User Interfaces for Anyone Anywhere

Vítor J. Sá
University of Minho (UM), Information Systems Department, Guimarães, Portugal
Computer Graphics Center (ZGDV), Visual Computing Department, Darmstadt, Germany
vitor.sa@dsi.uminho.pt / vitor.sa@zgdv.de

Abstract

In a global context of multimodal man-machine interaction, we approach a wide spectrum of fields, such as software engineering, intelligent communication and speech dialogues. This paper presents technological aspects of the shifting from the traditional desktop interfaces to more expressive, natural, flexible and portable ones, where more persons, in a greater number of situations, will be able to interact with computers. Speech appears to be one of the best forms of interaction, especially in order to support non-skilled users. Modalities such as speech, among others, tend to be very relevant to accessing information in our future society, in which mobile devices will play a preponderant role. Therefore, we are placing an emphasis on verbal communication in open environments (Java/XML) using software agent technology.

Keywords: multimodal interaction, software agents, speech technology

1. Introduction

Today, we are assisting a shifting from the traditional WIMP (Windows-Icons-Mouse-Pointers) interfaces to more expressive, natural, flexible and portable ones. To a certain extent, we can say that an interface is a necessary evil. The ideal one conducts the execution of our tasks without being noticed in its existence as an intermediary [van Dam 2000]. This permits a more generalized access to information sources. More persons, in a greater number of situations, will be able to interact with computers.

A key point for the Information Society is the improvement of the ways we interact with machines, computers or information appliances. The synergistic use of different communication channels (modalities or media) is important, but equally important or more so, is the mediation by intelligent assistants to make the interactive process really effective [Encarnação 2000].

What we are talking about are the so-called multimodal interaction systems, for which the need for 'multidisciplinary' cooperation (perception and production of natural modalities) and 'teamwork' in respect to individual technology components (e.g. computer vision, speech technology, distributed systems) is readily understandable.

In this paper, we touch on a relatively broad range of fields from a technological point of view, to introduce the important theme of multimodal interfaces. By gradually sharpening our focus, we begin to place the emphasis on verbal communication, through a philosophy of open environment (Java/XML) using software 'agent' technology.

Therefore, in section 2, we describe multimodal concepts in general, followed by the subjects that enable our particular realization. Section 3 is about software agents and, in section 4, we introduce the basic concepts of speech technology and voice dialogue representation (pointing out state-of-the-art tools). Finally, in section 5, we present some conclusions and future work.

2. Multimodal interaction

In the broad field of Human-Computer Interface or Interaction (HCI), several terms and nomenclatures are used, which also highlight several different areas of research are necessary. Intelligent user interfaces, multimedia interfaces, and Multimodal Systems are examples of this. Additionally, thinking that computer interfaces of one sort or another will potentially operate at all levels of our lives, some authors prefer the term Man-Machine Interface or Interaction.

Clarifying some nomenclatures: while a computer output presentation over multiple channels has become familiar under the designation of *multimedia*, the input channels or sources, also called input modes or modalities, are the basis of the kind of application said to support *multimodal* human-computer interaction. The next figure taken from [Schomaker 1995], depicts the taxonomy we are following. (Note the two existing loops: the intrinsic feedback as in eye-hand coordination, and the extrinsic loop imposed by the computer).

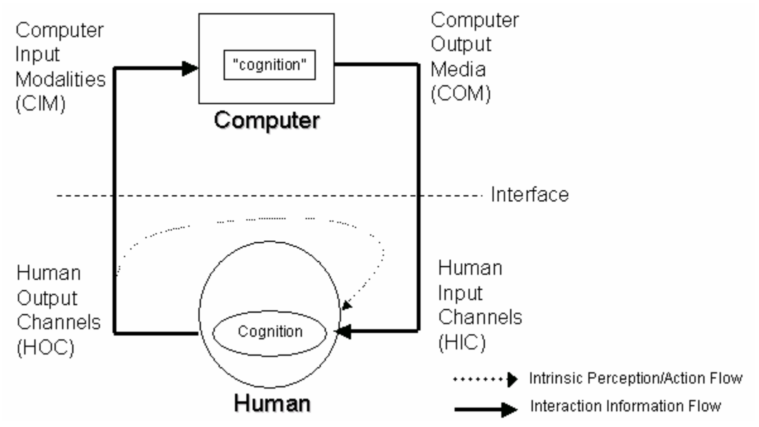
Computer Input Modalities (CIM)
"cognition"
Computer
Computer Output Media (COM)
Interface
Human Output Channels (HOC)
Cognition
Human
Human Input Channels (HIC)
Intrinsic Perception/Action Flow
Interaction Information Flow

Figure 1 – Basic model for HCI

Interaction with computers is still today quite asymmetric in respect to the input and the output. The amount of information or bandwidth that is communicated from computer to user is typically far larger than the bandwidth from user to computer [Jacob 1996]. This imbalance often influences both the intuitiveness and performance of user interaction. The goal of multimodal interaction is to contribute to the enhancement of that bandwidth in the user-to-system direction.

Therefore, multimodal systems should be able to model the information content in a high level of abstraction, in order to extract/express meaning automatically, while communicating using several channels.

2.1. Input modalities

Language use is deeply ingrained in human behavior. Therefore, speech appears to be the best form of interaction, especially in order to support non-skilled users. The challenge is to carry out a meaningful conversation with a computer as it is done with other people. Using an artificial language of special commands - or even a fairly restricted subset of natural language speech input - often fulfills user requirements. On the other hand, the closer we get to fully unrestricted natural language, the more difficulties we encounter.

Another very important modality involves facial expressions and pointing gestures. Corresponding data can be recorded in a video-based way. The resulting input mechanisms continue the trend toward naturalness and expressiveness by allowing users to perform "natural" gestures or operations and transferring them to the computer [Sá 2001]. Additional parts or characteristics of the user's body can be measured for this purpose and then interpreted as input data. Video-based input encompasses a large spectrum of modalities from *gestures*, *facial expressions* and *emotions*, to *lip-reading*, *eye tracking*, and *stick pointing*.

2.2. Output modalities

Complementary to speech recognition, speech synthesis components have to be addressed in the challenge of providing naturalness in interaction. The most difficult aspect is related to the fact that we are highly sensitive to variations and intonation of speech, and are therefore intolerant of imperfections in synthesized speech. Even though the effectiveness of this output modality is not easy to determine, we have to consider the unrestricted access that blind users have to this medium of communication.

In a human dialogue, communication not only consists of acoustical information, but also of visual information like facial expressions and gestures. To recreate this nonverbal communication, *avatars* are an effective solution that, in combination with speech output, conduct conversation with the user in a highly natural way.

Graphical user interfaces (GUIs) are state-of-the-art for the operation of electronic devices. However, while these GUIs simplified the interaction with devices for many people, they lead to new difficulties for the visually impaired, for instance. To solve this problem, new *tactile graphic output* devices can be used - pin plates give a tactile impression of the automatically condensed graphical information.

2.3. Integration

In respect to the integration process, parameters about temporal availability and the fusion possibility of the different modalities must be inferred. These values can have meaning or not, depending on the level of abstraction in which the data is being processed (e.g. the representation of speech input as a signal, as a sequence of phonemes or as a meaningful parsed sentence, are examples of different abstraction levels) [Nigay 1993].

Some authors proclaim that a good way to realize multimodal interaction systems is by following the capabilities already present in the Human Information Processing System [Schomaker 1995]. These capabilities are highly optimized in respect to the interactive process.

Basically, there are two distinct classes of multimodal systems - one integrates signals at the *feature level* and the other at the *semantic level*. The first one is based on multiple hidden Markov models or temporal neural networks and is adequate for closely coupled and synchronized modalities (e.g. speech and lip movements). The other one is based on amodal input and is appropriated when the modes differ substantially in the time scale characteristics of their features (e.g. speech and gesture input) [Wu 1999].

A typical architecture of a multimodal system, inspired from [Oviatt et al. 2000], is presented below:

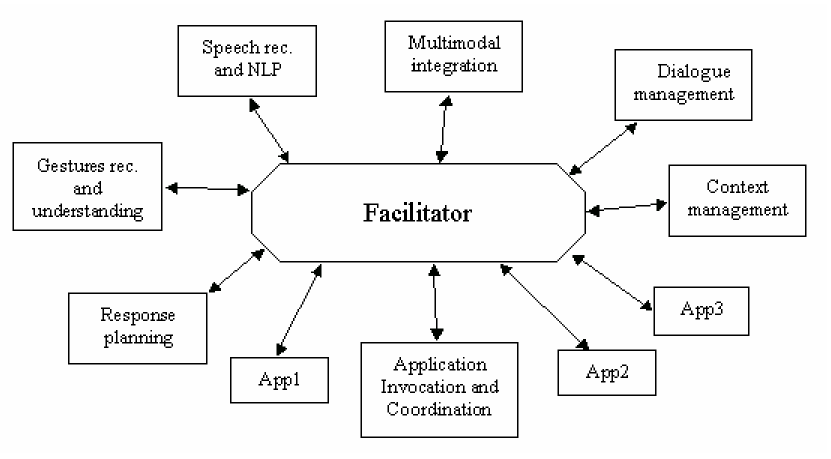

Figure 2 – Facilitated multimodal architecture

We are considering the integration process in a semantic level, which means a *late fusion* just after the recognition and analysis process of each modality. Other results of our work in this area can be found in context of the EMBASSI project [Hildebrand 2000].In the next section, we describe the software infrastructure concepts we have adopted to realize the concepts that we have begun to describe.

3. Software agents approach

From what we have described until now, it is easy to foresee that the realization of systems capable of interacting with the user in a multimodal (and intelligent) manner results in complex information processing, due to the number and diversity of the involved components. One way to realize this information processing flow as an architecture is to pipeline the components via procedure calls, or remote procedure calls in the case of a distributed but homogeneous system (in programming language). For distributed and heterogeneous software, this may prove difficult, and the solution lies in *agent-based software engineering*.

In essence, the several system components are "wrapped" by a layer of software that enables them to communicate via a standard language over TCP/IP. The communication is then processed directly based on some concepts of distributed systems, like asynchronous delivery, triggered responses and multi-casting, or, alternatively, by using a *facilitated* form. Actually, we are using a facilitated ("hub-spoken") multi-agent architecture. Due to the bottleneck derived from high-volume data transfer from the modality 'Recognizers' to the modality 'Analyzers', we are "by-passing" this approach, putting the Speech Recognition and Natural Language Processing as one single agent, as well as the Gesture Recognition and Gesture Understanding, as depicted by Fig. 2.

The term "agent" is used increasingly to describe a broad range of computational entities and sometimes tends to obscure the differences between different approaches. The two headings under which we can subsume most uses of agents are: the simplification of distributed computing (agents as intelligent *resource managers*) and, also, the overcoming of user interfaces problems (agents as *personal assistants* which adapt to the user). Therefore, one central component of the system architecture will be Context Management, which stores, manages and serves information crucial to implementing a user-adaptive interface. User and resource profiles should be used in a kind of situation and location awareness, influencing the interaction/operation process.

3.1. Development

Agents that will interact with one another require some method of communication in order to coordinate their activities and distribute and collect information. Therefore, we need some Agent Communication Language (ACL) and software tools to enable any operation with them.

To integrate all the independent and heterogeneous system components, we are using the *de-facto* standard KQML (Knowledge Query and Management Language) as the ACL. This decision was based on the effectiveness of KQML and the variety of existing tools. KQML is complementary to distributed computing approaches (e.g. OMG CORBA/IIOP), whose focus is on the transport level (how agents send and receive messages). The focus of KQML is the "language level" – the meaning of the individual messages [Finin 1993].

To this end, we have found several software tools to build agent-based systems. Some examples are OAA [Martin 1999], Jackal [Cost 1999] and JATLite [Jeon 1999]. The last one was made open-source software available under the GNU general public license since December 1998.

The tool we are using is based on the JATLite software, which consists of a tool for the use of KQML by agents written in the Java programming language. Java is a useful language for writing agents because it is relatively platform-independent and has good language support for multi-threading. JATLite is a successor of JAT (Java Agent Template) [Stanford 1998], intended to be a much lighter-weight package. It consists of Java classes and programs for creating new systems of agents that communicate over a network to perform distributed computation. The agents send and receive KQML messages, although other languages, such as the Foundation for Intelligent Physical Agents (FIPA) ACL standards, can also be used. There are some arguments that JATLite has some characteristics still missing in other tools, such as *reliable message delivery* and *migrating agent communication* [Jeon 2000].

One good characteristic, which has been demonstrated to be a good practice in the context of agent communication, is the separation from the language used to code the contents. KQML does not limit agents to the language of data representation; it only provides an extensive and standardized way to deliver data among agents. Conceptually, a KQML message consists of a *performative*, its associated arguments, which include the real content of the message, and a set of optional arguments, which describe the content in a manner which is independent of the syntax of the content language.

Therefore, we are adopting XML (eXtensible Markup Language) as the content language, providing two main advantages: the data are human readable, and there are a number of standard solutions for XML processing. For example, a message from a gesture recognizer, informing the analyzer about the 3D position and orientation of a pointing gesture, may look like this:

```
(tell :sender GestureRec
      :receiver DeviceSelection
      :reply_with Ge-Rec_Msg1.0
      :ontology spatialOntology
      :language XML
      :content (<event time=”23:59:59:321“>
                  <vector x=”1“ y=”2“ z=”2“
                          dx=”0.4“ dy=”0.75“
                          dz=”0.3“
                 actor=”hand” />
            </event> ) )
```

Figure 3 – Agents communication example

The previous communication example consists of a `tell` *performative* exchanged between two agents (a *sender* and a *receiver*) whose content is defined in XML language. In practice, to have a Java program send for and receive those kinds of messages from any other agent, it must be "agentified". In essence, this simply means, to import some classes, inherit another one, and provide name, address and port values. The following figure presents a usage example of the agents' software construction tool:

```
import KQMLLayer.KQMLmessage;
import KQMLLayer.ParseException;
import RouterLayer.AgentClient.KQMLmail;
...

public class MyClient extends Client {

    public MyClient(String Agentname, String Host, int port){
        super(Agentname, Host, port);
    }

    public boolean Act(String message){ //the method to be used

        KQMLmail mail = null;
        ...
        KQMLmessage kqml = mail.getKQMLmessage();
        String perform = kqml.getValue("performative");
        String receiver = kqml.getValue("sender");
        String receiver = kqml.getValue("receiver");
        String replyid = kqml.getValue("reply-with");
        String replyid = kqml.getValue("in-reply-to");
        String content = kqml.getValue("content");

        // do something
        sendMessage(result);
        return true;
    }

    public static void main ( String[] args ) {
        String Agentname = "agentenname";
        String Host = "host";
        int port = "port";
        new MyClient(Agentname, Host, port);
    }
}
```

Figure 4 – A JATLite-based template to "agentify" a client

This codification example can also demonstrate that the complexity of the enabling technologies for software agents is being increasingly hidden, making the fast setup of systems possible.

Today, there appear to be two groups using the concept of agent as an important tool for designing and implementing software. The first is the Artificial Intelligence community. The second is the Internet community.

4. Verbal communication

As stated in the beginning, out of the several interaction modalities by which is possible to communicate with machines, in this paper, we are emphasizing the speech modality.

Speech appears to be a very important form of interaction. Some examples of its application are: support provided to non-skilled users, communication with children in an edutainment context and the development of courseware for languages. Some related projects that have been realized are (1) "Living in Portugal" [Sá 1998] - a multimedia CD-ROM for learning/teaching the Portuguese language, in which the user can be involved in real dialogues with the possibility to record his/her own voice for later comparison with correct sentences; (2) the "Portuguese/Chinese/Portuguese Multimedia Encyclopedia" [UM 1998] - among other functionalities, sets of words can be recorded to offer an automatic suggestion about the user's pronunciation, and (3) the "Foguetão 2002" [Araújo 2000] - a CD-ROM to initiate children into reading and writing where, due to the age of the target users, all communication has to be performed using speech, at least from the computer to the users.

The basic features of speech technology are simple to figure out, but there are subtle and powerful capabilities provided by computerized speech, whose limitations – as well as strengths – are important to understand for the effective use of speech input and output in a user interface. On the other hand, at a higher level of abstraction, speech dialogues have to be established somehow, with the consequent need of representation and processing. An effective speech application simulates some of the core aspects of human-human conversation. How these dialogues can be computationally realized consists in a relevant research topic.

4.1. Speech processing basis

We can subdivide speech technology into main areas: one concerning the output - *speech synthesis* (SS) - and the other related to the input - *speech recognition* (SR). SS is the process of converting text to speech, while SR is the other way around, a speech-to-text conversion. After the SR, another relevant step is language understanding, also called *speech analysis* (SA) or *Natural Language Processing* (NLP).

Currently, we can find several speech engines on the market, as well as different API's that we can use in our applications. The next figure depicts the different implied layers for a speech-enabled system:

```
   Application
-----------------
       API
-----------------
  Speech engine
-----------------
      O. S.
-----------------
    Hardware
```

Figure 5 – Speech-enabled system

Speech Recognition

The goal of SR is to transform the user voice (an audio data stream) into a text string. This text string respects specific grammar rules and includes only syntactically correct words.

In order to implement a recognizer, some major steps must be performed: (1) grammar design – what words may be spoken and corresponding patterns; (2) signal processing – spectrum (frequency) characteristics analysis; (3) phoneme recognition – comparison of the spectrum patterns to the patterns being recognized (language phonemes); (4) word recognition – comparing the sequence of likely phonemes with the words and patterns specified by the active grammar; (5) result generation – recognized words, normally an utterance (the recognizer's best guess or, sometimes, the alternative guesses).

The primary way in which an application controls the activity of a recognizer is through control of its grammar. There are two main grammar types: rule grammar – application provides the recognizer with rules to determine what the speech engine recognizes (typically used for command-and-control applications); and dictation grammar – which uses the context of the spoken words to constrain a dictation vocabulary provided with the speech engine. The last one, having fewer restrictions, permits a kind of free-form speech input, but implies substantially more computing resources, a higher quality audio input, and more errors. It is developed by statistical training and, normally, the SR already has built-in dictation grammar.

Speech Analysis

The SR process maps speech sounds to words. This is sufficient for automatic dictation applications, however, speech-enabled applications normally have to extract some kind of meaning from the words – map the input messages into meaningful actions. The possible input messages are the *language*, and the mapping is the *understanding*. Some speech understanding techniques are (1) wordspotting – detecting the presence of certain keywords, (2) syntactic/semantic parsing – traditional approach, parsing text using a grammar, and (3) learning the mapping from language meaning – instead of writing grammar manually.

Speech Synthesis

SS is the conversion of written text into spoken language; also referred to as *text-to-speech* (TTS) conversion. The relevant steps are: (1) structure analysis – determining the start and end of the structures (paragraphs, sentences, etc.), and also the punctuation and formatting data; (2) text pre-processing – analyzing the text for special language constructs (abbreviations, acronyms, dates, times, etc.); (3) text-to-phoneme (i.e. conversion) – conversion of each word to phonemes (different languages have different sets of sounds); (4) prosody analysis – processing of the sentence structure (words and phonemes to determine the appropriate prosody), and also other features of speech other than word sounds (pitch, timing, pausing, rate, emphasis), and (5) waveform production – using the phonemes and prosody information to produce the audio waveform for each sentence (by concatenation or by formant synthesis).

4.2. Dialogue representation

In this section, we introduce the Voice Extensible Markup Language (VoiceXML), a language to create interactive voice response applications [Forum 2000]. It consists of a XML Data Type Definition (DTD) with strong capabilities to be extended in the future for a more general purpose, such as multimodal dialogues [Raggett 2001].

The VoiceXML DTD has several elements with sets of attributes, through which the elements' behavior can be controlled. Some of those elements are used more often due to the most common behavior regarding dialogues. These elements can construct a dialogue like the Wireless Markup Language (WML) does, to create a WAP application. The structure of a dialogue has almost the same structure the WAP applications have. In WAP, we have the "deck of cards" architecture where one card can execute some code and then pass the control to another card. In VoiceXML, we have documents that create dialogues, and that can pass the control of the dialogue to another one.

Going a bit deeper into technical details in order to provide some feeling about the language: the main elements of a document (within the `<vxml>` tag) are forms. A `<form>` tag is used to get user input and perform other associated functionality. Each form has the `id` attribute to define its name. This `id` attribute is important as it enables the program control to go back to the same form. Inside this tag and in order to play a prompt to the user within it, the `<block>` tag is used. The `<block>` tag allows the specification of the executable code. To specify a simple prompt, the `<prompt>` tag is used.

```
<form id="MusicStore">
   <block> Welcome to the MusicStore
      <prompt>
         Say singer or genre to start
      </prompt>
   </block>
</form>
```

Figure 6 – VoiceXML elements

Execution within a VoiceXML document flows in document order. Execution flows from the current dialogue to a different dialogue or document, based on either an explicit transition statement

```
<form>
       <goto next="#actors_list/>
</form>
```

or by speech recognition

```
<link next="operator_xfer.vxml">
       <grammar> operator </grammar>
</link>
```

A `<link>` specifies a grammar that is independent of any particular dialogue. If the speech recognition engine matches a grammar with document or application scope, which is defined in a different dialogue, the interpreter transitions to that dialogue.

In case the input is sufficiently abstract, contemplating actions like pointing, for instance, some choices can be made beyond using speech. With adequate synchronization mechanisms that can be inherited from some existing languages, such as the Synchronized Multimedia Integration Language [SMIL 2001], the goal of having multimodal dialogue systems in wide use is not far away.

4.3. Development

For reasons inherent to the adoption of the Java Technology, we are using the *Java Speech API* (JSAPI), which defines a software interface to state-of-the-art speech technology. The two core speech technologies (recognition and synthesis) are supported by JSAPI. There are several engines on the market with very reasonable accuracy for both technologies (e.g. IBM Via Voice and Microsoft Speech).

“Speech engine” is the generic term for a system designed to deal with either speech input or speech output. Speech synthesizers and speech recognizers are both speech engine instances. These two engines are created within an application and they can be adjusted to a specific goal.

The basic processes for using a speech engine in an application are (1) its location, creation and beginning, (2) the allocation of resources, (3) the beginning operation, (4) the usage of the engine and, finally, (5) the deallocation of resources.

As usual, to make it clear in practice, we present a small example in the next figure:

```
import javax.speech.*;
import javax.speech.recognition.*;
import java.util.Locale;
...
class RecognizerExample
{
   public static Recognizer rec = null;
   public recognizerExample() {
      ...
      rec = Central.createRecognizer(null);                     //(1)
      rec.addEngineListener(new TestEngineListener () );
      rec.addResultListener(new TestResultListener(o,si));

      rec.allocate();                                           //(2)
      rec.waitEngineState(Recognizer.ALLOCATED);

      rec.requestFocus();                                       //(3)
      rec.resume();

      RecognizerAudioAdapter raud = new TestAudioListener();      //(sys.)
      rec.getAudioManager().addAudioListener(raud);
      RuleGrammar gram = rec.newRuleGrammar("music");
      Rule search = gram.ruleForJSGF("singer{sin}|genre{gen}");
      gram.setRule("search", search, true);

      rec.commitChanges();                           //commit the grammar
      ...
   }
   ...
}
```

Figure 7 – Java usage of a speech recognition engine

In respect to XML parsing, we have adopted the Apache packages [ASF 2001]:

```
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.w3c.dom.NamedNodeMap;
import org.xml.sax.SAXException;
import org.w3c.dom.DOMException;
import org.xml.sax.InputSource;
...
public class I3mod extends javax.swing.JFrame {
        ...
        try {
            Stack xmlStack = new Stack();
            DOMParser parser = new DOMParser();
            parser.setIncludeIgnorableWhitespace(false);

            InputSource in = ...
            parser.parse(in);
            Document doc = parser.getDocument();
            xmlStack.push(doc);

            Node currnode;
            while(xmlStack.empty()==false){
            ...
        }
        catch (SAXException e) { System.err.println (e); }
        catch (IOException e)  { System.err.println (e); }
        catch (DOMException e) { System.err.println (e); }
        ...
```

Figure 8 –Java classes for XML parsing

Documents represented in XML have the benefit of the existence of several tools to process them. The tendency is that this kind of tool will be present "everywhere", since we can foresee that most of the interchangeable data will be represented in this language.

5. Conclusions

Computer graphics highlight human-computer interaction, from animation to the immersive environments. There are now a broad range of technologies available that, when well-combined, can make the interaction with machines really effective; an important added value is the possibility of their being used by different persons, in many situations.

To realize multimodal interaction, we need standards and tools, which we can decide to use when designing applications. Speech technology is actually fully available, and we can foresee the appearance of vision-based products for common environments. The goal should not be the improvement of individual components, but rather the synergistic use of them.

The global network has been the generalized access to the information, with the strong need of standardization. Speech dialogue applications are promising domain in terms of accessibility, which has to be expanded to meet the requirements in terms of combining not only speech, but also other modalities, like gestures (pen-based or in the 3D space).

In our laboratory experiments, we have tested the presented technology, using speech jointly with video-based pointing [Sá 2001] (for device selection), with good results. Since the interaction was based on a dialogue manager "driven" by voice, at this stage, we have simply mapped the other modalities to fit the dialogue representation requirements. We are working to change this approach to a generic multimodal dialogue representation.

6. Acknowledgements

This work is partially supported by "Fundação para a Ciência e a Tecnologia", through a scholarship in the context of the Information Society Operational Program (reference PRAXIS XXI/BD/20095/99).

The EMBASSI project is supported by the German Ministry of Science and Education as a focus project on Man-Machine Interaction (signature 01IL904).

The author would like to thank Mykhailo Andriluka and Eva Oliveira for the interesting discussions on the topic.

7. References

Araújo, M.H., Gomes, F.M., Sá, V.J., *Foguetão 2002 - Edutainment CD-ROM*, ISBN 972-0-61102-2, Porto Editora Multimedia, 2000.

ASF, *Apache XML Project*, Online documentation: http://xml.apache.org/xerces2-j/index.html, 2001.

DSI, "Multimedia Encyclopedia for the Teaching of Portuguese/Chinese/Portuguese", JNICT – PLUS/P/PDP/1180/95, Final Report, University of Minho, 1998.

Encarnação, J.L., "Computer Graphics in Europe", *IEEE Computer Graphics and Applications*, Jan/Feb 2000.

Finin, T., et al., *Specification of the KQML Agent-Communication Language*, University of Maryland, 1993.

Forum, *VoiceXML Forum*, Online documentation: http://www.voicexml.org, 2000.

Frost, H. R., *Java Agent Template*, Online documentation: http://cdr.stanford.edu/ABE/JavaAgent.html, 1999.

Hildebrand, A., Sá, V.J., "EMBASSI: Electronic Multimedia and Service Assistance", *IMC'2000 - Intelligent Interactive Assistance and Mobile Multimedia Computing*, Rostock, Germany, November 2000.

Jacob, R.J., "The Future of Input Devices", *ACM Computing Surveys* 28A(4), 1996.

Jeon, H., *JATLite*, Online documentation: http://java.stanford.edu/, 1999.

Jeon, H., Petrie, C., Cutkosky, M.R., "JATLite: A Java Agent Infrastructure with Message Routing", *IEEE Internet Computing*, March/April 2000.

Nigay L., Coutaz J., “A Design Space for Multimodal Systems: Concurrent Processing and Data Fusion”, *Proceedings of InterCHI'93, Conference on Human Factors in Computing Systems*, ACM, 1993.

Raggett, D., *Next steps for W3C work on Multimodal Standards*, Online documentation: http://www.w3.org/Voice/multimodal-rfp.html, 2001.

Sá, V.J.; “Development of an Application for the Teaching/Learning of the Portuguese Language: a Case Study”, Master Thesis, University of Minho, 1998.

Sá, V.J., Malerczyk, C., Schnaider, M., “Vision Based Interaction within a Multimodal Framework”, *Proceedings of the 10th Conference of the Eurographics Portuguese Chapter*, Lisbon, October 2001.

Schomaker, L., et al., “A Taxonomy of Multimodal Interaction in the Human Information Processing System”, A Report of the *ESPRIT PROJECT* 8579 MIAMI, WP1, 1995.

SMIL, *Synchronized Multimedia Integration Language (SMIL 2.0) Specification, 2001.*

Sun Microsystems, *Java Speech API Programmer’s Guide v.1.0*, USA, 1998.

van Dam, A., “Beyond WIMP”, *IEEE Computer Graphics and Applications*, Jan/Feb 2000.

Wu, L., Oviatt, S., Cohen, P., “Multimodal Integration – A Statistical View”, *IEEE Transactions on Multimedia*, 1(4):334-341, 1999.