapping from the original input space to a new smaller sized input space. This mapping is analysis based and therefore not optimal. If one had enough amount of data, the mapping could have been trained.

*System response generator* is responsible for the generating $O_t$. It is essentially a mapping from the space formed by the tuples ($S_t$, $I_t$). It looks at the current state $S_t$ and the input $I_t$, and generates an output $O_t$ that contains both speech and graphic contents. The output can contain three different components:

pre-stored state dependent strings, parameters extracted from the input it self and data obtained from the back-end database. Speech output is generated simply by concatenating these components appropriately. Graphical output is generated as an XML string. Since it is difficult to generate an XML string through simple concatenations, it is maintained a DOM (Document Object Model) tree which always represents the (graphical component of the) current output. The current tree is generated from the previous DOM-tree by tree-operations such as deletions and insertions. The graphical output is then obtained by converting the DOM tree to a string. Actually, these XML operations are performed using the fourth class mentioned above, *the XML processor*.

### 4.3 Combining Multimodal Integration and Dialog Management

The multimodal integration strategy described in subsection 4.1 has one main disadvantage. Namely, this strategy does not represent a full integration in the sense that a final concept set and its values are not extracted and processed to represent the information arrived through all modalities. Actually, this is completed later in the dialog server itself. It is important to consider both temporal and semantic information simultaneously to achieve a good alignment between different modalities. This is especially true, because the above procedure is entirely dependent on the arrival times of the information through different modalities. But alignment based on arrival times alone is not reliable because of the numerous network and processing delays as well as the fact that voice is a not a "point event" in time whereas tapping is nearly such an event. However, all those problems can be solved to a greater extent if one can combine semantic and temporal considerations, by either embedding multimodal integration in the dialog manger itself or making available the information about the current dialog state to the multimodal server. From an implementation point of view the first alternative is more convenient as it requires a simpler housekeeping and message passing strategy. Even though it goes against the principle of modularity, we decided to try it out.

Our implementation of the combined multimodal integration and dialog management is based on an extension of the finite state machine (FSM) representation of the latter. A typical fragment of the extended FSM is shown in Figure 4.

In Figure 4, states S1,S2,S3, and S4 are "normal" states, in which the system waits for an input indefinitely. The modification of the FSM is mainly due to the introduction of the states which are associated with timers; i.e. waiting for an input in such a state is time limited. While in a such a state, if an input arrives within the associated time-window and it is from a valid modality, the system jumps to the next state to continue the process. Otherwise the system tries to interpret the information arrived so far, and depending on the result of this, it either jumps to a "normal" state or an error handling state represented by small circles. Note that the FSM fragment shown in Figure 4, has three different paths extending from a "normal" state to another "normal" state. These three paths

correspond to the different combinations of modalities for a valid query, namely two tapping inputs and a speech input.

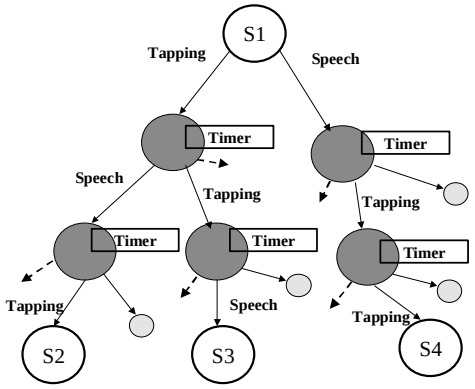


Figure 4: FSM fragment for dialog management combined with multimodal integration

As an example consider a composite user input which is comprised of two pen tappings and a speech component. If the order of arrival of these components at the multimodal integrator is “pen speech pen” and their arrival times lie within the timer values, then the system jumps from state S1 to S2. However, if their arrival order is “pen pen speech”, then the system’s final state would be S3. The system can even jump to S4 from S1 in the case where the order of arrival is “speech pen pen”.

Another example is that the user input contains only one pen tapping and a speech component. If their arrival order is “speech pen”, the system will be stuck at the right-bottom timer state. In this case, the system can try to interpret the these two input components together. If the interpretation makes any sense then the necessary action set is triggered and the system goes to another waiting state. Otherwise, the system will jump to an error handling state.

One disadvantage of this approach the high number of states. But, for an application of a moderate size, the number of states is manageable without any major difficulty.

## 5 Concluding Remarks

The multimodal system described in this paper, has been tested within the corporate local area network (LAN). Within this infrastructure the system works well and exhibits a near real time responsiveness. However, it is yet to be tested through a network path that goes beyond the corporate LAN. In that case we expect increased delays and therefore we may need more robust strategies for multimodal integration. Another issue which may need attention under highly

unpredictable network delays is the multimodal fission. Currently the information sent to the terminal are not tested for synchronism. But this has to be changed for obtaining an enhanced robustness.

However, for the time being, the main source of errors in the system is the automatic speech recognition. For example, wrong results from ASR causes the system to jump to inappropriate states causing many instabilities. Improving the dialog management can help reduce such problems.

Even though we have maintained a fairly good separation between the platform and applications, still the effort required to build a new application is not low. This is due to fact that there is no script-like API available for the development of applications. We plan to build an overlay which provides this facility of dialog specification using for example XML.

## 6 References


Almeida, L. et al., 2002. The MUST guide to Paris, in: *Proceedings of ISCA Tutorial and research workshop on Multi-Modal Dialogue in Mobile Environments, IDS'2002*, pp 49-51, Kloster Irsee, Germany.

Bolt, R., 1980. Put That There, Voice and Gesture at the Graphics Interface, in: *Computer Graphics*, 14(3), pp 262-270, ACM Press, New York.

DSR homepage. Retrieved August 02 2003 from: *http://www.etsi.org/frameset/home.htm?/technicalactiv/DSR/dsr.htm*.

Galaxy Communicator homepage. Retrieved August 02 2003 from: *http://communicator.sourceforge.net/*.

Johnston, M., Cohen, P. R., McGee, D., Oviatt, S. L., Pittman, J. A. & Smith, I., 1997. Unification-based multimodal integration, in: *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics Press.

Oviatt, S.L., Cohen, P.R., Wu, L.,Vergo, J., Duncan, L., Suhm, B., Bers, J., Holzman, T., Winograd, T., Landay, J., Larson, J. & Ferro, D., 2000. Designing the user interface for multimodal speech and gesture applications: State-of-the-art systems and research directions, in: *Human Computer Interaction*, 15(4), ACM press, New York. 263-322

SAPI homepage. Retrieved August 02 2003 from: *http://research.microsoft.com/srg/sapi.aspx*.

SpeechPearl homepage. Retrieved August 02 2003 from: *http://www.scansoft.com/speechpearl/*

Hickey M., 2000. Multimodal requirements for Voice Mark-up languages, Retrieved August 02 2003 from: *http://www.w3.org/TR/multimodal-reqs*.

Wu, L., Oviatt, S. L., & Cohen, P. R., 1999. Multimodal integration: A statistical view, in: *IEEE Transactions on Multimedia*, 1(4), IEEE, Piscataway, New Jersey. 334-341.