

Software Architectures for Networked Mobile Speech Applications

Originally published by In Automatic Speech Recognition on Mobile Devices and Over Communications Networks, Tan and Lindberg, Eds., Springer 2008. on March 2008

JAMES FERRANS
JONATHAN ENGELSMA

Abstract

We examine architectures for mobile speech applications. These use speech engines for synthesizing audio output and for recognizing audio input; a key architectural decision is whether to embed these speech engines on the mobile device or to locate them in the network. While both approaches have advantages, our focus here is on networked speech application architectures. Because user experience with speech is greatly improved when the speech modality is coupled with a visual modality, mobile speech applications will increasingly tend to be multimodal, so speech architectures therefore must support multimodal user interaction. Good architectures must reflect commercial reality and be economical, efficient, robust, reliable, and scalable. They must leverage existing commercial ecosystems if possible, and we contend that speech and multimodal applications must build on both the web model of application development and deployment, and the large ecosystem that has grown up around the W3C's web speech standards.

Author Biographies



JONATHAN ENGELSMA

Jonathan Engelsma is a Distinguished Member of the Technical Staff in the Application Research Center at Motorola Labs. His research interests are in mobile and pervasive applications. His current research focus is on platform innovations and enabling technologies that simplify and encourage device to device interaction. Jonathan joined Motorola in 1994 after completing his Ph.D. in Computer Science at Michigan State University.

JAMES FERRANS

Jim Ferrans is a Distinguished Member of the Technical Staff in the Applied Research and Technology Center of Motorola Labs. His research interests include platforms for mobile gaming and entertainment, and multimodal user interfaces.

Software Architectures for Networked Mobile Speech Applications

James C. Ferrans and Jonathan Engelsma

Abstract. We examine architectures for mobile speech applications. These use speech engines for synthesizing audio output and for recognizing audio input; a key architectural decision is whether to embed these speech engines on the mobile device or to locate them in the network. While both approaches have advantages, our focus here is on networked speech application architectures. Because user experience with speech is greatly improved when the speech modality is coupled with a visual modality, mobile speech applications will increasingly tend to be *multimodal*, so speech architectures therefore must support multimodal user interaction. Good architectures must reflect commercial reality and be economical, efficient, robust, reliable, and scalable. They must leverage existing commercial ecosystems if possible, and we contend that speech and multimodal applications must build on both the web model of application development and deployment, and the large ecosystem that has grown up around the W3C's web speech standards.

14.1 Introduction

In this chapter we explore architectures that support multimodal user interaction on mobile devices. Our particular emphasis is on those architectures that rely on speech engines located in the network instead of on the device. We will briefly survey the current state of speech recognition, then describe how voice-only applications have rapidly shifted to a standards-based web model of development and deployment. Because mobile devices already have very capable visual modalities, and because combining a voice and a visual modality greatly improves the user experience on mobile devices, mobile voice applications will increasingly be *multimodal*. After providing this background we present a conceptual model for categorizing multimodal architectures, describe several commercial multimodal systems, and discuss the standards needed before wide adoption of multimodal systems can occur. Our discussion is informed by a commercial-grade multimodal system developed by Motorola and partner companies.

14.1.1 Embedded and Distributed Speech Engines

Mobile devices first supported “speech recognition” via simple template matching: to enable voice dialing, the user first trained an embedded template matching algorithm by providing it with an audio input sample to associate with each phone book entry. To voice dial, the user would repeat the name or phrase associated with that contact, and the algorithm would compare the new audio sample against the stored waveforms to determine the contact to call. This approach is speaker-dependent and suffices for up to a few hundred contacts.

True speech recognition became practical commercially about a decade ago, and took two forms. *Transcription systems* on desktop PCs were speaker-dependent and required high-quality microphones and a quiet environment. The user would spend perhaps ten or fifteen minutes reading sample sentences to train the system, which would then do a fairly credible job of transcribing what the user said.

The second form of commercial speech recognition to arrive in the mid to late 1990s was the *network-based speech recognizer*. These were reached over circuit-switched voice calls and were speaker-independent. Instead of being able to transcribe all of a single user’s speech based on a relatively large dictionary, network-based speech recognizers could understand many users but had to be given strict constraints on what to expect them to say. Constraints were specified by context-free grammars much like those used to specify programming languages. So while a grammar based speech recognizer achieved speaker-independence by limiting what users can say, a transcription base speech recognizer achieved grammar independence by limiting the users who can speak with it.

As mobile devices have become more powerful, it became possible to embed grammar-based speech recognition systems on them. They remain less capable than their larger cousins running on network-based computers. They typically support vocabularies in the ten to twenty thousand word range, and take up roughly ten megabytes of storage. As a rule of thumb, network-based and desktop speech recognizers have vocabularies ten times larger than embedded recognizers, and have proportionately greater hardware requirements.

Transcription systems are now just starting to appear on mobile devices, where voice entry of SMS messages and email is a very valuable use case. So far these have had mixed results. Transcription systems are also moving into network-based server farms in configurations that support speaker-independent recognition, which is potentially a very significant development.

Speech recognition has steadily improved over the years, both in the network and on devices. We will see more speaker-independence, less restriction on what people can say, and other advances, although challenges remain (Deng 2004).

Speech synthesis has made parallel gains. Older formant-based systems synthesized speech from acoustic models, and tended to sound rather unnatural. But these are giving way to concatenative systems that string together segments of pre-recorded speech samples to sound far more human. As with speech recognition, speech synthesis systems based in the network are more advanced than those embedded on mobile devices.

14.1.2 The Voice Web

By the mid-1990s, speech technologies had matured to a point where voice applications could begin to displace existing touch-tone (DTMF) applications. Voice applications were initially deployed on proprietary interactive voice response (IVR) systems, which were connected to the public switched telephony network (PSTN) with specialized hardware that supported banks of incoming analog lines or digital T1 or E1 lines. In addition to integration with the PSTN, an IVR system contained speech engines, one or more voice applications, and also the back-end business logic, database interfaces, and legacy application interfaces needed to integrate the voice applications with the existing infrastructure. The proprietary nature of IVR systems meant that voice applications were costly to deploy, and difficult to port to other platforms.

In the mid 1990s, researchers at AT&T exploring ways to best implement web services realized that the web model for application development and delivery was as well suited for voice applications as it was for visual ones: it made no difference at all if the user was interfacing with microphone and speaker instead of a keyboard and display (Atkins, Ball, Baran, Benedikt, Cox, Ladd, Mataga, Puchol, Ramming, Rehor, and Tuckey 1997). The web model enables and encourages a clean division between each application's interface and its back-end business logic. All the application's legacy system integration, database access, and business logic could be factored out of the IVR platform and onto standard application web servers, using the rich variety of tools developed for visual web applications, and leveraging the simplicity of web application deployment.

This factoring required standards that would enable any IVR platform to render the same backend web application to callers in the same way. Standard web protocols such as HTTP and TCP/IP would be used of course, and resources such as audio files would be delivered to the IVR platform the same way as they would to a visual web browser. But how would the web application convey voice dialogs to the IVR platform? Some researchers proposed augmenting HTML with voice dialog constructs, but most concluded that the unique aspects of voice dialogs – the need to manage temporal flow, handle input errors, resolve ambiguous inputs, specify timings, and so on – required a new markup language.

Some early voice markup languages were AT&T's PML (Atkins, et al. 1997), HP's TalkML (Raggett 1999), IBM's SpeechML, and Motorola's VoxML (Ladd, Hay, McClaughrey, and Ferrans 1999). Commercial realities dictated there be only one, so in early 1999 AT&T, IBM, Lucent, and Motorola created the VoiceXML Forum, whose purpose was to develop a standard language. The Forum published VoiceXML 1.0 (Boyer, Danielsen, Ferrans, Karam, Ladd, Lucas, and Rehor 2000) and then gave it to the World-Wide Web Consortium (W3C) which published the VoiceXML 2.0 Recommendation (McGlashan, Burnett, Carter, Danielsen, Ferrans, Hunt, Lucas, Porter, Rehor, and Tryphonas 2004).

The industry eagerly adopted the VoiceXML standards, because they were a first major step in the disaggregation of proprietary IVR platforms into interchangeable components based on open standards. The IVR platform was transformed into a generic VoiceXML *voice server*, and it now rendered standard voice web applica-

tions hosted on standard application web servers (see Fig. 14.1). Web development and deployment technologies, coupled with these new standards, dramatically drove down the cost of voice applications, so that today literally billions of calls are processed each year by VoiceXML-based voice servers, and applications as large as the North American Directory Assistance service are based on VoiceXML.

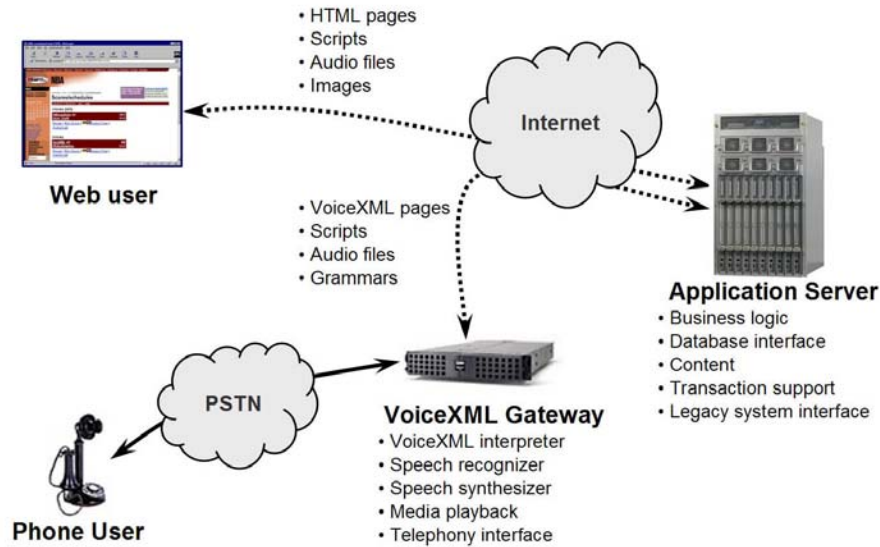


Fig. 14.1. Voice web architecture.

These new voice server platforms have become further commoditized by standards closely related to VoiceXML. The first is the Internet Engineering Task Force (IETF) Media Resource Control Protocol (MRCP), whose goal is to provide a standard control interface to speech engines, to make them easily interchangeable (Shanmugham, Monaco, and Eberman 2006). Developed with VoiceXML servers in mind, MRCP can be adapted to other contexts. It is comparable to HTTP, with each textual request to the speech engines specifying the prompts to play and the speech grammars to listen for, and each corresponding textual responses from the speech engines giving the recognition results. MRCP makes it far easier to integrate new speech engines into a voice server, to give it better speech technologies, customize it for new locales, or simply switch to a lower cost supplier.

While MRCP is the protocol for the control of speech engines and other media resources, Voice over Internet Protocol (VoIP) standards like the Session Initiation protocol (SIP) (Rosenberg, Schulzrinne, Camarillo, Johnston, Peterson, Sparks, Handley, and Schooler 2002) and RTP are protocols for directing audio streams to and from the speech engines (Sutherland and Danielsen 2006).

The first benefit of VoIP to the voice platform architect is that specialized hardware terminating incoming PSTN lines no longer needs to be located inside the voice

server platform itself. A *media gateway* can now terminate the lines and convert their time-division multiplexed (TDM) audio streams into VoIP. This significantly drops the hardware costs and makes the overall system more flexible. Without VoIP, the incoming audio channels need to be terminated at a telephony hardware card attached to some machine in the voice server, either one running speech engines or one doing media gateway-like conversion of the TDM audio into IP packets for processing in speech engines on another machine. This means that machines must handle some multiple of the incoming PSTN line size. In North America this typically means the speech engine box has to support one or more 23-channel T1 lines, while in Europe it has to handle one or more 31-channel E1 lines. If the box could comfortably handle, say 80 incoming calls instead of 69 (three T1s) or 62 (two E1s), that extra capacity is wasted. With VoIP, the media gateway can deliver exactly the right amount to each box so that fewer are needed. And VoIP-based voice platforms can serve pure IP traffic such as Skype calls directly, with no need for a media gateway.

The combination of MRCP and VoIP also allows the architect of a voice server platform to cleanly separate the VoiceXML dialog interpreter from the speech engines and place them in various convenient and efficient topologies. For example, platforms are usually composed of self-contained “pods” of machines, each of which operates independently and handles several hundred callers. A pod supporting two hundred callers had to dedicate a speech recognizer channel to each possible incoming call, but now with MRCP and VoIP they can easily get by with, say, fifty speech recognizers, each of which is shared among many calls. For each prompt and collect cycle, a VoiceXML interpreter will use SIP to establish a session to an available speech recognizer and a media player and to set up the RTP audio pathways to each. Then the interpreter will use MRCP to tell the engines what to play to the user and what to listen for. When MRCP returns the recognition results are returned to the VoiceXML interpreter, the interpreter closes the SIP session to release the speech engines for another caller to use. This greatly increases the scalability and flexibility of the voice server platform architecture. This efficiency is possible because people interacting with voice applications spend much more time listening and thinking than they do speaking.

The W3C’s Call Control XML (CCXML) is a final standard used to open up voice server platform architectures. VoiceXML cleanly separates the application and business logic from the voice platform, MRCP provides a generic “plug-and-play” control interface to the speech engines, and VoIP standards enable very flexible internal audio pathways in the voice server platform. But the VoiceXML interpreter is still coupled to platform-dependent call control operations for accepting incoming calls, placing outgoing calls, disconnecting calls, transferring calls, etc. Call control needs to be factored out in a standard way, and this is what CCXML enables (Auburn 2007). A CCXML interpreter now becomes part of the platform, and is driven by web pages in the CCXML markup language. These tell the interpreter how to establish and tear down call legs between two or more human and computer endpoints. The platform then uses CCXML to start up sessions and to bring in new participants as needed (as in teleconferences). A VoiceXML interpreter participating in such a session implements its call control operations by sending markup to the

CCXML interpreter. Platform dependent call control interfaces are now encapsulated inside the CCXML interpreter.

We covered these topics at some length to convey something of the scale and commercial importance of the voice web, and to lay groundwork to return to later in our discussion. It turns out that this sophisticated network infrastructure, with only minor change, can support multimodal applications as well as voice-only applications. Multimodal architectures that leverage the VoiceXML-based voice web ecosystem will therefore have significant commercial advantages.

14.1.3 Multimodal User Interfaces

Mobile devices are physically small, making interaction with the keypad, stylus, and display relatively difficult. These difficulties are compounded when we have accessibility problems like arthritis or poor eyesight. “Situational impairments” are also problematic: we may be wearing gloves, walking on an uneven sidewalk, or trying to read the screen in bright sunlight. Various user studies quantify these difficulties: a joint study at Columbia and Google analyzed one million Google Mobile Search queries and found that the average time to enter even short one to four character search terms on a mobile keypad was over 40 seconds, with 30-34 character searches taking over 90 seconds. Stylus input was faster, at 25 and 50 seconds respectively (Kamvar and Baluja 2005).

These times are very problematic from a usability standpoint. But while mobile devices are poor at keypad entry, they are highly optimized for audio interaction, which makes voice input especially attractive in mobile search: assuming the speed and accuracy of the system is high enough, speech entry of search terms can take just a few seconds.

But pure speech applications have their own issues. We do not want to blurt out personal information, and complex spoken output is much harder to remember than visual output. Speech interfaces have their own accessibility issues, eg, for people with accents and hearing problems, and they have associated situational impairments such as background noise and laryngitis.

Conveniently, the weaknesses of mobile visual user interfaces are offset by the strengths of speech user interfaces: while it is slow and difficult to type (or even spell) *Albuquerque* in a mobile airline application, it is quite fast and easy to say it. And likewise, the strengths of visual interfaces offset the weaknesses of speech interfaces: visual information often is faster to process and remember than spoken information, while disambiguation of speech input can be done quickly with a visual drop-down menu of the alternatives (Oviatt 2000). The weaknesses of one modality are offset by the strengths of the other, which makes mobile multimodal applications very attractive (Suhm, Myers, and Waibel 2001).

14.1.4 Distributed Speech Recognition

Speech recognizers should be given the highest quality audio input to reduce misrecognition, but telephony channel quality is generally not of the best quality. Land-

lines deliver only about 4 kHz of bandwidth, though they are circuit-switched and tend not to drop segments of audio. Mobile audio channels use codecs that favor low bandwidth over audio fidelity, and they also drop packets. IP telephony channels can also drop packets, but their codecs can use more bandwidth.

To deliver high-quality audio to speech recognizers over mobile channels, the ETSI Aurora group developed the Distributed Speech Recognition (DSR) standards (Pearce 2000). They achieve this by moving the earliest stage of audio processing from the speech recognizer to the mobile device. This stage converts the raw audio into a digital stream of audio samples, called feature vectors. These are encoded in an RTP stream transmitted by a UDP/IP data channel to the speech recognizer. In this way channel loss is reduced, the fidelity of the audio signal is kept high, and bandwidth is reduced. Moreover, DSR front-ends on mobile devices can do special processing to eliminate background noise, approximately halving the error rate due to background noise (Pearce 2004).

These benefits are substantial, but compete against alternative approaches such as using existing audio channels and accepting higher recognition error rates, or shipping the full raw audio over reliable broadband connections to the speech server. The best chance for widespread adoption of DSR will be to pair it with distributed multimodal systems, since its benefits are synergistic with those of multimodal systems.

14.1.5 Multimodal Architectures

The Open Mobile Alliance (OMA) is a standards group formed in 2002 to develop open standards for the mobile phone industry. It consists of mobile operators, device manufacturers, software vendors and others. One of their working groups is Browser Technologies, and a subgroup called Mobile Application Environment recently published a conceptual multimodal architecture (Open Mobile Alliance 2006).

This architectural view is at a high enough level to cover cases where the speech engines are in the network and cases where they are embedded on the device. Figure 14.2 illustrates the key entities in their architecture. Each user interface modality is controlled by a *user agent* (UA), which has zero or more *processing engines* (PEs) supporting it. A web browser is a canonical example of a user agent for the visual modality; one of its processing engines might be its input processing subsystem, another processing engine could be the subsystem that renders output using HTML, CSS, and so on. Similarly, a VoiceXML-based voice browser is a user agent: its speech recognizer is a processing engine for speech input, and its audio output subsystem, which includes speech synthesis, forms a second processing engine.

Multimodal systems are those that have at least two user agents (modalities). Typically, they are comprised of a visual modality and a voice modality, but many other combinations are possible. For example, we can add in a third modality for hand-writing recognition and, perhaps, cursive handwritten output. A haptic modality can be driven by motion input detected by a three-axis accelerometer and can generate motion output by causing a transducer to vibrate at various frequencies: using it you could turn pages by flicking the phone left and right, and get feedback when you try to go beyond the first or last page through feeling a particular vibra-

tional pattern. A pulse sensor could be part of an input-only modality used in wellness applications: during physical activity, the pulse modality can be linked to an audio output modality that coaches the user on how intensely to exercise.

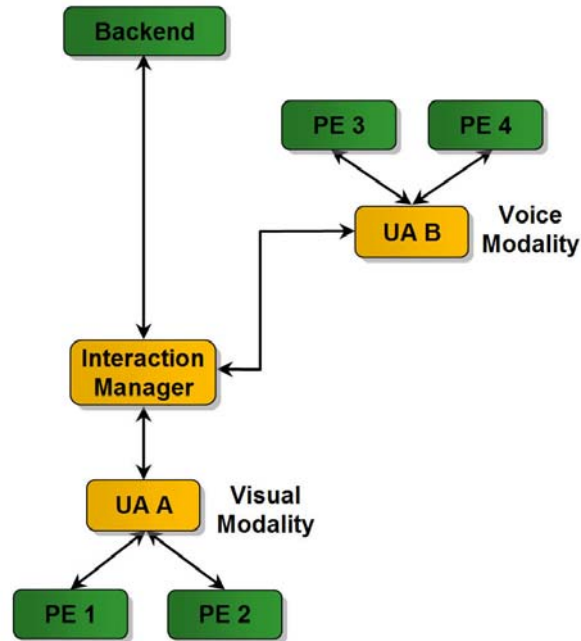


Fig. 14.2. Conceptual multimodal architecture (OMA).

Some fairly unusual multimodal systems have been developed using modalities other than speech. A head-mounted sensor can track one's gaze to determine what is being looked at, and therefore forms a sort of ocular input modality. An "emotional" input modality is even within reach of current technology: several startups are working with low-cost electroencephalogram (EEG) sensors that measure "focus" and "tranquility". At the Consumer Electronics Show in 2006, a startup called NeuroSky demonstrated a multimodal computer game with three modalities: (1) a standard computer display showing a 3D world of objects, (2) a head-mounted gaze sensor to pick out what object the player is looking at, and (3) a head-mounted "emotion" sensor that measured focus and tranquility. The system caused objects looked at with a high degree of focus to be moved closer to the player, and caused objects looked at with tranquility to float off the floor (NeuroSky 2007).

When using a speech modality, the visual modality need not be a standard form-based interface or web browser. It could be a game engine (Zyda, Thukral, Jakatdar, Engelsma, Ferrans, Hans, Shi, Kitson, and Vasudevan 2007) or an avatar interface.

Modalities may or may not support both input and output. A voice modality can have speech recognition but not generate audio prompts. Or the visual modality can

be used for output but not for input. An example illustrating this point is a Bluetooth service discovery application that features speech input and visual output to connect rapidly to location-based services (Engelsma and Ferrans 2007).

Returning to the OMA conceptual architecture in Figure 14.2, there is a need to coordinate the user agents (modalities). For instance, the results of a speech recognition may affect the visual display (ie, field values are updated) or a typed input value affects the active speech recognition grammar. The *interaction manager* (IM) is the OMA architectural element that effects this coordination: it synchronizes the data and execution flow between the user agents.

Multimodal applications generally do not operate in a vacuum and therefore must obtain external information and update external state. A weather application needs to look up weather conditions, download spoken weather reports, download radar images and other visuals, and download advertisements. It also might upload user preferences. The OMA architectural element representing the external world is called the *backend*, and the IM communicates with the backend. The backend is generally the web and all its applications and services, but in a self-contained system it might be a local web server, a set of local files, etc. The minimal backend is probably a static specification file defining a multimodal dialog.

In practice of course a real multimodal system will differ from this ideal view. A visual web browser's processing engines are not necessarily distinctly separable, since input and output have close cross linkages. And it is very common for each user agent to fetch needed resources directly from the backend rather than use the IM as a client-side proxy. But overall, the OMA model is a very helpful tool for understanding and comparing variant multimodal architectures.

Looking again at Fig. 14.2, one can draw a horizontal line across it at various heights to effect divisions between client and server components. Each possible division defines a class of multimodal architectures. Draw the line at the top, with only the backend above it, and it describes the family of multimodal architectures with everything resident on the device. Draw the line above the visual modality's user agent ("UA A"), and you describe a family where everything but the visual user agent is in the network. We will explore these families in more depth later.

14.1.6 Simultaneous and Sequential Multimodality

Multimodal systems are divided into two broad categories depending on whether the user interacts with the modes simultaneously or not. In a *simultaneous multimodal* system, more than one mode is active at the same time. In a *sequential multimodal* system only one mode is active at any time. A simultaneous multimodal map application could both display a map and play a voice prompt at the same time, and allow input by keypad, touch screen, or voice at any time. For instance the user could select a "zoom in" menu item or say "zoom in" (Maes and Saraswat 2003).

A sequential multimodal map application would only have one mode active at a time. For example, the user could place a voice call to establish the current location and the destination for a trip, hang up, and then start a visual application that downloads this information and the turn-by-turn directions for that route. Or in a

sequential stock trading application the user might again interact first by voice, then later get an SMS or multimedia message containing a trade confirmation. Sequential multimodal systems offer some of the same advantages as simultaneous multimodal systems, but are less complex to architect and implement.

Simultaneous multimodal systems are further subdivided into *composite* and *non-composite* multimodal systems. In a non-composite multimodal system the inputs from the various modalities are independent and are presented to the application in the order that they occur, even if they occur at nearly the same time. In a composite multimodal system, inputs from two or more modalities that occur at or close to the same time are considered to be a single coordinated input, so they must be composed or “fused” into a single input before being given to the application. Take an application for finding out movie theater shows and show times. In a non-composite approach the user might first select a theater from a list or a map, and then a moment later say “show times, please”. In a composite approach, the user might draw a circle around a theater push-pin on the map while saying “show times” (Maes 2003). Composite multimodal systems are potentially faster and easier to use, but have not yet been introduced commercially.

14.1.7 Mode Composition

The OMA architectural model supports hierarchical decomposition: a user agent can itself be decomposed into an interaction manager and two or more lower level user agents. For example, consider adding a new voice modality to an existing user interface that supports visual output, input from the keypad, input from a virtual keyboard with touch screen and stylus, and input by handwriting recognition with the stylus. The existing user interface is already multimodal, and so must consist of an interaction manager, a couple of lower level user agents, and some internal processing engines (eg, the handwriting recognizer). To add the new voice modality then, one has to add the voice modality’s user agent and processing engines, and couple the voice user agent to the existing system with a new higher level interaction manager.

14.2 Classes of Multimodal Architectures

We now turn to how best to architect a multimodal system. We consider only simultaneous multimodality: sequential systems are a kind of “degenerate” case of simultaneous multimodality where a relatively lengthy context switch has to take place to shut down one mode and activate another. Simultaneous multimodal systems require much tighter coordination, and hence are more difficult to architect than sequential multimodal systems.

Architecting a multimodal system is a complex process with no one right solution: each family of multimodal architectures has its own comparative advantages.

Figure 14.3 shows the OMA conceptual multimodal architecture with five alternative horizontal dividing lines between client and server. Each division identifies a family of simultaneous multimodal architectures. We consider only the very com-

mon case of a visual modality plus a voice modality: other architectural families are possible when combining other modalities.

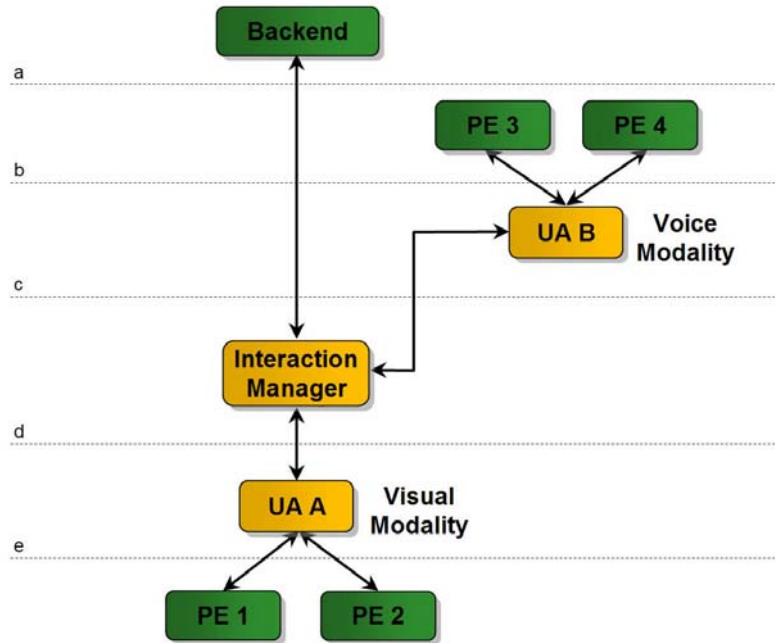


Fig. 14.3. Five families of simultaneous multimodal architectures.

14.2.1 Fully Embedded or “Fat Client” (a)

Let’s consider each class in turn. First we consider the case where every component is placed on the mobile device.

Dividing line (a) places only the backend on the server¹. All other components are on the mobile client: the visual modality, the voice modality, and the interaction manager linking them together. This approach is necessary if multimodal applications must operate when the device is not connected to a network, but it requires a fairly powerful device. On the surface it would seem to be the class of multimodal architecture that makes the least use of network bandwidth, but that depends in large

¹ Here and subsequently we gloss over the special, and relatively rare case where the application backend is entirely local to the client device.

part how self-contained the speech applications are. An embedded driving direction application with voice entry of addresses would need to download huge speech grammar files for each town or postal code, but a networked driving direction application would only have to send a relatively short audio stream up to the voice server.

One instance of this architecture is a prototype created by IBM and Opera on a Windows Mobile handset (Kennedy 2005). In this prototype, the visual user agent is the Opera XHTML browser, and the voice user agent is an IBM embedded VoiceXML 2.0 interpreter. The processing engines for the voice user agent are from IBM (embedded ViaVoice). The interaction manager is IBM client middleware.

This prototype's demonstration application was voice-activated local search. Search terms were entered by voice, and after each term was recognized on the device, it was sent to the Yahoo local search web service to obtain the results. Mobile local search is a very compelling multimodal application: it is valuable to people, requires rich visual output, and works far better with voice input than with keypad input. The IBM application is authored in the XHTML+Voice Profile (X+V) markup language, a clean unification of XHTML for the visual modality and VoiceXML for the voice modality (Axelsson, Cross, Ferrans, McCobb, Raman, and Wilson 2004).

14.2.2 Distributed Processing Engines (b)

Dividing line (b) defines the class of multimodal architectures where the speech engines are distributed to a network-based voice server, but nothing else is. (A variant on this would be to distribute the speech recognizer, but leave the speech synthesizer on the device.) The natural protocol to communicate with distributed speech engines is MRCP, which as described above is the IETF's textual protocol, patterned after HTTP, that sends prompt-and-collect requests to the speech engines and gets recognition results in the corresponding responses (Shanmugham 2006). Before we talk about this family of architectures in particular, we will take a lengthy discursion into the benefits of placing speech engines in the network.

There are some very highly significant advantages to distributing speech engines. If they are on the device they take up substantial memory, even though only a minority of device owners may be using them. They are also compute-intensive, which can make battery drain an issue (Delaney, Simunic, and Jayant 2005). Administration is far easier if speech engines are on the network: it is much more efficient to patch the speech recognizer on a thousand voice servers than ten million mobile devices. The speech application itself is much easier to tune and update in a distributed architecture: usability experts can listen to recorded sessions to find places where users run into difficulties, and use that data to revise prompts and tune speech grammars. Testing itself is much easier, since only one set of speech engines must be tested, not a multiplicity of speech engines and versions on scores and hundreds of different types of mobile device.

A final advantage of distributing speech engines to the network is that it can greatly minimize network traffic and delay in many common scenarios. The speech recognizer needs to have both the audio to recognize, and compiled speech recognition grammars to tell it what to look for. The audio originates on the handset, while

the grammars originate in the backend application. There are two pathways into the speech recognizer: the inexpensive high-speed wired network, or the expensive, slower-speed wireless network. The relative size of the audio and the speech grammars, the frequency of change in the speech grammars, and the speeds of the two networks all must be taken into account by the architect in deciding where to place the speech recognizers.

One anti-pattern to avoid is using an embedded speech recognizer driven by huge frequently changing speech grammars generated in the network. For example, if the user is browsing an online music store with five million songs divided into a hundred categories of 50,000 titles per category, with new titles added each day, then each day and for each category the user triggers a speech recognition grammar download of perhaps five megabytes, and an embedded grammar compilation step that together might take five to ten minutes and substantial battery power. In this case, it is far better to send up a couple of kilobytes of DSR-compressed audio to the voice server: the results will be back in a couple of seconds, and the battery will barely be affected. This tradeoff turns out to be fairly common: think of mobile search, map applications with points of interest being added and removed each day, corporate directory access, access to back end enterprise data, looking for auctions on eBay, ordering books from Amazon, and so on².

On the other hand, if the application backend is on the mobile device, it is better to do the speech recognition on the device, otherwise the device would have to generate a potentially large speech grammar for the network-based speech recognizer.

This tradeoff is captured in the Pearce Principle (Pearce 2002), which states that speech recognition should be done at the point closest to the location of the speech grammar being listened for. This provides a rationale for *hybrid* speech recognition systems, which leverage local recognition for local applications, and remote recognition for network-based applications³. This is similar to the data-intensive supercomputing principle of locating computation where the data resides, rather than moving the data to the point of computation (Bryant 2007). This insight has also long been known in the area of query processing in distributed databases.

Returning from our discussion into the virtues of distributed speech engines, the Distributed Processing Engines family of distributed multimodal architectures has a

² One optimization would be to do the grammar compilation in the network instead of the device, but then each application needs to have the grammar compiler for each possible mobile device configuration, and know each device's configuration, a complex task. Even if this reduced data transmission, battery drain, and elapsed time by an order of magnitude, the resulting delay would still make the experience very painful for the user.

³ When high-quality embedded transcription engines become practical, and application developers take advantage of them, the dynamics change: transcription systems do not use speech grammars.

significant disadvantage in that it requires MRCP or a protocol at the same level to go over the wireless network to the server hosting the speech engines. This is relatively expensive in terms of bandwidth and round trips: MRCP was designed to be a lower-level protocol used within a voice server platform and hence it has many more, and much larger messages than a higher level protocol would have.

14.2.3 Thin Client (d)

We will return to architectural family (c) after we discuss (d) and (e).

Family (d) is the “thin client” multimodal architecture. This places the full voice modality in the network, along with the interaction manager. This approach is fairly balanced for contemporary mobile devices and networks. It turns out to be second best in terms of network bandwidth, but there can be some awkwardness in writing applications where some logic has to be broken out into an explicit interaction manager off in the network. But overall it shares many virtues with family (c), which we believe edges it out in desirability.

14.2.4 Remote Visual Interface (e)

With the dividing line drawn beneath the visual user agent, as in (e), we have an architecture class where everything but the visual user interface rendering and the input subsystem is distributed to the server. A protocol for driving a remote user interface needs to be developed, and the mobile device just contains a module that does the lower levels of the visual user interface. Most of the logic driving the visual user interface is in the network.

This is the same approach that the X Window System takes. One instantiation of the Remote Visual Interface architecture would be to put an X Server on the mobile device and drive it from an X Client in the network. They would communicate with the X11 protocol. In this approach, the X Server corresponds to the OMA visual processing engines and the X client corresponds to the OMA visual user agent.

Auvo, an early multimodal startup (ca. 2000-2002), used this architecture, but unfortunately they were years ahead of the market and ran out of funding.

The main drawback of remoting the visual user interface over a mobile data network is of course bandwidth and latency. Bandwidth is becoming less and less important, but a protocol that introduces many round trip delays will be less useful than one that has few delays. Another drawback is that this architecture makes it very hard to expose the full power of the native visual user interfaces on each device: it almost invariably presupposes that each device runs a client that understands a “least common denominator” protocol and API.

14.2.5 “Pudgy” Client (c)

The final major family of multimodal architectures is described by dividing line (c), the so-called “Pudgy” Client. This is a slight variation on Thin Client, moving the

interaction manager from the server over to the mobile client. This makes it a bit fatter than Thin Client, hence the name.

This approach is more optimal in terms of network usage (the interaction manager has somewhat more work to do to drive the visual interface than the voice interface, and hence should be located with it). It is more intuitive for developers, who tend to view the mobile client as the proper locus of control, just as it is for purely visual applications. The notion that voice is a sort of supplemental input method under control of the client software has proven to be especially appealing.

We cover an implementation of Pudgy Client at length in Section 14.3.

14.2.6 Discussion

We have just described five main families of distributed architectures that support simultaneous multimodal interaction.

The Fully Embedded architecture is well-suited for more powerful devices and applications that reside on the device itself. It has trouble running applications that require significant fetching of speech grammars from a network-based source, since these can take very significant amounts of bandwidth and time to download and compile. Embedded speech engines lead to various administration difficulties, and also make voice application testing more complex and problematic. Nevertheless, devices and networks are both gaining in power and speed, diminishing some of these difficulties. As device-based speech recognition becomes more transcription-based (open vocabulary), the need for speech grammars will diminish. We therefore believe that this will be an effective architecture going forward.

The four remaining approaches leverage network speech engines and do not share many of the above limitations. On the other hand, they cannot be used in a disconnected mode. Of the four, Distributed Processing Engines requires the client to exercise detailed low-level control and therefore requires more network message round trips, introducing delay. The Remote User Interface also requires a lot of network traffic, and pushes off on the multimodal server a lot of user interface control logic, which means that the server has to support a single generic abstract visual user interface with least common denominator functionality, and that therefore the native user interface capabilities of each device cannot be fully leveraged.

The Thin Client and Pudgy client architectures are both nice balances that minimize network traffic and are easy to develop applications for. Of the two we have a moderate preference for Pudgy: it is more natural to have the interaction management done close to the visual modality than the voice modality, as the client application is the natural locus of control.

14.3 The “Plus V” Distributed Multimodal Architecture

Motorola began working on a distributed multimodal system connected to a standard VoiceXML server in the network in 2001. Our initial architecture was primarily Thin Client (d), with the interaction manager consisting of a few extensions to the

VoiceXML Form Interpretation Algorithm, so the voice dialog actually drove the visual dialog as a side effect. We quickly found this to be awkward and unnatural, as developers believed interaction management belonged in the client device.

In early 2002 we tried another approach, where the interaction management was explicitly made a module in the client software. This was an implementation of Pudgy Client. A major motivating factor was a series of unpublished simulation studies we did to evaluate the architectural families. The goal was to determine bandwidth and latency costs of each approach on GPRS networks. We found that Pudgy Client was much better overall than the others we tested. Our subsequent implementations confirmed this: on the 2.5G GPRS and the 2G iDEN networks, our system takes between 0.8 and 2.0 seconds between the end of speech and the visual display of the recognition result, substantially faster than even today's multimodal systems running on 3G data networks. This speed is due to Pudgy's low messaging requirements, its terse binary message format, and the use of the DSR codec, which takes only 5.6 kbps of bandwidth, on the audio channel.

In this approach, the client is fully in charge of the interaction. The networked VoiceXML server is under its control and merely adds the voice modality to the interaction, hence the architecture's more formal name "Plus V". A key advantage of Plus V is that it supports any visual user interface. We have created three instances: one that connects a Java J2ME MIDlet on the handset to the networked voice server (J+V), another that connects a C++ application using Qt user interface on Linux handsets to the voice server (Qt+V), and a third that connects a version of the Konqueror XHTML browser using the X+V multimodal markup language (Axelsson 2004). Any visual interface can be supported: for instance the Torque 3D game engine could be used in a "Torque+V". This agnosticism to the graphical user interface is a strong advantage.

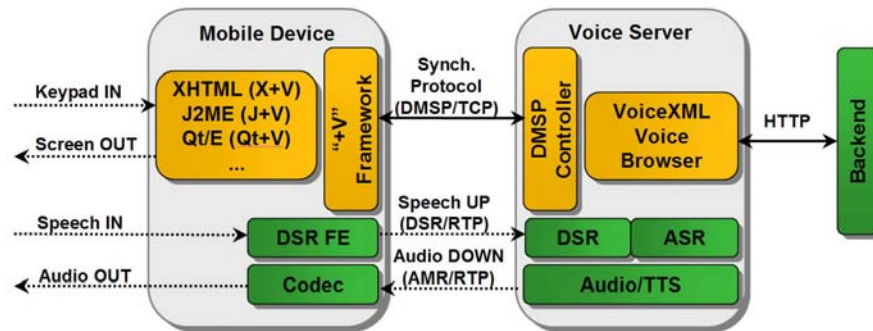


Fig. 14.4. Plus V multimodal architecture.

Figure 14.4 shows the Plus V multimodal architecture at the next level of detail. The voice server is a very slightly modified VoiceXML server. We started with the commercial SandCherry Voice Platform (see www.sandcherry.com) and dropped in

the commercial Motorola VoxGateway VoiceXML 2.0/2.1 interpreter (Ferrans 2003) and the Nuance OSR 3.0 speech recognizer, which supports the DSR codec.

On the client we have the standard codec for audio output, a DSR front end to do the encoding of the audio, a native user interface, and one of the Plus V implementations as described above.

In the OMA terminology, the visual user interface and the VoiceXML voice browser are the user agents, the speech engines are the voice modality's processing engines, and the visual modality's processing engines are elements of the graphical user interface software. The interaction manager is represented by the Plus V device-side framework (the client application can do some interaction management).

The client drives the voice server using the Distributed Multimodal Synchronization Protocol (Engelsma and Cross 2007) over a reliable TCP/IP channel. The client tells the voice server which VoiceXML page to load and which VoiceXML dialog on that page to run. Once the dialog is running, if the user speaks to the system, the voice server uses DMSP to convey the recognition result back to the client. If the user types, the Plus V Framework sends the new field value to the voice server via DMSP, where it causes the VoiceXML dialog to advance. If the user scrolls through the visual form's fields, the client also tells the voice server the new focus field. This level of coordination is necessary because each visual field may have a distinct audio prompt introducing it, and each field typically also has a speech grammar associated with it. Mixed initiative dialogs are also supported by this approach. DMSP is currently an IETF Draft, and for performance it seeks to minimize messages, message size, and round trips. The message format is a very condensed binary format, with an optional XML format for use when message size is not an issue.

The efficiency of DMSP and of the DSR speech recognition codec makes Plus V the fastest distributed multimodal architecture we are aware of in terms of recognition response latency. As mentioned above the time between the raising of the push-to-talk key and visual confirmation of the user's speech runs between 0.8 and 2.0 seconds on the 2G iDEN network, and 1.0 and 3.0 seconds on the 2.5G GPRS network. These times are also at least as fast as the embedded speech approaches we are familiar with.

The DMSP protocol has its client endpoint inside the Plus V Framework; its voice server endpoint is the DMSP Controller. The Controller in turn has some hooks inside the VoiceXML interpreter's main loop: the Form Interpretation Algorithm (FIA), which determines what field to prompt and collect at each iteration. The FIA just needs to stop and check for commands coming from the client, and if it is in its listen phase when control commands come in, it needs to break out of that speech recognition to see what to do next. It was not at all hard to make this modification: we estimate that the effort needed to multimodal-enable a VoiceXML interpreter is at most two or three percent of the effort needed to write that interpreter.

14.4 Other Distributed Multimodal Architectures

Plus V is by no means the only way to architect a multimodal system. In this section we briefly sketch several other commercial distributed architectures. The goal is to show how varied the solutions are, not to exhaustively enumerate them.

14.4.1 Video Interactive Services with VoiceXML

In 2005 several people realized that VoiceXML could be adapted to video telephony quite easily. It already supported the playback of recorded audio, identified by URL and media type. It also already supported the recording of audio, of a given media type, and the posting of that audio to a web server. Why not plumb the voice platform to carry mixed audio and video streams via SIP and RTP, link those streams to the mobile handset, and support the idea of video prompts and video recordings?

This turned out to be relatively straightforward, and the only impact on VoiceXML itself was a desire to generalize the name of the “audio” prompt element.

The resulting platforms support multimodal applications that combine voice and video modalities. A video answering machine application can play different video prompts based on the caller, and take video messages from callers. Support applications can now show videos of procedures and accept videos showing problems to support representatives. Many other interesting multimodal applications are enabled by this approach (Burke and McGlashan 2006).

Because the modes used are voice and video, these systems do not fit neatly into our architectural families, but it is somewhat analogous to the Remote User Interface (e). The drawbacks of the Remote UI approach do not apply when using a video user interface instead of a graphical user interface: video playback is very standard and not highly interactive.

14.4.2 Multimodal for Set-Top Boxes

PromptU (www.promptu.com) began a few years ago as a company specializing in using voice to interact with the electronic program guide (EPG) displayed on televisions via the cable operator’s set-top boxes. The EPG application runs on the head-end equipment in the operator’s infrastructure, and is controlled by keys on the television remote.

In the PromptU system, the remote is augmented by a microphone and a push-to-talk button. When the user speaks (“Find actress Penelope Cruz”), the audio from the remote goes to the set top box, where it is encoded by an Aurora DSR Front End (Pearce 2000) and sent up to a voice server located in the head-end. The voice server runs an application that maps the voice commands into actions on the EPG, and the output is sent back to the set-top box for display on the television.

More recently, PromptU has been moving into the general mobile multimodal application space, supporting music download, ring tones, games, and so on. The PromptU architecture is in the Thin Client family.

14.4.3 Bare Minimum Mobile Voice Search

Plus V was developed at a mobile handset company, where we had luxuries to do things that others cannot. We wrote DSR front end encoders for DSP chips, ensured that audio packets could be streamed using RTP, and even influenced the future MIDP 3.0 J2ME implementation.

A company that wanted to get a multimodal application out to its customers on as many handsets as possible would have to start from a different point, deploying a system that made the least possible assumptions about those handsets, and then influencing the industry to add the sort of enablers that we put into Plus V. Let's assume this company is doing a mobile multimodal search application.

By necessity, this company would choose a distributed architecture, since that offloads a huge amount of complexity and variability from the mobile devices. On the client they would probably select Java J2ME for its ubiquity. Their Java client application would present the visual interface, use the JSR 135 Mobile Media API to gather voice input, and use HTTP to post that audio up to the server. Along with the audio, the HTTP request would contain the location, from GPS or the carrier's cell tower ID information. The request might contain a cookie identifying the user, and perhaps other contextual information.

On the server receiving this request runs the server side of their application. This first would send the audio over to a speech server for recognition, a process that probably would take into account the user's desired search location and radius. The speech server sends back the results, and the server-side search application feeds them to the existing web services API for the search service. At the same time the server-side application could interact with an ad server to get contextually relevant advertising to show the user. The server-side application then sends back the HTTP response with the search results, advertisements, and other response information.

The architecture described is not highly optimized or general, but it can be improved on handsets that support streamed audio, and if the application is successful, the industry will quickly try to add enablers to improve the user experience.

14.4.4 A Transcription-Based Architecture

Our last example architecture is from Mobeus, a startup just coming out of stealth mode in May 2007 (<http://www.podtech.net/scobleshows/search/Mobeus>). They have server side technology for doing speaker-independent transcription, which is ideal for mobile multimodal search applications, voice to SMS and email applications and so on. Their view of how this should be integrated with a visual user interface on the client is radically simple: provide a text entry widget connected to this transcription server, plus controls for speaking into it and editing the result to correct any errors or select from the "n-best" alternatives for each word. The results are very impressive, and while again it may not be the fastest or most general system (audio prompting is not addressed, eg), at this stage these sorts of approaches can unlock a lot of value.

14.5 Towards a Commercial Ecosystem

The World-Wide Web Consortium (W3C) has been working in the area of multimodal standards since early 2002. Progress has been slow mainly because of a lack of early proprietary implementations, but as we have seen above this is soon going to change. As the value of multimodal systems becomes apparent, there will be a renewed push to create interoperability standards to grow the industry. Where do things stand today?

The W3C Voice Browser and Multimodal Interaction working groups (www.w3.org) are working on a future markup language. This will be philosophically similar to X+V (Axelsson 2004) in that a combination of XHTML and VoiceXML is called for. The framework that integrates the two markup languages will be a markup language called State Chart XML (SCXML) which is closely patterned on David Harel's State Chart formalism (Harel 1987). The challenge will be to create a language accessible enough to attract developers from ad hoc approaches.

The W3C is also working to "modularize" VoiceXML into a subset appropriate for use in a multimodal system (for instance it makes no sense for the executed VoiceXML to do call control operations like disconnect in a multimodal configuration). They are also revising VoiceXML's stand-alone event model to allow control events to come in from external sources, a task necessary if VoiceXML interpreters need to be controlled by interaction managers.

The IETF is to protocols what the W3C is to web markup languages. We have described at a very high level one such control protocol between the interaction manager and user agents: DMSP (Engelsma 2007) which has been submitted to the IETF as an Internet Draft. The outcome of this submittal is not yet clear, but it will probably take the upcoming impetus of successful proprietary multimodal systems to push this forward.

The 3GPP, an industry standards body focused on GSM standards, has approved the use of DSR for multimodal applications, and 3GPP2, the parallel organization for CDMA standards, is also considering it. DSR should offer continued incremental benefits even in a world of huge bandwidth.

Other standards would be needed to mature this ecosystem. There needs to be a standard for how a control protocol like DMSP drives a VoiceXML interpreter, perhaps a standard for authoring languages other than the W3C's StateChart-based one (eg, for Java or C++ application authoring), standard APIs for integrating XHTML browsers on the mobile device, and so on.

14.6 Conclusions

Multimodal user interaction is very natural and is about to become a common part of our lives. Systems like our Plus V platform demonstrate conclusively that multimodal technology is practical, fast, and efficient even on older mobile data networks. Speech recognition has advanced to the point where complex and commercially

important applications like mobile voice search, voice media search, and voice to SMS and email transcription can be implemented.

Commercial interest from companies like Google, Microsoft, and Nuance is very high and focused in the area of multimodal local search. It seems inevitable that Google will merge their new 1.800.GOOG411 voice directory assistance application in with their visual Google Local Mobile. Microsoft paid \$800m in early 2007 to acquire TellMe for their deep experience in voice directory assistance and driving directions. Nuance has acquired at least two companies with multimodal capabilities, Lobby7 and Mobile Voice Control, and acquired BeVocal for their application hosting capability. Japanese mobile operator KDDI deployed the EZ Navi Walk pedestrian navigation multimodal application (with DSR) in late 2006. Other players like Yahoo, PromptU, V-Enable, Kirusa, and VoiceBox are entering this arena. All of these are deploying distributed multimodal architectures.

This wide range of proprietary architectures will inform standards efforts at the W3C and elsewhere. Multimodal interaction will remain a fruitful area of research, especially as other innovative modalities are developed.

References

- Atkins, D., Ball, T., Baran, T., Benedikt, M., Cox, K., Ladd, D., Mataga, P., Puchol, C., Raming, J.C., Rehor, K., and Tuckey, C. (1997) Mawl: integrated web and telephone service creation, *Bell Labs Technical Journal*, 2 (1), pp. 19-35.
- Auburn, R. (2007) Voice browser call control: CCXML version 1.0, W3C Working Draft, <http://www.w3.org/TR/ccxml/>
- Axelsson, J., Cross, C., Ferrans, J., McCobb, G., Raman, T., and Wilson, L. (2004) XHTML+Voice Profile 1.2, *VoiceXML Forum*, March 2004, <http://www.voicexml.org/specs/multimodal/x+v/12/spec.html>
- Boyer, L., Danielsen, P., Ferrans, J., Karam, G., Ladd, D., Lucas, B., and Rehor, K. (2000) Voice Extensible Markup Language (VoiceXML) version 1.0, *VoiceXML Forum*.
- Bryant, R. (2007) Data-Intensive Supercomputing: The case for DISC, *CMU Technical Report CMU-CS-07-128*. May 10, 2007.
- Burke, D. and McGlashan, S. (2006) Video interactive services with VoiceXML, *VoiceXML Review*, 6(2), March/April 2006, http://www.voicexml.org/Review/Mar2006/features/video_interactive_services.html
- Delaney, B., Simunic, T., and Jayant, N. (2005) Energy-aware distributed speech recognition for wireless mobile devices, *IEEE Design and Test of Computers*, 22(1), pp. 39-49, Jan/Feb, 2005.
- Deng, L. and Huang, X. (2004) Challenges in adopting speech recognition, *CACM*, 47(1), pp. 69-75, January 2004.
- Engelsma, J., and Cross, C. (2007) Distributed Multimodal Synchronization Protocol, *IETF Internet Draft*, (Work in Progress), January 2007.
- Engelsma, J., and Ferrans, J. (2007) Bypassing Bluetooth device discovery using a multimodal user interface, In *Proc. of the 4th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2007)*, Philadelphia, PA, August 2007.
- Ferrans, J. (2003) The Motorola VoxGateway, lessons learned, *VoiceXML Review*, 3(4), July/August 2003, <http://www.voicexmlreview.org/Jul2003>.

- Harel, D. (1987) Statecharts: a visual formalism for complex systems, *Sci. Comput. Programming* 8, pp. 231-274.
- Kamvar, M. and Baluja, S. (2005) A large scale study of wireless search behavior: Google Mobile Search, *Proc. ACM SIGCHI Conf. on Human Factors in Computing Systems (CHI 2005)*, pp. 701-709, April 2005.
- Kennedy, N. (2005) Igor Jablovok interview on multimodal search, 16 October 2005, http://www.niallkennedy.com/blog/archives/2005/10/igor_jablokov_interview_on_mul.html
- Ladd, D., Hay, M., McClaghrey, P., and Ferrans, J. (1999) VoxML 1.1 Language Reference, <http://www.w3.org/Voice/1999/VoxML.pdf>.
- Maes, S. and Saraswat, V. (2003) Multimodal interaction requirements, W3C Note, <http://www.w3.org/TR/mmi-reqs>
- McGlashan, S., Burnett, D., Carter, J., Danielsen, P., Ferrans, J., Hunt, A., Lucas, B., Porter, B., Rehor, K., and Tryphonas, S. (2004) Voice Extensible Markup Language (VoiceXML) version 2.0, W3C Recommendation, <http://www.w3.org/TR/voicexml20>.
- Neurosky (2007) <http://www.neurosky.com>.
- Open Mobile Alliance (2006) OMA multimodal and multi-device enabler architecture, OMA-AD-MMMD-V1_0-20061011-D, October 2006. http://member.openmobilealliance.org/ftp/Public_documents/BT/MAE/Permanent_documents/OMA-AD-MMMD-V1_0-20061011-D.zip
- Oviatt, S., (2000) Taming recognition errors with a multimodal interface, *CACM*, 43 (9), pp. 45-51, September 2000.
- Pearce, D. (2000) Enabling new speech driven services for mobile devices: an overview of the ETSI standards activities for Distributed Speech Recognition front-ends," *Applied Voice Input/Output Society Conference (AVIOS 2000)*, San Jose, CA, May 2000.
- Pearce, D. (2004) Robustness to transmission channel – the DSR approach, *COST278 & ICRA Research Workshop on Robustness Issues in Conversational Interaction*, Aug 2004.
- Pearce, D., Engelsma, J., Ferrans, J., and Johnson, J. (2005) An architecture for seamless access to distributed multimodal services, *Proc. 9th European Conf. on Speech Communication and Technology (Interspeech 2005)*, pp. 2845-2848, September 2005.
- Pearce, M. (2002) Pearce principle, private communication, January 2002.
- Raggett, D. (1999) Introduction to TalkML, <http://www.w3.org/Voice/TalkML/>
- Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E. (2002) SIP: Session Initiation Protocol. IETF RFC 3261, June 2002, <http://www.ietf.org/rfc/rfc3261.txt>
- Shanmugham, P., Monaco, P., and Eberman, B. (2006) A media resource control protocol (MRCP). IETF RFC 4463, April 2006, <http://www.rfc-editor.org/rfc/rfc4463.txt>
- Suhm, B., Myers, B., and Waibel, A. (2001) Multimodal error correction for speech interfaces, *ACM Transactions on Computer-Human Interaction*, 8 (1), pp. 60-98, March 2001.
- Sutherland, I. and Danielsen, P. (2006) VoiceXML and Voice-over-IP. *VoiceXML Review*, 6(3), September/October 2006. <http://www.voicexml.org/Review/Oct2006/features/voip.html>.
- Zyda, M., Thukral, D., Jakatdar, S., Engelsma, J., Ferrans, J., Hans, M., Shi, L., Kitson, F., and Vasudevan, V. (2007) Educating the next generation of mobile game developers, *IEEE Computer Graphics and Applications*, 27(2), March/April 2007.



Motorola, Inc.
1303 E. Algonquin Road
Schaumburg, Illinois 60196 U.S.A.
www.motorola.com

MOTOROLA and the Stylized M Logo are registered in the U.S. Patent and Trademark Office. All other products or service names are the property of their registered owners. © Motorola, Inc. 2009