System Support for Multi-Modal Information Access and Device Control

Anthony D. Joseph, Barbara Hohlt, Randy Katz, and Emre Kiciman
EECS Department, CS Division
University of California, Berkeley
http://iceberg.cs.berkeley.edu/

Abstract

Technological progress is yielding a convergence of communication technologies, mobile devices, and "smart spaces" (environments that contain computer-controllable sensors, actuators, and I/O devices). Empowered by these advances, users are demanding access to and control of existing and new environments, information repositories, and applications.

There are many interesting complications associated with taking advantage of this progress: each of the communication networks has very different characteristics from the others (e.g., bandwidth, latency, cost, quality of service, etc.), the new devices have user interfaces that are quite different from their predecessors' graphical interfaces (e.g., they may have constrained screen sizes, no screens at all, or include new interfaces, such as audio or speech), and one significant complication is that every smart space is unique — no two have the same devices or interfaces.

It is the challenge and task of a user interface developer to provide users with many devices and many modes of interactions with a single, uniform interaction environment. Users should be able to choose the device that they want to use to interact with their information or environment — multi-modal user interfaces enable this choice.

There are two requirements in addressing the challenge of building multi-modal interfaces: application-level tools for describing and constructing interface components and system-level tools for automatically composing the components. Given these tools, user interface developers will be able to implement transcoding or transformational operators along with multi-modal interfaces, and then rely upon the environment to dynamically compose operators and interfaces to serve new and existing devices.

One of the goals of the University of California, Berkeley's Iceberg project is to provide user interface developers with the system-level tools that they need, by exploring how to merge the different design philosophies and requirements that are associated with each of the communication networks, devices, and smart spaces. The project is constructing a large-scale testbed that will incorporate current and prototype technology. Once it is deployed, the testbed will provide a proving ground where one can explore ideas for ubiquitous access to information, anywhere, anyplace, anytime, and using any I/O device.

1 Introduction

A convergence of communication technologies, mobile devices, and environments is occurring. Technological advances are yielding many new, diverse communication networks and mobile devices for using these networks. The advances are also resulting in the creation of "smart spaces," environments that contain computer-controllable sensors, actuators, and I/O devices (*e.g.,* cameras, microphones, thermostats, etc.). Simultaneously, users are demanding access to and control of existing and new environments, information repositories, and applications.

The new networks include two-way pager networks, wireless packet radio, digital cellular, etc. The networks have very different characteristics, in terms of bandwidth, latency, cost, quality of service, and methods for carrying information.

The new devices include pagers, Personal Digital Assistants (PDAs), palmtop computers, cellphones, and laptop computers. The capabilities of these devices varies significantly along several axes: display technology (size, resolution, color depth, etc.); computational power; memory; battery power, and communications (2-way RF or IR, 1-way or 2-way paging, continuous/intermittent connectivity, asymmetric bandwidth, etc.).

The technological advances represented by these new networks and devices are a distinct change from traditional stationary and mobile computing environments. Many of the new devices rely on user interfaces that are quite different from their predecessors' Windows, Icons, Menu, and Pointing (WIMPy) graphical user interfaces. The new devices have constrained screen sizes or no screens at all. Some also include new interfaces, such as audio or speech.

Smart spaces are environments that are under computer control. These environments are a natural extension of today's office environments, which contain a variety of computer-controlled devices (*e.g.,* Heating / Ventilation And Cooling systems, door locks, elevators, slide projectors, TV monitors, A/V devices, etc.). Cost reductions are making it possible to extend computer control to the home environment, where users are building spaces that contain cameras, microphones, movable blinds, DVD players, and other A/V devices. Users want to be able to access and control these devices using their mobile devices and their new communications networks. One big complication is that every smart space is unique — no two have the same devices or interfaces.

1.1 Multi-Modal User Interfaces

It is the challenge and task of a User Interface (UI) developer to provide users with many devices and many modes of interactions with a single uniform interaction environment. Users should be able to choose the device that they want to use to interact with their information or environment — multi-modal user interfaces enable this choice [3, 4, 12].

One of the goals of University of California, Berkeley's (UCB) Iceberg project is to provide UI developers with the system-level tools that they need to build and dynamically compose multi-modal user interfaces. The idea of multi-modal user interfaces has been around for some time [1, 14]. These interfaces support user interaction based upon multiple, distinct modalities (e.g., typing, writing, speech, and gesture) [17].

There are two requirements in addressing the challenge of building multi-modal interfaces: application-level tools for describing and constructing interface components and system-level tools for automatically composing the components. The idea is that UI developers should be able to construct transcoding or transformational operators and design interfaces and then rely upon the environment to dynamically compose operators and interfaces to serve new and existing devices.

This paper discusses the tools that enable automatic interface construction, however, we do not directly address the issue of constructing the interfaces themselves. Other portions of the Iceberg project and other projects at UCB are exploring the difficulties associated with automatically creating easily usable interfaces [6, 8, 11].

1.2 The Iceberg and Ninja Projects

The Iceberg project is exploring architectures for combining voice and data services in many, diverse, interconnected networks: third-generation and beyond digital cellular (GSM, CDMA, and UMTS / IMT2000), IP, PSTN, wireless IP, and 2-way pager networks [7, 10]. This set of networks covers a wide range of transport technologies, input/output interfaces, and user interfaces. To test our ideas for the Iceberg architecture, we are constructing a large-scale testbed that will include all of these networks.

The primary goal of the Iceberg project is to design and build a unified communications operating system and architecture for the twenty-first century. This paper addresses one of the aspects of convergence in the communications architecture — user interfaces that span many different modes of information access and device control.

Supporting the Iceberg project is the Ninja project. Ninja is developing the infrastructure for building reliable, incrementally scalable, highly-available, persistent distributed services on commodity networks of workstations. The prototype Ninja environment for services is iSpace. The iSpace environment allows a service developer to concentrate on service functionality without having to worry about the availability, persistence, reliability, or scalability aspects of the service.

Both the Iceberg and Ninja projects use Java as the development environment. Our intention is that operators and services developed in Java using the Iceberg and Ninja environments will be able to be used

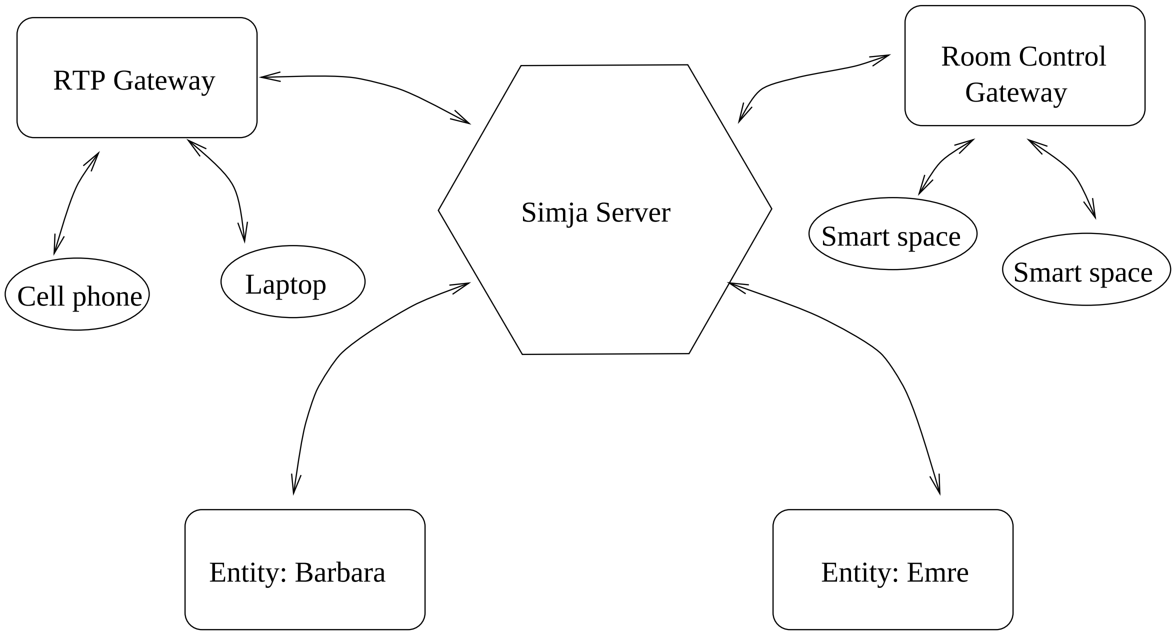

Figure 1: The Simja environment. This figure shows a Simja server, two local users of the environment (entities), an Mbone RTP gateway to remote users and devices, and a room control gateway. The Simja server provide routing and transformational / transcoding services between the various components.

anywhere (Our motto is "Write operators once, run them anywhere!").

2 Simja: An Environment for Multi-Modal Interfaces to Services

We have designed and implemented a research environment for building multi-modal interfaces to smart spaces, Simja[1].

Figure 1 provides an overview of the Simja environment. There are several components: local users of the environment, gateways to remote users and devices, and a Simja server. Users and devices are entities.

The Simja server automatically and dynamically transcodes information from the input format to an entity's desired output format. Simja allows users to use graphical, text, or speech-based user interfaces to interact with one another and to control the devices in smart spaces. For example, Barbara could speak a message into her computer and Emre could choose to receive the message as audio or as a text popup message (Simja would transcode the speech to text). One important consideration is that the source entity does not have to know the target entity's output format.

Simja performs the automatic combination of operators for transcoding information from one format to another by dynamically forming paths to convert from the input format (speech, graphical, or text) to the appropriate output format (device control commands for a smart space, speech, graphical, or text).

2.1 Architectural issues

One of the key challenges in designing Simja was to provide an environment that enables developers to rapidly add new devices and communications networks to a service infrastructure. During the design of Simja, we exposed several important architectural issues: strongly-typed transcoding components and information, automatic path creation (APC), channels for conveying control or metadata information.

[1] Simja stands for Simulated Ninja. We created Simja because the Ninja environment is still under development. Eventually, the Simja tools will be migrated to the Ninja environment.

2.1.1 Strongly-Typed Interfaces

The interfaces to transcoding operators, input and output devices, and information streams in Simja are strongly typed. Interfaces are defined using eXtensible Markup Language (XML) documents, consisting of key, value pairs that specify the attributes of the items. The idea is that developers will provide XML descriptions of input and output devices and the infrastructure will use automatic path creation to connect the two.

In addition, a service author should be able to easily create a service from existing Java code. Other than associating the XML description with the operator, no code change should be required. Furthermore, the system will provide automatic dynamic composition of the services to create new services on-demand.

Here is an example of an XML definition for a 44Khz 16-bit WAV stream.

```
<stream>
  <!-- other stream related elements >
  <meta>
    <tag key="PRIMARY"><value>audio</value></tag>
    <tag key="audio.format"><value>wav</value></tag>
    <tag key="audio.samplerate"><value>44000</value></tag>
    <tag key="audio.samplesize"><value>16bit</value></tag>
  </meta>
</stream>
```

We have not yet finalized the XML Document Type Description (DTD) for describing operators. In particular, we are exploring how to expand the DTD to allow arbitrary Java classes to act as operators. Currently, an operator in Simja must define several specific functions and receive it's inputs and outputs *not* as function arguments, but rather through calls to `getInputData()` and `setInputData()`. This approach works fine, but it would be easier for developers writing operators if they could simply define an operator's functions and then write a mapping between the Java function's arguments and return values and an operator's inputs and outputs. Again, the goal is to allow authors to write services without having to make extensive code changes.

In addition, we are considering and exploring additional metadata that could be useful as components of the descriptions. For example, it would be useful if the description incorporated dynamic information from the environment, such as the current load on a computer. However, while it is appropriate to store static information about an object in its XML description, where should dynamic information about an object be stored? Should the information be completely decoupled from the object's XML description or should the XML description contain instructions or pointers for retrieving the relevant information dynamically? These are open questions that we are exploring.

2.1.2 Automatic Path Creation

Given a set of typed input and output devices, operators, and data streams, Simja automatically creates a path from the input device to the output device. Automatic Path Creation involves two steps: choosing the operators for the path and instantiating the path. Path instantiation involves taking a path that's been created and instantiating the operators and connecting them together.

Currently, APC simply attempts to find short paths between inputs and outputs. However, there are other "operator cost" functions, in addition to path length, that could be considered (*e.g.,* computational cost, latency, etc.). We are exploring ways to include developer-specified cost functions, both in the specifications of operators and in the APC process.

There are several research areas that are related to Simja's automatic path composition, including software reuse and component composition technology, and AI planning and search algorithms. The InfoSpheres project at CalTech is researching the composition of distributed active mobile objects that communicate using messages [2].

2.2 Future thoughts on APC

One area that we're exploring is the type system used by automatic path creation. Often, the keys of the key, value pairs are related to each other. In the above audio description example, the "samplerate" key only makes sense when the primary type of the data is audio. For example, such relationships between keys and types could be explicitly stated within the type model, perhaps as a hierarchical structure of keys, where "samplerate" would really be "audio.samplerate"[2].

An alternate choice would be to use recursive definitions, where the XML tags themselves and third party DTDs would be used to describe data. This approach would provide the advantage of being able to leverage other people's work in categorizing the world. The disadvantage is that this approach starts to mix too much content data with meta-data raising the question of where to draw the line of data versus meta-data, especially when leveraging XML DTDs. Using third party DTDs, a stream of audio might look like the following:

```
<stream>
  <!-- other stream related elements >
  <meta>
    audio
    <format>
      wav
      <samplerate>44000</samplerate>
      <size>16bit</size>
      <REFERENCE TO STREAM SOURCE />
    </format>
  </meta>
</stream>
```

The idea is that XML is designed to markup data with context information, and that's exactly what we want — data streams marked up with context information. The actual DTD that is used would depend upon the type of data. An interface definition could use multiple DTDs (*e.g.,* describing a speech sample would require a language DTD and an audio DTD).

The main difficulty with this approach would be how to handle type checking. Options that we're considering include only allowing a subset of XML DTDs to work (*e.g.,* no element attributes) to simplify the task of type-checking (currently, we're only dealing with elements). Other problems are the fact that XML is designed to markup data, and not to be embedded within the data. Since XML can't be embedded within our data stream, it is not entirely clear how a third party DTD would be used.

Other interesting APC research areas that we're exploring are semantic information (what is the domain of problems that can be solved using APC), dynamic paths (changes to paths that occur during the lifetime of a service), service discovery (how to locate operators and services), the operator programming model, and heuristics / caching techniques can be used to create paths more efficiently.

2.2.1 Control / Metadata Channels

The choice of an operator may not be solely a function of the XML interface, but rather also a function of the input / output devices and users. For example, we have found that generic (large vocabulary), speaker-independent speech recognition can be slow and inaccurate. However, if the N-gram used by the recognizer is constrained, the result is rapid, accurate speech recognition. The constraint information may come from speaker identification (*e.g.,* caller line identification from a phone) or from knowing the target application or device (*e.g.,* controlling the devices in a smart space).

Simja provides a control or metadata information channel that is used to convey this information between operators, streams, and input / output devices during APC. The control channel can also be used during

[2]The XML examples shown above already have keys like "audio.samplerate." However, that's just a name, and doesn't represent a hierarchy to Simja.

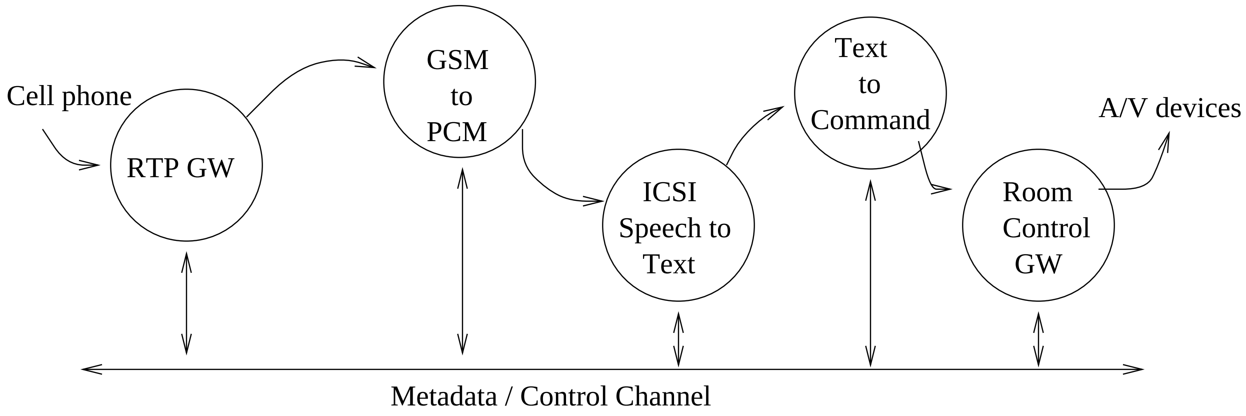

Figure 2: One direction of a sample Simja path.

operator execution as a mechanism for maintaining state information. The channel allows a service to specify requirements such as execute this operator twenty times. The service can then monitor this fact using a variable like "operatorhasrun = 20" and the operator can add one to this variable number each time it is executed. Note that the operator would then be inserted into the path 20 times.

2.2.2 An Example Path

One direction of a example path between a cellphone and a smart space is presented in Figure 2. Simja uses APC to dynamically create the path and instantiate the appropriate operators based upon the input device (a GSM cellphone) and the output device (A/V devices in a smart space).

Once path creation and instantiation is complete, the control path is used to select the N-gram used by the speech recognizer and to select the grammar used by the text to speech.

3 Simja Implementation

The current implementation of APC in Simja uses depth-first search to both create and instantiate paths, which can cause problems with cycles in the graph of operators (future versions will use breadth-first search). The version of APC being implemented for iSpace relies upon breadth-first search.

Simja uses XML descriptions to describe operators, however, the Java class that interprets the XML description and "models" the operator is the class `AbstractOperator`. This is the super class of all operators, and during path creation, creating a model of an operator and creating the operator itself is exactly the same thing. Either way, the XML description is used to initialize the number of inputs and outputs, their types, etc. When path creation is complete, the result is the complete set of usable operators and an "implemented" (instantiated) path.

This approach is why the current Simja graph search is based upon depth-first search. Using depth-first search, it is easier to keep the state of a single "current path" and move up and down the search tree than it is to create the multiple "current paths" that would be required when using breadth-first search. The iSpace version of APC uses a separate operator model, making path creation more light-weight, but also requiring that path implementation be handled separately.

We have implemented several operators using wrappers around various programs: audio format conversion tools, speech-to-text recognition based upon the International Computer Science Institute's speech-to-text recognition project [9], text-to-speech generation using the University of Edinburgh's Festival project [5], and room control programs. We are in the process of developing a generic interface for speech recognition programs, so that we can more easily import other speech recognizers into our environment [15, 16, 18].

Within the Simja environment, adding support for new devices is a simple matter. For example, we added an interface for GSM cellphones to the room control application within only a few hours.

4 Simja Applications and Testbed

We are in the process of implementing the Simja architecture and several applications on top of the Iceberg testbed. These services will be used on a daily basis by the campus community. The usage of the services will provide us with an invaluable repository of access patterns and information about the effectiveness of the architectures.

4.1 Multi-Modal Control of Smart Spaces

The first service that we have built with the Simja environment is a multi-modal interface for controlling smart spaces. The service supports text and speech (microphone, wired or wireless IP, and GSM) interfaces to the smart spaces (classrooms and conference rooms) in our computer science building, Soda Hall. Users of the service can control A/V equipment (*e.g.,* moving cameras, turning lights on and off, etc.) using various input devices.

Simja automatically constructs the path from the input source to the room control application and vice versa. Speech input is processed as follows, it is converted to Pulse Code Modulation (PCM) audio, speech-to-text conversion is performed, natural language processing (currently, simple word spotting) converts the text to commands, and the commands are passed to a room control gateway. Responses are converted to text, then PCM speech, and finally GSM audio. For text input, the process begins with Natural Language Processing (NLP). Responses are converted to text for presentation to users.

This service supports Simja's control / metadata channel, each operator selects the appropriate customization information (*e.g.,* the speech-to-text operator selects a room control N-gram and the NLP operator selects a room control command grammar).

Using Simja, it has been a simple task (only a few hours of work) to extend the system to support a new device, GSM cellphones, or service, multi-modal access to web-based services (*e.g.,* news headlines and stock market symbol reports). Some of these devices and services require different transcoding steps from the ones in the room control service. However, Simja allows developers to rapidly add support for such services by simply specifying the interface to the service. We are in the process of building several of additional services (*e.g.,* weather reports, search, driving directions, access to a user's personal information management repository)

Currently, we manually create interfaces for each service. However, we are exploring ways of automating the specification of speech interactions (*e.g.,* Motorola's VoxML effort [13]).

4.2 Campus-wide Testbed

Simja services are being built on top of a campus-wide testbed that will include wired and wireless telephony and IP networks encompassing the latest networking technologies. The testbed will include early prototypes of digital cellular and high-density wireless LAN technologies. A significant portion of the testbed will be operational by early 1999.

In addition to being used in a research context, Simja will include several services that will be used for educational purposes. The testbed's technology will be deployed throughout the UC Berkeley campus and in several classrooms to provide collaborative learning services for student use. Among the services will be multi-modal access to: instant messaging and web information, shared class repositories, and personal information management applications.

5 Summary

This paper discusses several of the challenges associated with providing system-level support for multi-modal interfaces and presents the Simja environment and testbed, a research prototype for exploring solutions to the challenges.

While developing the Simja environment, we have begun to address several important architectural issues: the use of strongly-typed transcoding components and information, techniques for automatic path creation (APC), the use of control and metadata channels for conveying information between transcoding elements.

The results of our work have been very encouraging, in particular implementing support for new devices is a task that takes only a few hours.

We are in the process of deploying Simja services on a campus-wide basis, an environment that will provide us with a significant amount of data about the usability of our ideas. In addition, we are continuing to explore multi-modal interfaces and new devices.

References

[1] R. A. Bolt. Put-that-There: voice and gesture at the graphics interface. *Computer Graphics*, 14:262–270, 1980.

[2] Caltech. *Caltech Infospheres Project.* California Institute of Technology, Pasadena, CA, 1998. http://www.infospheres.caltech.edu/.

[3] P. R. Cohen, M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clow. Quickset: Multimodal interaction for distributed applications. In *Proceedings of ACM Multimedia 97*, pages 31–40, Seattle, WA, November 9-13 1997.

[4] J. Flanagan and I. Marsic. Issues in measuring the benfits of multimodal interfaces. In *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 163–166, 1997.

[5] Centre for Speech Technology Research. *The Festival Speech Synthesis System.* University of Edinburgh, Edinburgh, Scotland, 1998. http://www.cstr.ed.ac.uk/projects/festival/festival.html.

[6] Group for User Interface Research. *SILK: Informal Tools for Early Stage User Interface Design.* Computer Science Division, University of California at Berkeley, Berkeley, CA, 1998. http://guir.cs.berkeley.edu/projects/silk/.

[7] David J. Goodman. *Wireless Personal Communication Systems.* Addison-Wesley Longman, Inc., Berkeley, CA, 1997.

[8] Todd Hodes and Randy Katz. Enabling "smart spaces:" entity description and user interface generation for a heterogeneous component-based distributed system. UCB Technical Report CSD-98-1008, Computer Science Division, University of California at Berkeley, Berkeley, CA, July 1998. DARPA/NIST Smart Spaces Workshop, Gaithersburg, MD.

[9] ICSI. *Speech research in the Realization Group at ICSI.* International Computer Science Institute, Berkeley, CA, 1998. http://www.icsi.berkeley.edu/real/speech.html.

[10] Rajan Kuruppillai, Mahi Dontamsetti, and Fil J. Cosentino. *Wireless PCS.* McGraw-Hill, San Francisco, CA, 1997.

[11] James A. Landay. *Interactive Sketching for the Early Stages of User Interface Design.* PhD thesis, Computer Science Department, Carnegie Mellon University, Technical Report CMU-CS-96-201, Pittsburgh, PA, December 1996.

[12] C. Mignot, C. Valot, and N. Carbonell. An experimental study of 'natural' multimodal human-computer interaction. In *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems – Adjunct Proceedings*, pages 67–68, 1993.

[13] Motorola. *VoxML: The Mark-up Language for Voice Applications.* Motorola, Schaumburg, IL, 1998. http://www.voxml.org/.

[14] C. Schmandt and E. A. Hulteen. The intelligent voice-interactive interface. In *Proceedings of Human Factors in Computer Systems*, pages 363–366, 1982.

[15] Sun. *Java Speech API.* Sun Microsystems, Palo Alto, CA, 1998. http://java.sun.com/products/javameda/speech/.

[16] S. Sutton, D. G. Novick, R. Cole, P. Vermeulen, J. de Villliers, J. Schalkwyk, and M. Fanty. Building 10,000 spoken dialogue systems. In *Proceedings ICSLP 96: International Conference on Spoken Language Processing*, 1996.

[17] A. Waibel, M. T. Vo, P. Duchnowski, and S. Manke. Multimodal interfaces. *Artificial Intelligence Review*, 10:299–319, 1996. http://www.is.cmu.edu/papers/multimodal/95.aij.ps.gz.

[18] H. Xuedong, A. Acero, F. Alleva, H. Mei-Yuh, J. Li, and M. Mahajan. Microsoft Windows highly intelligent speech recognizer: Whisper. In *Proceedings of 1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 93–96, 1995.