FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Context-based Multimodal Interaction for Mobile Collaboration Systems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering/Internet Computing

eingereicht von

Ursula Fida
Matrikelnummer 8825745

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Univ.Prof. Mag.rer.soc.oec. Dr.rer.soc.oec. Schahram Dustdar
Mitwirkung: Mag.rer.soc.oec. Dr.techn. Christoph Dorn

Wien, 21.01.2011

(Unterschrift Verfasserin) (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Ursula Fida
Diamantgasse 40
1210 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 10.12.2010 _____

(Unterschrift Verfasserin)

Abstract

Mobile devices and wireless networks give us the possibility to use the Web where ever we go, in all situations, at all times. But devices like PDAs have small displays and limited or even no keyboards. Enabling speech as an additional input and output mode will help to overcome these physical limitations. In a multimodal application the user may use his preferred mode or the most appropriate mode for the situation or he may use voice, keyboard, mouse and pen in a synergistic way. Moreover multimodal applications allow one-handed and hands-free operations. Mobile collaboration systems handle a lot of information so it is important for the user to get the right information at the right time and at the right place. Making the speech output and input context sensitive will prevent overloading the user with unwanted information and leads to better speech recognition rates.

Zusammenfassung

Mit mobilen Endgeräten und kabellosen Netzwerken haben wir die Möglichkeit das Internet zu nutzen wo auch immer wir hingehen, in allen Situation und zu allen Zeiten. Aber mobile Geräte wie z.B. PDAs haben nur kleine Bildschirme und eine kleine oder sogar keine Tastatur. Diese physikalischen Grenzen kann man mit Sprache als zusätzlichem Eingabe- und Ausgabemodus überwinden. In einer multimodalen Anwendung hat der Benutzer die Möglichkeit den Modus zu verwenden den er bevorzugt oder der für die Situation am besten passt oder er verwendet Sprache, Tastatur, Maus und Stift gemeinsam. Zusätzlich erlauben multimodale Anwendungen ein- und freihändige Tätigkeiten. Mobile Collaboration Systeme verwalten eine Vielzahl von Daten, sodass es für den Benutzer sehr wichtig ist, die richtigen Daten zum richtigen Zeitpunkt und am richtigen Ort zu erhalten. Wenn man die Sprachausgabe und –eingabe an den Kontext anpasst, kann man den Benutzer vor einer Informationsflut schützen und bessere Spracherkennungsraten erzielen.

Contents

Abstract **4**

Zusammenfassung **4**

1. Introduction **7**

1.1. Motivation and Problem Description 7

1.2. Structure and Goal of this Thesis 9

2. State of the Art Review **10**

2.1. XML-based Languages that facilitate Multimodal Input 10

2.1.1. VoiceXML 10

2.1.2. SALT 12

2.1.3. XHTML+Voice 17

2.1.4. EMMA 22

2.2. Development Tools 27

2.2.1. Microsoft Speech Application SDK 27

2.2.2. IBM WebSphere Multimodal Toolkit 30

2.2.3. Eclipse Voice Tools 32

2.3. Speech Recognition and Speech Synthesis Technologies 35

2.3.1. Speech Recognition 35

2.3.2. Speech Synthesis 36

3. Related Work **38**

3.1. Voice-only Applications 38

3.2. Multimodal Applications 40

3.3. Context-aware Mobile Applications 42

3.3.1. Location Awareness 42

3.3.2. Working Context and Social Awareness 44

3.3.3. Environment Context and User Situation 46

3.3.4. Collaborative Working Environments 47

3.4. Composite Device Computing 50

3.5. Multimodal Architecture 51
3.6. Usability 53

4. Framework 54
4.1. Scenario – Schedule Meetings & Calls 54
4.2. Design 57
4.3. GUI and VUI Design 65
4.4. Implementation 69
4.4.1. Java on a Mobile Device 70
4.4.2. OSGi Framework 71
4.4.3. Multimodal Browser 72
4.4.4. Team Scheduling Server Application 74
4.4.5. Team Scheduling Client Application 75
4.4.6. Deployment 77
4.4.7. Deployment Problems 79
4.5. Evaluation 81

5. Conclusion and Outlook 88

List of Figures 91

Bibliography 92

1. Introduction

1.1. Motivation and Problem Description

Mobile workers use mobile computing devices like a PDA to have access to their working environment where ever they go to collaborate and communicate with other team members and to have access to their project data in all situations and at all times. But these handheld computers have small displays and limited or even no keyboards. A text may be quite difficult to type and a large table may be difficult to read under these physical limitations. Furthermore there is a need for applications allowing one-handed and hands-free operations, for example while driving a car the user needs his hands on the wheel and the eyes on the road to pay attention to the traffic. Thus users of mobile devices need additional output and input modes than the common ones like display, keyboard and mouse. Most mobile devices are equipped with speakers and microphones, so voice input and output is technically possible. An application may use text-to-speech mechanism to read out the content to the user and the user may use voice commands to fill in Web forms, select a link on a list, and start another service and so on. A voice modality can be another possible interaction mode for mobile applications on PDAs. And using speech for interacting with a device will keep the hands free which is important for mobile users.

Having speech as the only input or output mode is not practical in every situation. A user may prefer a stylus or keypad for giving inputs due to background noise or to keep his privacy. An output like a map or a diagram can hardly be presented or described by voice. With a multimodal application the user has the possibility to interact with the interface using his preferred mode or the most appropriate mode for the current situation. Furthermore he may use voice, keyboard, mouse and pen in a synergistic way. And the output can be audio as spoken prompts and playback, using text-to-speech or visually on graphical displays.

Mobile workers are often involved in more than one project and handle a lot of information in their working environment. Thus it is important for the user to get the right information at the right time and at the right place. Making the speech output context sensitive and relevance based will prevent overloading the user with unwanted information. Without adapting the speech output to the current context, the user must listen, for example, to a long list of unwanted information before he gets the needed data. Information about the current location can be used to choose the adequate mode

for the actual situation. If the user is in a meeting speech output will disturb the others, so the application should present the information only on the display. Including context and relevance information in the application to present the user only the needed information and using an adequate mode will improve the usability of the application or is even a necessity.

Context and relevance based information can also be used for improving speech input. A grammar can be pretty large or of ambiguity without having detailed information about the context. An address book or a list of contacts for example can contain two persons with the same name, one working at one company and a different person with the same name who works at another company. Context-based information can be used to identify the notional wanted person without asking the user for additional information. The speech recognition component can use context and relevance information to generate a reduced list of expected speech commands to generate well-fitting grammar rules which leads to better speech recognition rates.

Apart from these aspects making multimodal application context-aware will reduce costs such as network bandwidth and battery consumption, because only the relevant pieces of context information will be transferred, for example smaller lists or grammar rules.

This work presents a team scheduling application as an example of a context-based multimodal application for mobile collaboration systems. Starting from a simple Web application for a PDA for planning team meetings, we developed a multimodal application where the user has the possibility to use speech and text as input modalities and where the application gives the user a recommendation on the best modality for the current situation. The decision-making on the best mode, text or speech, depends on many parameters, some are dependent on the user and some are dependent on the surrounding environment. This part will be implemented in a general way so that it can be used for other applications too.

1.2. Structure and Goal of this Thesis

Chapter 1 gives a general introduction to the problem area and the motivation of this thesis and defines the goal of this work.

Chapter 2 gives a review of the languages and development tools that can be used to implement multimodal applications and a description of the underlying technologies for speech recognition and synthesis.

Chapter 3 examines related work.

Chapter 4 describes the design, implementation and evaluation of the team scheduling application.

Chapter 5 concludes the thesis, gives a brief summary of the presented work and an outlook for future work.

The goal of this thesis is to elaborate an overview of the current technologies and the present applications that support multimodal interaction with a computer system. Furthermore this research concerns with context-awareness and how integrating context and relevance based criteria can improve multimodal applications. Examining the state of the art and building a context-based multimodal application will help to have a look at the future possibilities of the Web in conjunction with mobile devices and collaboration.

2. State of the Art Review

2.1. XML-based Languages that facilitate Multimodal Input

There are three XML-based languages for multimodal development:

- SALT, the Speech Application Language Tags [58],
- XHTML+Voice, or short X+V [82] and
- EMMA, the Extensible MultiModal Annotation markup language [19].

SALT and X+V both use underlying speech engines to do the work of recognizing and generating human speech. Both languages have language elements (tags) that specify what the speech-recognition engine should listen for and what the synthesis engine should "say". EMMA is a complimentary language, it is not authored by developers like SALT and X+V, but generated by interpretation components like speech recognition engines. Moreover EMMA serves as a standard data interchange format between the components of a multimodal system and provides semantic interpretation for various modalities including but not necessarily limited to speech, natural language text, GUI and ink.

Whereas VoiceXML [74] is a standardized format for speech-only applications. VoiceXML has been developed over several years and is mostly used to develop user interfaces for phones, as used in Call Centers.

2.1.1. VoiceXML

The Voice eXtensible Markup Language, an XML-based markup language for creating distributed voice applications, was defined and promoted by the VoiceXML Forum [73]. It was founded 1999 by AT&T, Lucent, Motorola and IBM and submitted the VoiceXML 1.0 to the W3C's Voice Browser Working Group [75] in May 2000. The next major release is VoiceXML 3.0. While SALT was designed to develop also multimodal application that run on multiple devices like PDAs, smart phones, tablet PCs and conventional PCs, VoiceXML was designed to support voice-only user interfaces for telephones and cell phones. VoiceXML has become a standard for developing Interactive Voice Response (IVR) systems. With VoiceXML audio dialogs can be created featuring text-to-speech and pre-recorded audio files, recognizing spoken as

well as keystroke input (DTMF – Dual Tone Multi-Frequency), recording telephony and mixed-initiative conversations.

VoiceXML can be used to create a lot of applications in different domains. Typical voice applications best suited for VoiceXML are information retrieval, for example personal voice newsletter containing news, sports, traffic, weather and stock information. Also directory assistance and telephone services like for example personal voice dialing can easily be voice-enabled with VoiceXML. Additionally customer service applications of electronic commerce like package tracking, account status, support and catalog applications are other areas where voice services can be used. Certainly many other domains can benefit from voice-enabling their interfaces and services.

VoiceXML is interpreted by a voice browser with audio output coming from a text-to-speech synthesizer or consisting of recordings and audio input handled by a speech recognizer and keypad input. A voice browser runs on a voice gateway node which is connected to the internet and to the public switched telephone network. The voice gateway supports a large amount of simultaneous callers and can be accessed by any phone from conventional up to modern mobile phones (Figure 1).

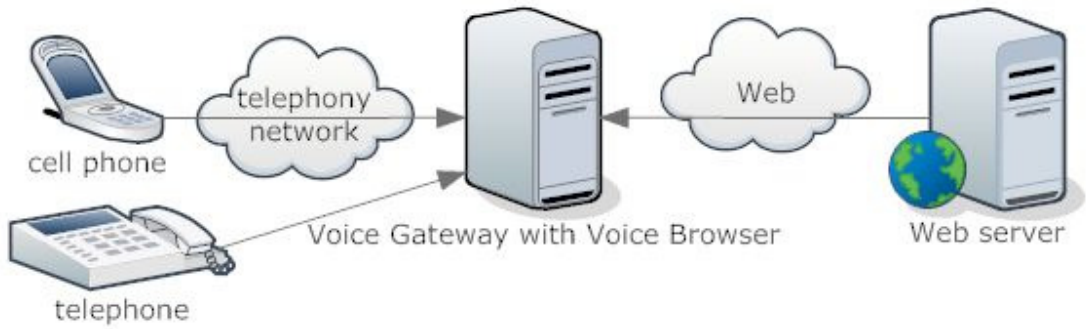

Figure 1: VoiceXML architecture

A VoiceXML application consists of one or more VoiceXML documents. Just like in all other XML documents a VoiceXML document must start with `<?xml version="1.0"?>`, the rest is surrounded with `<vxml version="2.0">` and `</vxml>`. Inside the `<vxml>` tag the document contains several voice dialogs which provide information to the user or request input from the user. In VoiceXML there are two types of dialogs: `<form>` and `<menu>`. Whereas a VoiceXML form is similar to an HTML form, saving user input into variables and doing something depending on these variables, a menu element is like a link list in HTML, providing a list to the user where he can choose out of it. The `<block>` element is one of the elements allowed within a form tag and contains text or executable VoiceXML elements. Text is queued by the VoiceXML interpreter and played to the user with the text-to-speech engine. The

`<audio src=”hello.wav”>` element can be used to play an audio file to the user. With the `<goto>` element the dialog flow can be redirected to another dialog in the current page or in another document. The element `<record name="userinput">` records text and saves it in the variable `userinput`. The result can be forwarded via a script to a server for saving it. The `<field>` element is used to get input from the user. It contains `<prompt>`s with user instructions, `<grammar>`s that describe the allowable user input and instructions that are executed when the field is filled. A variable is declared and initialized with `<var name=”abc” expr=”hello”>`, the element `<assign name=”def” expr=”100”>` sets a variable to a new value and `<value expr=”abc” />` can be used to play a variable value inside a `prompt`. A call can be forwarded with the element `<transfer dest=”phone://555”>` or disconnected with the tag `<disconnect/>`.

The VoiceXML code for the “Hello World” example would look like this:

```
<?xml version="1.0"?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
  <form>
    <block>
      <prompt>
        Hello world!
      </prompt>
    </block>
  </form>
</vxml>
```

More details about VoiceXML and especially about grammars are described further on in chapter 2.1.3 that explains XHTML+Voice, a markup language that uses a subset of VoiceXML for the voice part of a multimodal application.

2.1.2. SALT

Speech Application Language Tags (SALT) extend HTML pages to add a speech and telephony interface to Web applications and services. SALT contains a set of tags for specifying voice interaction. SALT was developed by the SALT Forum as a competitor to VoiceXML. The SALT Forum was founded 2001 by Microsoft and other companies. 2002 the SALT 1.0 specification was submitted to W3C for review. But VoiceXML prevailed and Microsoft decided 2006 to join the VoiceXML Forum to stay competitive.

Whereas the Microsoft Speech Server 2004 supports only SALT, Microsoft Speech Server 2007 additionally supports VoiceXML. Today nearly every other company had committed to VoiceXML and the SALT Forum is not active anymore.

The great advantage over VoiceXML is that SALT can also be used to speech-enable Web applications for multimodal clients while VoiceXML is limited to voice-only interactions. For multimodal applications a visual page like an HTML form can be extended with SALT to support speech input and output. The recognition may be started by clicking on a button, which activates a grammar relevant to an input field and the recognition result will be bound to that input field. Voice-only applications like telephony applications do without a graphical display, the user of such an application has access to information via a phone, and he calls a certain number with his phone or mobile phone to access data by using voice commands. SALT can be used like VoiceXML for applications that are accessible by telephone, in such applications SALT is used to describe the rules of the dialog-flow. Such speech processing applications are important to call centres, to make them more efficient, to forward calls to corresponding scopes or to gain particular information. So the two major scenarios for the use of SALT are multimodal and voice-only applications.

The most important elements of the SALT markup language are:

- `<prompt>` for speech output, plays audio recordings or synthesized speech from the text-to-speech engine
- `<listen>` for speech input, it executes and handles the speech recognition, a distinction is drawn between three listen modes: automatic, single and multiple
- `<grammar>` for specifying words and phrases the user might say
- `<record>` for capturing spoken speech or other audio
- `<bind>` for integrating recognized words and phrases with the application logic
- `<dtmf>` for telephony applications to recognize telephone touch-tones
- `<smex>` for communications between the speech platform components

Microsoft provides an extension to SALT, the `<audiometer>` element, which shows a visual cue for the speech recognition proceeding.

SALT dialogs can be used with both voice-only and multimodal browsers, thus allowing telephony and multimodal applications. But SALT requires a host environment, it uses data models and execution environments of its containing environment, these are HTML forms and scripting. A simple “Hello World” example would look like the following:

```
<html xmlns:salt="http://www.saltforum.org/2002/SALT">
   <head>
      <title>Hello World</title>
   </head>
   <body onload="hello.Start()">
      <salt:prompt id="hello">Hello World</salt:prompt>
   </body>
</html>
```

The architecture of a SALT implementation (Figure 2) consists of four components: a Web server, a telephony server, a speech server and a client device. The Web server generates Web pages with HTML extended with SALT, for voice-only interaction an embedded script is managing the dialog flow. The telephony server has an incorporated voice browser which interprets the HTML, SALT and script and connects to the telephone network. The speech server does the speech recognition, plays audio prompts, and gives a response to the user. The client device, for example a Pocket PC, can interpret HTML and SALT with a special version of Internet Explorer respectively a plug-in for local and remote speech recognition. The speech add-in for Pocket Internet Explorer supports only remote speech recognition, for example on the Microsoft Speech Server [41].

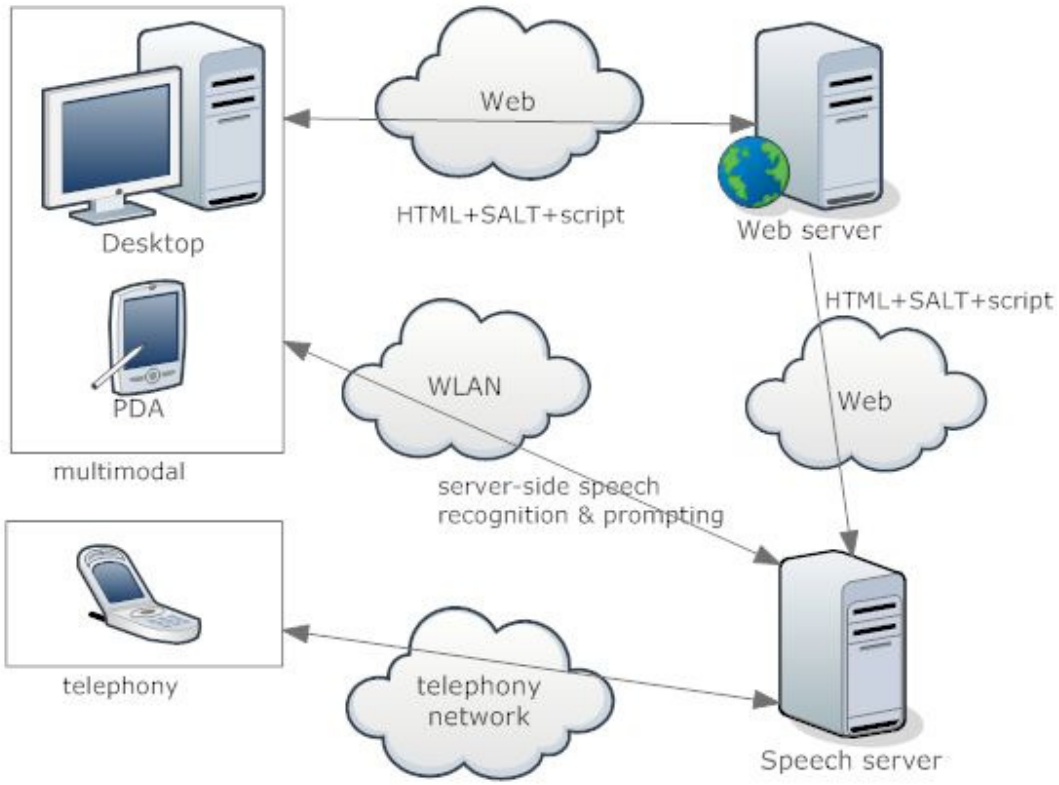

Figure 2: SALT architecture

The Microsoft Speech Server (MSS) contains the Telephony Application Services (TAS) and the Speech Engine Services (SES), as Figure 3 outlines. TAS comprises basically the SALT interpreter. SES includes speech recognition and text-to-speech engine which consists of a speech synthesis engine and a prompt engine. The prompt engine can use pre-recorded prompts and can furthermore even construct prompts by concatenating pre-recorded prompt-segments. SES converts spoken speech into text and text into human-sounding speech. The Telephony Interface Manager (TIM) is not included but resides on MSS. TIM interacts between MSS and the telephony card.

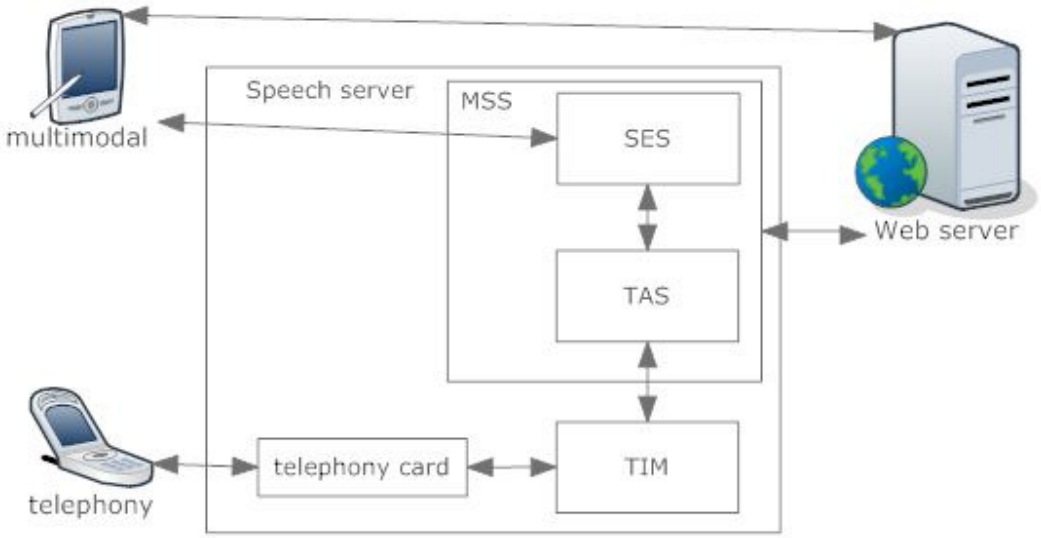

Figure 3: Speech Server

In a voice-only scenario the telephone call comes in through the telephony card, goes through TIM and then TAS, which informs the Web server and asks for the start page to begin the dialogue with the caller. Then the system asks questions which the user answers, the order for playing these prompts to the user is managed by an embedded script. Telephony or voice-only applications accessed by a conventional telephone or a mobile phone accept input in form of spoken words and phrases or of numerical digits by pressing the keypad of the phone, this is called DTMF (Dual Tone Multi-Frequency).

In a multimodal interaction with a PDA or smart phone the connection goes directly via Web to the Web server, which initiates a session with SES for server-side speech recognition and prompting. The recognition result and the prompts are transmitted over wireless WAN or LAN while the Web server delivers the pages. The local client on the mobile device handles the pages containing SALT. The SALT interpreter included in the speech add-in for Pocket Internet Explorer interprets SALT tags, HTML and the embedded script. If the multimodal interaction takes place with a desktop computer there is also a speech add-in for Internet Explorer, so that speech recognition and prompting can take place locally. But if wanted SES can be used again for server-side recognition. Both Internet Explorer Versions for PC and for Pocket PC require the installation of a speech add-in to enable SALT. Multimodal applications are as GUI-only applications often user-directed, so the user clicks or points to specify the kind of data he will enter and then enters the data while voice-only applications are system-directed, where the system guides the user through a dialogue of questions.

For the development of multimodal SALT applications Microsoft Visual Studio .NET and the Speech Application Software Development Kit (SASDK) can be used, described in

chapter 2.2.1. Multimodal and voice-only applications built with SASDK are deployed to the Web server.

2.1.3. XHTML+Voice

XHTML+Voice, short called X+V, is another markup language for voice-enabling Web pages. A developer creates a multimodal application by adding voice markup to XHTML [80] pages so that a user can interact with the application by voice as well as traditionally with text. The developer simply adds voice markup for each visual element of the user interface and specifies which voice snippet should be activated when. Whether the user interacts with the multimodal application using voice or keystrokes, the input is returned to the application, which recognizes the information automatically and handles it accordingly. There is no need for developing separate applications for each type of input. X+V uses standard XHTML for the visual part, a small and simple subset of VoiceXML for the voice markup and XML Events [83] to associate the VoiceXML snippets with specific visual elements of the Web page. So X+V is based on open standards.

XHTML, an application of XML, is used for the visual part. It is like HTML, but must be well-formed XML. An HTML page can be converted to XHTML by doing some simple structural changes. For example in XHTML all elements must be closed, in HTML there are some elements that are always empty and need no closing tag, e.g. `<br>`. In XHTML `<br />` must be used for a line break. In HTML it is allowed that some elements have no ending tag, like `<li>` or `<p>`, that is not valid in XHTML. XHTML is like XML case-sensitive whereas HTML is not. And there are a few other differences, see [80] for details.

The voice part of X+V is a subset of VoiceXML, primarily the `<form>` element and its children [81], which define a speech dialog:

`<form>` represents a voice handler which is activated in response to an HTML or VoiceXML event; it collects user input and presents information to the user using speech

`<initial>` can be used to prompt for form-wide information. It has prompts, catches, and event counters

`<field>` defines an input field in a form and the speech dialog between the user and the application. It has prompts, catches and event counters like the `<intial>` element and additionally grammars and `<filled>` action

`<record>`	records the spoken input
`<block>`	contains executable content, that is executed if the block's form item variable is undefined or if there is a `cond` attribute then it is executed only if that attribute evaluates to true

The elements `<catch>` and `<throw>` are used for catching and throwing events. For some main events there are own tags to catch them: `<error>` catches all error events, `<help>` catches the event thrown when the user says "help", if a timeout occurs while waiting for a user input a noinput event is thrown which can be caught with `<noinput>` and `<nomatch>` catches the event thrown if the user input does not match the active grammars.

Other VoiceXML elements supported in X+V are the elements for speech input: `<grammar>` defines a speech recognition grammar, `<option>` specifies alternatives for the user within the `<field>` element and `<lexicon>` contains a reference to an external pronunciation lexicon. For speech output and audio the elements `<audio>`, `<enumerate>`, `<prompt>`, `<reprompt>` and `<value>` can be used. Where `<audio>` plays an audio file or an audio variable and can define an alternate content if the audio sample is not available and `<prompt>` plays a recorded audio file and synthesized text to speech. A subdialog of the current dialog can be invoked with the element `<subdialog>`, this is a reusable dialog.

The element `<assign>` assigns a value of an expression to a variable of either the voice or the visual part. It can be used to update XHTML control values as `<input>`, `<button>` or `<select>` elements and JavaScript variables defined within XHTML `<script>` elements. With the VoiceXML element `<clear>` one or more variables or form items can be reset. Conditional logic can be implemented with the tags `<if>`, `<elseif>` and `<else>`. The element `<filled>` specifies an action which is executed after some input fields are filled. With `<log>` logging and debugging messages are generated and `<var>` declares a variable in the VoiceXML form. `<property>` sets a speech parameter for the VoiceXML form or an input item of the form that affects platform behaviour like timeout and confidence level of the speech recognition engine.

X+V additionally offers the tags `<sync>` and `<cancel>`. The `<sync>` element binds an input field of an XHTML form to a VoiceXML field, it synchronizes data entered via speech or text. With `<cancel>` a running speech dialog can be cancelled.

An important part of the voice interface is the grammar, because it influences the

accuracy of speech recognition. The developer must define all words and phrases that are accepted for a certain prompt, so that the speech recognition engine can detect and interpret the spoken input. The speech recognition engine compares the utterances of the active grammar with the incoming speech. A grammar can be just a simple list of words or it is more complex to recognize natural language with phrases and whole sentences. Grammars should not be too complex and should not contain too many words because size and complexity degrade the performance. Grammars can be created inline or in external files. It is not recommended to use inline grammars because they cannot be reused. A grammar consists of a set of rules defining all utterances that can be recognized. For writing grammars the Java Speech Grammar Format, short JSGF [28], can be used, a platform and vendor independent textual representation of grammars. The developer can specify semantic interpretation tags in the grammar which are used to translate speech recognition results into another format e.g. for reformatting dates and numbers.

The following sample code shows a JSGF grammar which recognizes “yes” and “no” or similar words users might use for meaning “yes” or “no”:

```
#JSGF V1.0 iso-8859-1;
grammar yes_no;

public <yes_no> = <yes> { $ = true; }
     | <no> { $ = false; };

<yes> = yes [please] | sure | okay | fine | yep | yup |
     affirmative;
<no> = no | nope | no thanks | negative;
```

A call to this grammar in VoiceXML will look like the following:

```
<vxml:grammar src="yes_no.jsgf" />
```

A multimodal application in X+V consists of a visual markup that specifies the visual interface and voice markup snippets for each component specifying the voice interface. XML Events are used to create the correlation between visual and voice elements, so that the browser knows which snippet of voice markup is related to which visual part and when to activate the voice snippet. The familiar event types from HTML like `on mouse-over` or `on input focus` are used to associate a visual element of the user interface like an input field with the corresponding voice markup. An event handler defines the action to be done when a special event, for example a mouse click, occurs. In X+V an

event handler enables the interaction between visual and voice markup. X+V supports the XML Events event types and additionally the following VoiceXML event types: `nomatch`, `noinput`, `error` and `help`. And the `vxmldone` event, which is generated when a voice handler completes is also supported by X+V. Instead of an XML event and event handler the X+V element `<sync>` can be used to bind visual and voice elements. Figure 4 shows an example for speech enabling a Web form with the three elements: visual markup, voice markup and the X+V markup for the correlation of visual and voice.

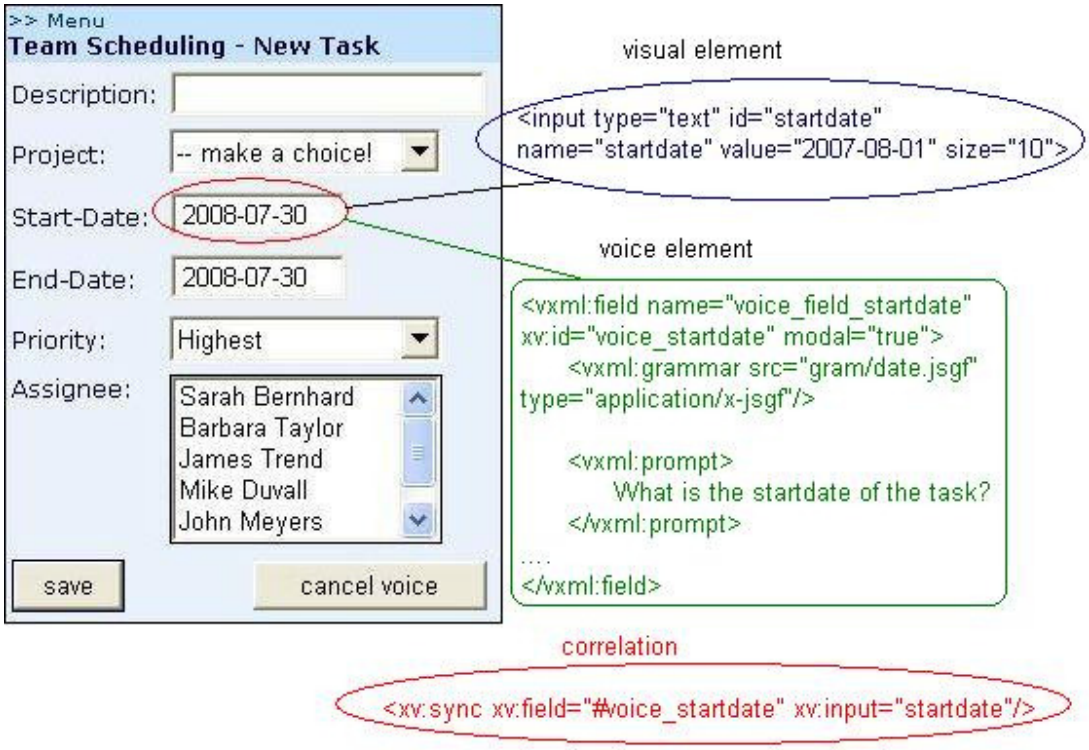

Figure 4: A speech enabled Web form with X+V code segments

The following X+V example is a “Hello World” example. It will output the words “Hello World” by voice if it is loaded in a voice enabled browser:

```
<!DOCTYPE html PUBLIC "-//W3C/DTD XHTML+Voice 1.2/EN"
"xhtml+voice.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
      xmlns:ev="http://www.w3.org/2001/xml-events">

<head>
  <title>Hello, World</title>
  <form xmlns="http://www.w3.org/2001/vxml" id="sayHello">
    <block>Hello World!</block>
  </form>
</head>

<body ev:event="load" ev:handler="#sayHello">
  <h1>"Hello World!"</h1>
</body>
</html>
```

A simple type of speech interaction is the “directed dialog”. This means that each input field of a Web form is speech enabled so that if the user moves from one field to another he gets a voice prompt as well as a visual one. While the “mixed initiative” interaction is more powerful and allows a more natural voice input, for example the user says the input in a whole sentence to fill in all fields at once. X+V also supports mixed initiative interaction.

X+V applications can be developed using the Multimodal Toolkit for IBM’s WebSphere Studio [27], presented in chapter 2.2.2. But for authoring X+V pages there is not really the need for a specific tool, any HTML or XML editor can be used. The voice capability, the speech recognition engine and speech synthesizer, is included in the multimodal browser. The Multimodal Tools package includes two multimodal browsers, one based on the Opera Browser V 7.55 [47] and the other is based on NetFront Browser V 3.1 by Access Systems [45]. Both are enhanced with IBM speech recognition and synthesis technology and can be used to interact with multimodal applications written in X+V. Both browsers have three listening modes using the Push-to-talk or microphone button in a certain manner:

- Push-to-talk or Hold key while talking:
  the user presses and holds the Push-to-talk or microphone button on the device while he speaks and releases it afterwards
- Push-to-activate or Press key, then talk:
  the user presses and releases the button and then he talks. After he has finished talking the system detects silence and automatically stops the listen mode. But when using the `<record>` tag, the user must press and release the button, begin to speak and to signal the end he pushes and releases the button again.
- Auto-push-to-activate or Key not required to talk:
  the browser sounds a tone to indicate that it is ready to record and after the user has finished speaking it detects silence and automatically stops listening

The listening mode and the Push-to-talk or microphone button can be changed in the Voice Preferences of the multimodal browser.

The great advantage over SALT is that X+V uses standard markup languages as XHTML and VoiceXML, so developers need not to learn a whole new language. And because all these parts of X+V are XML-compliant the voice markup can be separated from XHTML by packing each in one file. This makes the development more flexible, voice dialogs can be built by speech interface developers and XHTML can be built by traditional Web design professionals. Another advantage of separating voice and visual markup into two files is that VoiceXML dialogs can be reused in many other XHTML pages and even in other containers than XHTML for example in a VoiceXML document for building a voice-only application. With X+V a single application can be created which supports multimodal browsers as well as GUI-only browsers and voice-only systems.

2.1.4. EMMA

EMMA, the Extensible MultiModal Annotation markup language, an XML markup language, is in contrast to SALT and X+V not authored by developers but generated by the components of a multimodal system. Interpretation components such as speech recognizers, handwriting recognizers and other input media interpreters are generating EMMA automatically to provide semantic interpretations of user input taken in any modality for example voice, keyboard, ink, GUI and others, see Figure 5. EMMA serves as a data exchange format between input processors and the interaction manager of a multimodal application [76]. Where the interaction manager coordinates and manages multiple input and output modality components. One of the advantages of a multimodal application is that the user can use one modality in combination with another for giving input, for example he points to a position on a displayed map and asks for further

information for that point. Also the multimodal application may use two modalities in combination for giving output, for example playing spoken prompts and showing information on the display. Each modality component uses the EMMA notation to express the user input and then the fusion module combines them into a single EMMA document representing the user command and its parameters. But EMMA not only provides the semantic interpretation for an input modality component but also annotations on the input interpretation such as confidence scores, timestamps and input medium. These various attributes assist the multimodal fusion engine to combine the inputs into one single interpretation of the user input for the interaction manager.

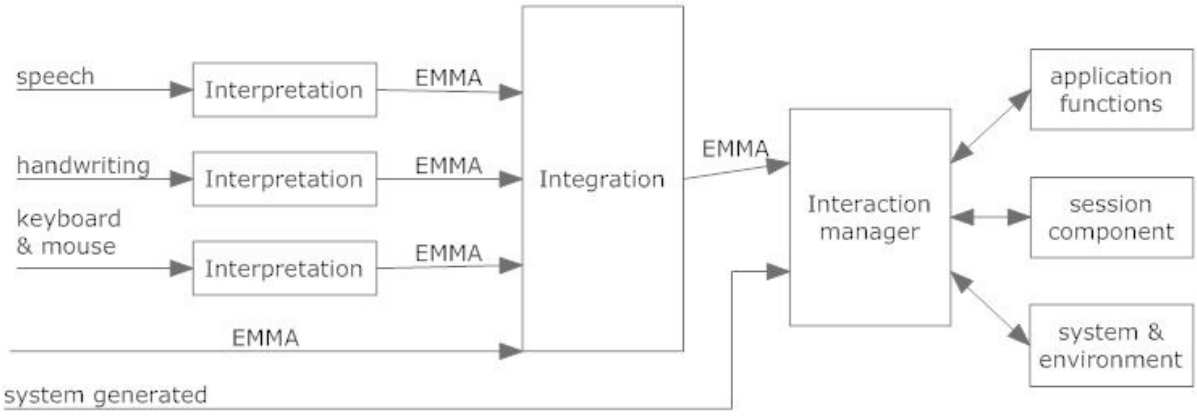

Figure 5: Input components of a multimodal system generating EMMA

The main components of an EMMA document are the instance data contained within an `<emma:interpretation>` element, the data model which is optionally specified as an annotation of that instance and the metadata annotations that may be applied at different levels of an EMMA document. The `<emma:interpretation>` element represents the user input interpreted by an input processor such as a speech recognizer and contains application specific markup. An EMMA document may contain multiple interpretations of a single input.

The root element of all EMMA documents is `<emma:emma>`, containing attributes like EMMA version, namespace and schema declaration. The core of an EMMA document consists of the container elements `<emma:group>`, `<emma:one-of>`, and `<emma:sequence>` and a number of `<emma:interpretation>` elements containing the semantic interpretation of the user input. Each container element is again a container for one or more interpretation elements or container elements. The element `<emma:one-of>` is used as a container for the N-best interpretations of the user input where each interpretation is contained within an `<emma:interpretation>` element. The element `<emma:group>` is used for grouping user inputs and the element

<emma:sequence> for the representation of sequences of inputs.

An EMMA document consists of the root node, a tree of container elements and a number of interpretation elements. The interpretation elements are representing the interpretation of the user input or containing a <emma:lattice> or a <emma:literal> element. The <emma:lattice> element represents lattices, a compact representation of possible recognition results like speech, gesture, handwriting or other interpretations of multimodal inputs. And the <emma:literal> element is for semantic results in the form of string literals without any application-specific markup. An <emma:interpretation> element must contain a single interpretation of the user input represented in application specific markup or a single <emma:lattice> element or a single <emma:literal> element or it must be empty. An EMMA document consist of at least the root node, a <emma:interpretation> element, a <emma:one-of> element, a <emma:literal> element and the EMMA attributes emma:no-input, emma:uninterpreted, emma:medium and emma:mode. All other elements and attributes are optional.

The EMMA annotations are a series of attributes and elements which are used to provide metadata associated with the user input. EMMA annotation elements can appear more than once within an element and can have an internal structure whereas EMMA annotation attributes are represented as attributes and occur on <emma:interpretation> elements and some can occur on container elements, <emma:lattice> elements and elements in the application-specific markup. Annotations of the element <emma:one-of> apply to all contained <emma:interpretation> elements. The attributes emma:medium and emma:mode must be set for all EMMA interpretations either directly on the <emma:interpretation> element or on an ancestor <emma:one-of> element or on an earlier stage of the derivation list in the element <emma:derivation>. The attributes emma:medium and emma:mode provide a classification of the input modality, indicating the input medium and the mode of communication used on that medium, for example the value of modality may be voice and the medium acoustic. With the annotation attribute emma:function other uses than interactive dialog for example recording can be outlined and to distinguish verbal mode from non-verbal the attribute emma:verbal can be used. The attribute emma:uninterpreted="true" indicates that no interpretation for the input was produced, then the <emma:interpretation> element must be empty. Also the interpretation is empty if the attribute emma:no-input is set to true. The previous stage of processing of an interpretation and the fusion of multimodal inputs may be represented with the annotation elements <emma:derived-

from> and <emma:derivation>. The attributes emma:start and emma:end are for timestamps representing start and end of the user input signal and emma:signal is an URI indicating the location of the input signal, for example an audio file. Then the attribute emma:signal-size contains the size of that file. The attribute emma:source provides a description of the device that captured the input. A description of the processing stage which resulted in the current interpretation is emma:process which has the value asr if the process is speech recognition and it then specifies the speech recognizer version. The attribute emma:lang provides the language spoken, emma:media-type indicates the MIME type of the signal and contains a location for specifying codec and sampling rate. The attribute emma:grammar-ref provides the grammar resulted in that interpretation, the grammar used in the processing is specified with the element <emma:grammar> under the root element <emma:emma>. So multiple grammars are allowed to be specified and referenced for each interpretation. Likewise multiple data models can be specified and referenced with the emma:model-ref attribute for each interpretation and the element <emma:model> contains the inline specification of the data model of the semantic representation. emma:tokens and emma:confidence are attributes of the <emma:interpretation> element containing particular strings of recognized words and a confidence score between 0 and 1. A container for application and vendor specific annotations is the <emma:info> element.

The following code is an example of an EMMA document representing input to a flight reservation application. The document represents two semantic interpretations of the user input, where the speech recognizer is uncertain about what the user meant, the annotations show the confidence scores and timestamp for the input.

```
<emma:emma version="1.0"
   xmlns:emma="http://www.w3.org/2003/04/emma"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.w3.org/2003/04/emma
   http://www.w3.org/TR/2009/REC-emma-20090210/emma.xsd"
   xmlns="http://www.example.com/example">
  <emma:grammar id="gram1" ref="grammarURI"/>
  <emma:model id="model1" ref="datamodelURI"/>
  <emma:one-of id="r1"
     emma:medium="acoustic" emma:mode="voice"
     emma:function="dialog" emma:verbal="true"
     emma:start="1282990164000" emma:end="1282990167000"
     emma:grammar-ref="gram1" emma:model-ref="model1">
```

```
    <emma:interpretation id="int1" emma:confidence="0.75"
       emma:tokens="from boston to denver">
      <flight>
        <origin>Boston</origin>
        <destination>Denver</destination>
      </flight>
    </emma:interpretation>
    <emma:interpretation id="int2" emma:confidence="0.68"
       emma:tokens="from austin to denver">
      <flight>
        <origin>Austin</origin>
        <destination>Denver</destination>
      </flight>
    </emma:interpretation>
  </emma:one-of>
</emma:emma>
```

2.2. Development Tools

In this section we discuss different development tools for multimodal and voice-only applications using the languages SALT or X+V for multimodal development and VoiceXML for creating voice applications. It's not necessary to use such a development environment but it facilitates creating, editing, testing and deployment of multimodal and voice applications.

2.2.1. Microsoft Speech Application SDK

Microsoft, one of the founders of SALT, has built a Speech Application Software Development Kit (SASDK) [40], integrated in the Visual Studio .NET development environment. This SDK contains a set of development tools for building and testing speech-enabled ASP .NET Web applications that use SALT. The SASDK includes a speech add-in for Internet Explorer and Pocket Internet Explorer that makes IE speech capable. It interprets SALT tags on Web pages, but for speech-enabling a Web application on a Pocket PC the MS Speech Server is needed for speech recognition and remote speech output, while speech-enabled applications running on desktop computers can use local installed speech engines.

The SASDK is seamlessly integrated into Visual Studio .Net, it contains authoring and debugging tools, the ASP .NET speech controls, a graphical grammar editor, a prompt database and management tools, call inspection, log analysis and reporting tools, testing and debugging tools, a telephony application simulator, reference applications and sample code. With the Speech Controls Editor, integrated with the Web Form Editor, speech controls can be placed on Web pages graphically by drag-and-drop just like other ASP.NET server-side Web controls, but with additional properties that enable speech. Apart from that they work like other Web controls of .NET.

In the SASDK there is a template for creating speech applications included, the "Speech Web Template". This template sets default application settings and the default application mode is voice-only, but all can be changed. Additionally it creates a default grammar library file and if he wants the developer can create a new prompt project. While creating a voice-only project a default.aspx Web page will be built, containing two basic speech controls and a prompt project and a grammars folder is built by default. While creating a multimodal project there are no controls on the default.aspx Web page and no prompt project will be created by default.

The developer uses the Speech Control Editor to specify prompts to speak to the user,

to recognize answers using grammars and to confirm the answers. The developer does not need to do textually SALT programming, Visual Studio .NET creates the corresponding HTML and SALT tags. The SASDK includes two types of speech controls, basic dialog controls and application speech controls. Especially in a voice-only application the speech controls are used for controlling the dialog-flow between user and application. The most commonly used dialog control is the QA control, which contains a prompt and a response, a question and an answer. Contrary to voice-only applications a multimodal application uses visible controls instead of prompts to initiate dialogs with the user and there is a need of one or more prompts to provide descriptive information to the user about the handling of the application, which phrase can be spoken and what to do in which order and so on. An extension of the basic dialog controls are the speech application controls, which are for building typical user interactions. For example, the YesNo control is used to collect a yes or no answer. And the developer can create custom controls; a sample solution file is installed with the SASDK.

In order for the speech recognition engine to return recognition from audio input, a grammar must be added to each Speech Control. Speech applications developed with SASDK are not open to any spoken text, they are restricted to certain phrases. The speech engine utilizes grammar rules to interpret what the user is saying. The developer defines them in multiple grammar files and associates them with a particular user interaction. The application developer must try to anticipate every spoken phrase the user may utter. This requires more work but has the great advantage that there is no need for a training process to interpret the users' speech. The speech engine returns the recognized text in the Semantic Markup Language (SML) [66]. The Grammar Editor is the tool for creation and graphical representation of speech grammars; it exports the SRGS (Speech Recognition Grammar Specification) [68] grammar format. The grammar file is an XML file, and when saved using Speech Grammar Editor, has a .grxml extension. The Speech Grammar Editor displays a graphical representation of the relationships within the grammar. The grammar can be compiled into a context-free grammar (.cfg) using the grammar compiler (SrGSGc.exe) provided. Compiled grammar files increase the speed of the application, they have reduced file size and loading into memory is done faster. Common grammar rules for handling numbers, dates, credit card information and rules for yes/no responses are defined in the library.grxml file which is included in each project by default. If the data content is likely to change then it is useful to build dynamic grammars where the creation of the grammar files take place during the application execution time.

Another important tool is the prompt editor for specifying prompts and recording wave files and associating them with a prompt. A prompt is an utterance, a single phrase or a whole sentence that is used to speak to the user of the application. It is recommended to

use recorded prompts because they are more natural than the text-to-speech engine. A prompt project consists of one or more prompt databases, each prompt database contains all potential prompts needed to communicate with the user. Resources like grammar files and prompt projects will be preloaded and cached by the Speech Server to improve performance.

The speech debugging console, working in conjunction with the Visual Studio debugger, has a simple GUI interface, which shows client and server-side SALT. A developer can view and edit the speech recognition results, they are shown in SML files, and simulate errors and exceptions and he can use speech and text emulation while interacting with the application. While debugging a voice-only application the Telephony Application Simulator (TASim) can be used to simulate the client experience and to test telephony applications, for example for giving DTMF input. With SASDK the speech application log analysis tools can be installed which can be used to analyse and extract important data from the trace log files created on the Speech Server.

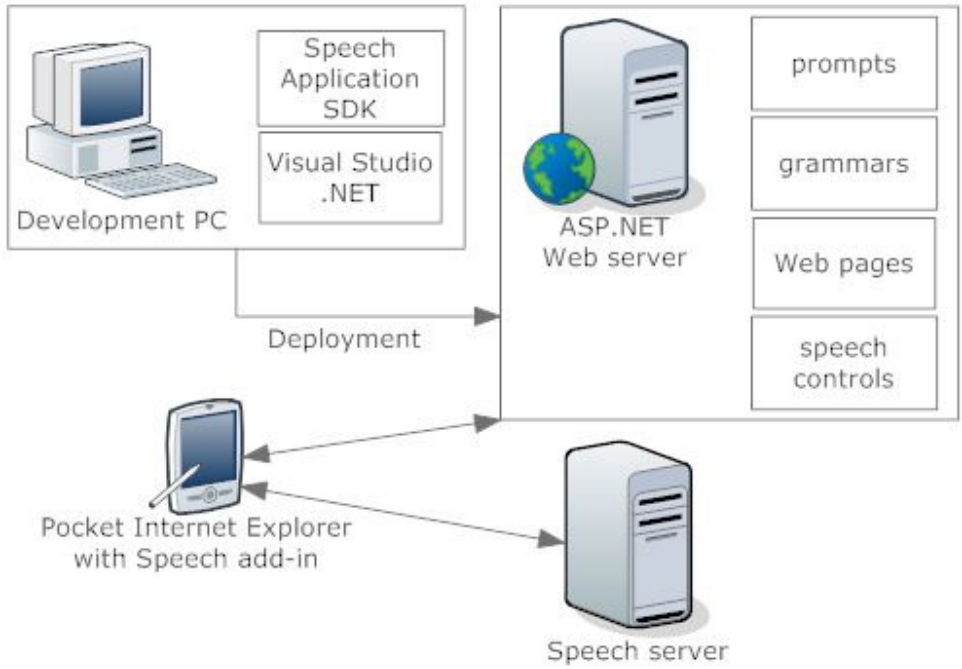

Figure 6: Deployment of SALT applications using Visual Studio with SASDK

A speech application can be manually deployed following the same steps that are used to deploy any other ASP.NET Web application. The .NET deployment model involves replicating an application's entire directory structure and dropping it at the target, so the application directory and subdirectories can easily be zipped up and then unzipped on the target. This should work as well as creating a Web Setup Project. The SASDK provides a Project Wizard to create a self-installing .msi for a speech application. This

makes deploying the application to a Web server simple and convenient (Figure 6).

A detailed description and introduction in developing speech applications using SASDK can be found in [56].

2.2.2. IBM WebSphere Multimodal Toolkit

The Multimodal Toolkit [26], an extension to IBM's WebSphere Studio framework, is an integrated development environment with multiple tools, editors and views to create, test and run multimodal applications written in XHTML+Voice. Additionally it includes reusable dialog components for common functions like credit card, date and numbers. The Multimodal Toolkit contains two multimodal browsers based on Opera Browser and ACCESS Systems NetFront Browser that are both enhanced with extensions including the IBM speech recognition and text-to-speech technology, allowing to view and interact with multimodal applications that have been built using X+V, see Figure 7.

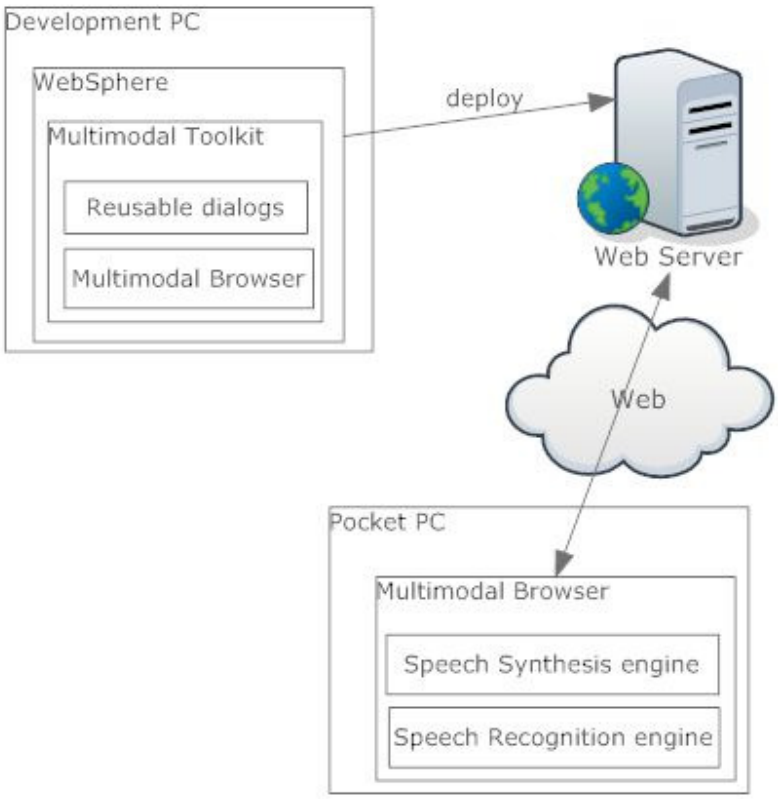

Figure 7: The Multimodal Toolkit for developing X+V applications

With the Multimodal Toolkit X+V applications are created with a multimodal project. The main part of a multimodal project is one or more X+V files with the file extension .mxml.

An X+V file can consist of parts, the visual and the voice code of the application. Figure 8 shows the basic steps to create a multimodal application with X+V. When using the development kit for creating an X+V file a basically prefilled .mxml file including the X+V DTD and the definitions for the namespaces is opened in the X+V editor. In the next step the developer adds the visual component by using an existing HTML or XHTML file or by writing the code directly. A useful feature of the editor is Content Assist showing all valid tags for a current position to help the developer inserting correct code. After coding the visual part the XHTML file should be validated to ensure that it is XML-compliant.

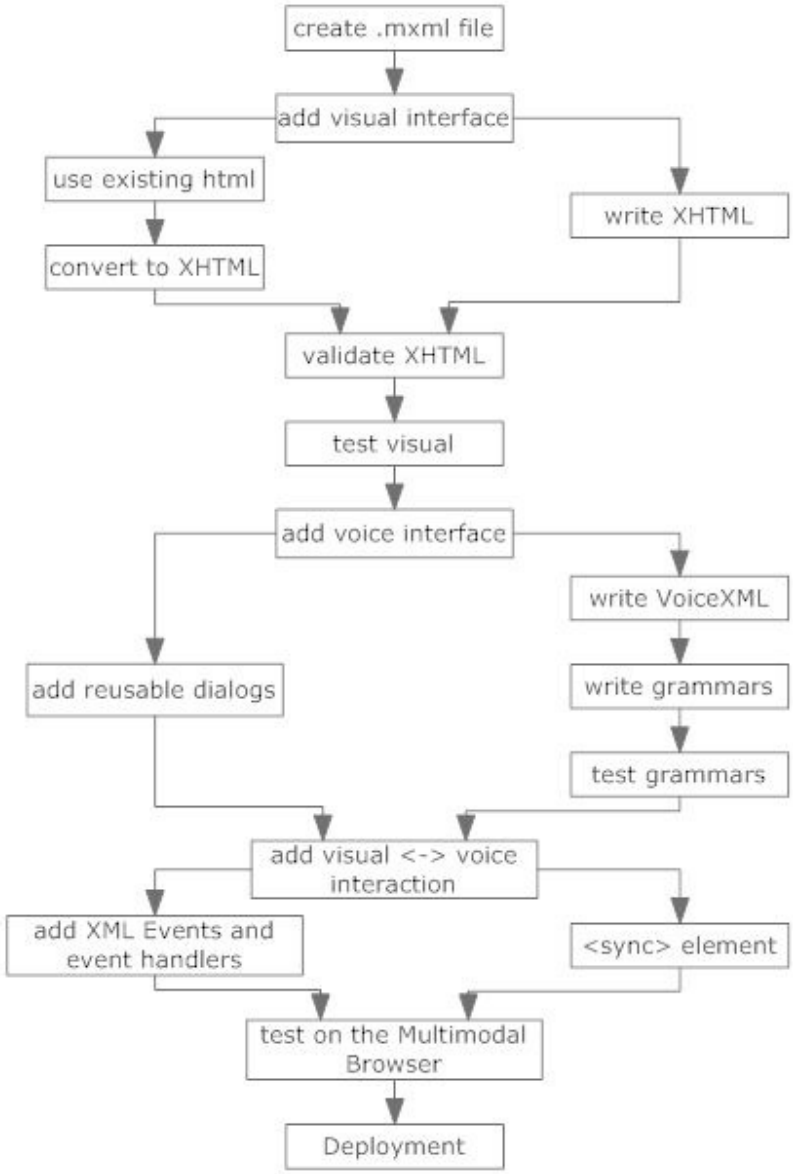

Figure 8: Basic steps to create an X+V application

In the next step, a voice portion for each visual field that should be voice-enabled is

added. The Multimodal Toolkit contains many reusable dialog components for common functions which either can be added easily or the developer can write his own grammar and VoiceXML code. The voice part is commonly added in the `<head>` element to separate it clearly from the visual part. Another feature of the Multimodal Toolkit is the Reusable Dialog Component Wizard that helps integrating an existing subdialog. The developer can code VoiceXML from scratch, consisting of a VoiceXML `<form>` element containing other VoiceXML elements like `<grammar>` which defines the grammar to be used and `<prompt>` which contains the text for the speech synthesizer or a reference to an audio file for prompting the user. The Multimodal Toolkit includes a SRGS grammar editor to write, check, compile, validate, convert and test grammars. Grammar code generated with the editor is stored in an external file with the extension .grxml although inline grammars will work, but are not recommended. The speech recognition engine provides default pronunciations for words included in grammars, but the developer can customize pronunciations for his grammars by creating pronunciation pool files with the Pronunciation Builder tool. A helpful feature for testing the grammars is the grammar test tool of the development kit.

After coding the visual and the voice part of the multimodal application the developer has to add the interface between the two parts. Therefore he specifies an event type and an event handler in the XHTML element where he wants to access a voice part. The event type specifies the event when the VoiceXML will be executed and the event handler binds the event to the VoiceXML element. An alternative to event and event handler is the X+V element `<sync>` which can be used to synchronize visual and voice element. After developing a multimodal application with the Multimodal Toolkit it can be tested using the Multimodal Browser which can be launched in the development environment.

2.2.3. Eclipse Voice Tools

The development platform Eclipse [18] is widely used as a Java development environment but additionally the Eclipse Foundation hosts various projects for creating different applications using diverse languages. One of these projects is the Voice Tools Project [17], a development environment for voice applications based on the Eclipse Platform, so that developers are able to use the same tools as for developing visual applications. The OpenVXML Designer is the primary software package, the graphical development environment of the Voice Tools Project that is based on a drag and drop interface.

The main components within the OpenVXML Designer are the Voice and the Application, so the logic is separated from the presentation. All audio and grammar files that are necessary for a voice application are included in a voice project. With the OpenVXML Designer grammar files in the common format GRXML are created easily. Pre-recorded audio files are organized in subfolders of the Media Files folder of the Voice Project so that OpenVXML can use the pre-recorded snippets for playing dynamic data automatically, for example dates and phone numbers. The VoiceFormatter associated with the voice project uses audio files or text-to-speech (TTS) to render dynamic values. Shared content like predefined prompts that can be invoked at any point in the application can be modified in one place and updated throughout the system because they are stored in the Voice.xml file which can easily be edited in the Voice Project Editor. The Logic of the voice application is handled by an application project, call flow and configuration information are organized into a simple structure. Primitive objects like the PlayPrompt module and complex structures like dialogs can be used to build the voice application in the application editor. Built-in components like databases, business objects and Web services can be used for data access. A voice project is created for a specific language, but the application project can be multilingual if an additional Voice for another language is added.

After having created the voice application with OpenVXML Designer it is exported and uploaded as a WAR file to a Web server running a Java servlet container like Apache Tomcat. The Eclipse Voice Tools Project can interact with any VoiceXML 2.1 compliant platform. The voice browser requests VoiceXML from the Web server when a caller dials into the system. The requested VoiceXML contains the dialog flow, the audio files and TTS prompts to be played to the caller, what is expected as input from the caller and other actions depending on the caller's response. Figure 9 shows these basic components of a voice application using OpenVXML.

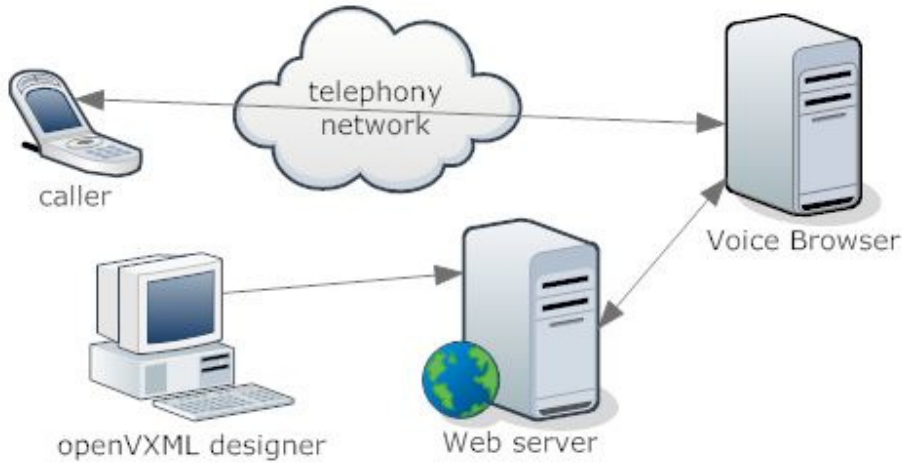

Figure 9: An OpenVXML voice application

In the past there was the Multimodal Tools Project for Eclipse [43] that enhanced the Eclipse Voice Tools with a few helpful plug-ins for developing and testing XHTML+Voice applications. It contained an editor for X+V with syntax-highlighting that facilitates coding and could validate XHTML. Additionally the package included voice-enabled Multimodal Browsers for testing the developed multimodal applications.

2.3. Speech Recognition and Speech Synthesis Technologies

Speech is the most natural and easiest way for humans to communicate and interact. Talking to a computer is not science fiction anymore, many advances have been made in the field of speech technologies in the past years. Speech recognition and speech synthesis are the key technologies in the area of speech technologies. Speech recognition engines have considerably improved, from speaker dependent and relying on a time-spending training phase or limited vocabulary to larger grammar support and enhanced with noise filtering technologies. Synthesized speech has changed from robotic and unnatural sounding towards almost undistinguishable from recorded human voice.

2.3.1. Speech Recognition

Speech recognition engines are used to get a semantic interpretation [65] for speech recognition results. A semantic result is a computer processable representation of a natural language utterance. Grammars implement the semantic interpretation to associate spoken words and phrases with meaningful application data. A speech recognition engine understands all words and phrases that are defined in one or more grammars. Grammars can be defined inline or in an external file. The preferred markup language for grammars is SRGS, the Speech Recognition Grammar Specification.

Grammars should be created in consideration of accuracy and efficiency of the speech recognition. Accuracy is composed of the engine and the application accuracy. The engine accuracy quantifies how often the speech recognizer returns the correct phrase. The application accuracy expresses how often the application provides the intended result. The measured application accuracy can be better than the engine accuracy because even though the speech recognition engine was not accurate, the application can handle the utterances correctly. The application accuracy can be increased by adding phrases that users prefer to say. These phrases can be determined with usability tests. But the engine accuracy decreases the more phrases are enabled, because they can be confused with each other.

The grammar efficiency can be measured with two values: recognition time and compile time. The simpler the grammar, the shorter the recognition time and the compile time. Furthermore developers can use pre-compiled grammars to shrink compile time. This avoids compilation but the grammar efficiency now depends on the network connection and the load time of the compiled grammar. Also pronunciations are generated at compile time, so if the grammar has unusual words an application-specific pronunciation

dictionary can be used to fasten compile time and moreover to improve the engine accuracy.

A semantic interpretation can also be used to convert the user input. Grammars can return such as a symbolic representation of the recognized words or phrases. So that the user is allowed to speak naturally, but the application gets the form needed, for example the zip code of a country, the user says “Austria” and the grammar returns “A”.

Accuracy and efficiency of the speech recognition can be improved by setting adequate speech recognizer properties. Speech recognizer properties like confidence level, sensitivity and speedvsaccuracy can be set. The result of the speech recognition engine has a confidence value that indicates how certain the engine is of that result. With the confidence level property results below that certain level are rejected. Specify a low sensitivity property to avoid interpreting background noise as valid input. Speech recognition engines can operate in a mode optimized for accuracy, but then speech is reduced and vice versa. A higher speechvsaccuracy value means greater accuracy and slower speed whereas lower values mean faster speed but lower accuracy. The timeout property sets when the speech engine considers that the user has finished talking.

2.3.2. Speech Synthesis

Speech synthesis engines are used to generate speech output from text input. Speech output can be pre-recorded audio or output from a TTS engine. Recorded audio files sound more natural and require less memory and CPU time but if the prompts change new recordings are needed. There are several speech synthesis technologies with different advantages and disadvantages, the primary are: formant and concatenative speech synthesis [26]. A formant synthesis engine operates algorithmically and generates speech output from scratch. Whereas a concatenative synthesizer concatenates segments of pre-recorded speech to produce speech output. The segments which are assembled into speech output can be entire phrases or even smaller than syllables. The advantages of formant TTS are that it does not require a lot of memory or disk space whereas concatenative TTS requires more memory and disk space. Formant and concatenative synthesis engines operating with syllables or smaller segments have an unlimited vocabulary. That is an advantage over concatenative speech synthesis engines who work just with a fixed set of phrases, where new phrases must be pre-recorded. But the great advantage of such TTS engines is that their speech output sounds so natural that it is possibly not distinguishable from pre-recorded audio. Speech output from formant TTS does not sound very natural.

Mostly the characteristics of the synthesized voice like gender, age, speed and volume can be configured. But recorded prompts should be used whenever possible because of natural pronunciation. One of the most important qualities of TTS is naturalness, the synthesized speech should sound like human speech. Nevertheless naturalness is not always the goal of a synthesizer, sometimes high-speed synthesis and smaller memory and disk space is more important, for example in mobile devices a formant TTS is maybe more advantageous.

The Speech Synthesis Markup Language (SSML) [69] is an XML-based markup language for documents provided as input to a speech synthesizer. With SSML different aspects of the generated voice output can be defined such as pronunciation, volume, speed, pitch, rate and emphasis and voice characteristics like name, gender and age can be selected.

3. Related Work

Within this thesis we evolved a mobile context-aware multimodal application for a PDA. Our implementation takes context information to improve speech input and output as well as text input and output. In terms of multimodal the user of the application can decide if he will use pen or voice or both for giving input. And the application takes the context information of the device to decide which the best output modality is in the current situation of the user or the device.

In the literature exists some related work dealing with speech or multimodal technologies and several are combining context-awareness information within speech and multimodal applications respectively. Much research takes place in the medical domain, but also other domains like personal information management functions and mobile guides concern with multimodal interaction. Especially mobile applications for PDAs, smart phones, tablet PCs benefit from allowing multimodal interaction, because they are not usually equipped with common hardware input devices like mouse and keyboard and mostly have just small displays. So using other input and output modes like voice, gesture or handwriting increases the usability of mobile devices and even makes their usage possible under certain circumstances like driving a car or for people with disabilities.

3.1. Voice-only Applications

A great help for people with disabilities, especially for blind and visually impaired users who have troubles with reading, is a speech enabled Web interface. A voice or multimodal interface facilitates or even makes possible that blind and visually impaired users can interact with Web applications. [30] investigates how people with visual and motor disabilities use mobile devices. A result of the study is that even though these people have accessibility problems with common mobile devices, they favour commodity devices over specialized devices which are specially designed for people with disabilities. Improving the accessibility of common mobile devices with additional input modalities like voice will be a great advantage for people with disabilities too. eVALUES (e-library Voice Application for European Blind, Elderly and Sight-impaired) [21] provides blind and visually impaired users downloading of online-books and documents and listening to it. This Web service uses advanced text-to-speech technology and works not only with PCs but also with PDAs.

Another great advantage for visually impaired persons are interactive voice browsers, which facilitate using internet applications, which is relevant also for car drivers and mobile workers who need their hands free. [23] describes Vox Portal, a scaleable VoxML client, which supports ubiquitous voice-driven access to multiple information services for a range of devices. VoxML is a markup language that specifies the dialogs of voice applications and it features speech synthesis and recognition technologies. Vox Portal uses a dynamic converter for translating HTML to VoxML and vice versa to enable interactive input of data and to submit form data to the Web server. Much of the context of an HTML document is derived implicitly through the document structure and the layout of the content; users of a graphical interface get orientation and navigation out of the structure of a document. So translating a visual document to a voice browser means not only converting the content but also getting out the structure of the document. The HTML-VoxML converter uses an analytical algorithm to elicit structure and context out of an HTML document and generates VoxML output which is forwarded to the Vox Portal. To make structural elements explicit to the listener the audio rendering combines the use of descriptions, earcons, multiple voices, prosody, announcements and pausing.

Impromptu [62] is an audio-only platform supporting multiple voice services, for example news, radio, personal audio to-do lists, telephone and chat. It uses speech recognition for user input and speech synthesis to play sounds or text. While only one application can be active at a time, others will run asynchronously in the background and can give an alert sound so that the user takes attention and can activate the application if desired, for example an incoming phone call which the user is awaiting for.

3.2. Multimodal Applications

The combination of speech and pen use for input, the so called tap and talk interaction, makes speech processing easier [9]. It reduces speech recognition and understanding errors because the selection of a form field indicates what the user wants to say, for example a name or a zip code while filling in an address form. As well as tap and talk offers significant advantages over the pen-only interface. MiPad, short for Multimodal Intelligent Personal Assistant Device, is a wireless mobile PDA prototype embedding several keysubareas of spoken language technology to enable users to accomplish many common tasks using a multimodal interface with speech, pen and display. It is developed to provide PIM (Personal Information Management) functions like email, calendar, notes, tasks and contact lists. MiPad has a tap and talk interface which combines speech and pen input. Also in Search Vox [49], an automated directory assistance application, the user can use speech and text input in combination. With this application the user can request telephone and address information. If the speech recognition fails to interpret the speech result correctly, it presents an nbest list where all possible recognized phrases are displayed as words. So the user can not only select an entire phrase but also select words of the list to compose a phrase to refine his query and receive a correct speech recognition result. And on the other hand a user can type text hints like an initial letter to help the speech recognizer identify the query better. Additionally the user has the possibility to use verbal wildcards, he can use the word “something” as a placeholder if he is unsure about a word or phrase in his query.

In contrast to directed dialogs where the system always controls the dialog, mixed-initiative dialogs allow both human and computer to influence the dialog flow. ISIS [39], which stands for Intelligent Speech for Information Systems, is a spoken dialog system providing real-time stock market information, managing simulated personal portfolios and handling simulated financial transactions via mixed-initiative conversational interactions. Additionally the system is augmented with a display so that the user can also input textual queries by typing or writing with a stylus. The user can delegate tasks like the monitoring of a financial feed to a software agent which is working in the background until the defined alert conditions are met. Alerts are stored in an offline queue and to minimize disruption ISIS informs the user with visual icons about the arrival of new alerts. Whereas InCA [29] uses a conversational agent, its features are spoken natural language input and speech output with facial animation. It provides making and listing appointments, email reading, weather reports, exchange rates and news headlines. The interesting part of it is the facial expressions of the agent, for example the agent raises its eye brows and looks up before the system is ready to speak and while it is listening to the user.

A multimodal system for home-healthcare nurses is Care View [34]. It visualizes historic trajectories and identifies trends in patient's conditions. A nurse captures a number of patient data, vital signs like blood pressure, but to make a decision about the appropriate action she has to classify the current data and analyze it in the context of the patient's clinical history. Care View presents all data such as vital signs and lab results two-dimensional along a horizontal timeline using a colour-scheme to identify values below, within or above the normal range. During the examination of a patient a nurse needs her hands free to support this CareView uses speech recognition to navigate between the categories of data or to enter data. Also WARD-IN-HAND [1] is a project in the medical domain. Doctors and nurses in a hospital are equipped with handheld computers that are connected to a server by a wireless network, to receive information about patients, clinical records, results of tests and so on. The main mode of interaction with the system is voice and secondary pen and touch-screen. Whereas the SHARE system [67] offers a multimodal user interface allowing voice and text input, an interactive digital map, location-based services and intelligent information processing and indexing. It is an information and communication system that supports fire departments and emergency teams during rescue operations and management of catastrophes. The system is based on the Push-to-Share service which extends the Push-to-Talk technology. Push-to-Talk allows direct user communication by simply pressing a button, but it is just an audio communication. The Push-to-Share service enlarges the communication with other media types like images, video and text. Another multimodal application is presented in [15]. With a handheld device a police officer can create an intersection or road diagram for documenting traffic accident reports. He can use speech and pen inputs to place icons representing the participants involved in the accidents and attributes of their vehicles like colour, turning indication or lights to create the accident diagram on a PDA instead of drawing it on a paper.

Users not only prefer multimodal interactions they are even more efficient with this kind of interface [6]. This is the result of the evaluation of the multimodal service Record & Replay. With this service the user has the possibility to write and record a structured document for the World Wide Web, e.g. a Blog or a Wiki. Furthermore the paper describes the design and realization of a multimodal platform.

3.3. Context-aware Mobile Applications

3.3.1. Location Awareness

Mantoro and Johnson [35] establish location as the most important aspect of context for mobile users. In their work they present the helpfulness of location awareness history for a context-awareness environment. To identify user's location a history database of events is created. Location data is created whenever a user identifies him via iButton or login to the network and with receptors, sensors and actuators like Web cams and badges. Users can give speech commands for finding the nearest resources, navigating or locating objects and people. The Speech Context Agent interprets the speech commands as SQL commands to the location awareness history database. The result is sent to a speech synthesizer, thus the system supports speech input and speech output and therefore allows the user to interact with his computational environment more in the way he does with other people. Whereas ComMotion [36], another context-aware communication system for a mobile computing platform, uses speech and text for input and output. Its main feature is the delivery of dynamic user-defined content, delivered when the user is at the relevant learned location. The system recognizes frequented locations, stores the GPS data and asks the user for a virtual location string tag. The user can assign to-do-lists to locations or class of locations. Other users can send reminders to a user at a specific location or query the user's position. The system also provides mobile access to location-based information from the Web. When the user enters a virtual location he receives an auditory cue that informs him of relevant messages to this context. Different auditory cues are associated with the various types of information (reminders, to-do-lists, subscribed content information). All text data can be received as text or synthesized speech and the interaction with the system is done with speech commands or through the graphical user interface. Also GCCM (GeoCollaborative Crisis Management) [5] uses geospatial information, it describes a collaboration system for managing crisis situation. It provides a multimodal user interface where team members can communicate and collaborate through speech and gesture with shared maps and with each other.

The Mobile Reality framework [24] offers the user a seamless, location-dependent, mobile multimodal interface. The interface allows the user to navigate through a three-dimensional graphical view, using location-sensitive speech interaction with the objects around him and it supports mobile collaboration like shared VRML browsing with annotations and a full-duplex voice channel. The application area for such an interface is mobile maintenance. PlaceMemo [20] is another mobile system that supports mobile workers. In this case infrastructure managers and road inspectors who need to identify,

report and take care of defects along the roads. The system uses a handheld computer and a GPS-receiver to allow users to record voice annotations connected to geographical locations. While working on the road the user can record a voice memo and the system saves the geographical position in addition. When the user reaches a previously marked location once again, the system will play the voice memo associated with that location. The system has a stationary mode to administer placed voice memos. Based on the routes driven the system shows a simple map with a grid where routes are presented as lines and flags along the lines symbolize the recorded voice annotation. The user can select a flag to get additional information like time and date or to mark it for deleting. He can display a list of all recordings and he can listen through the recordings.

Many multimodal applications are mobile guides that inform users like tourists, museum visitors or hikers about interesting places, show the route on a map and are a guidance for the environment. LoL [55] is such a tourist guide for the city of Vienna, the abbreviation stands for Local Location Assistant. [54] implements a tourist guide as a demonstration application to show the usefulness of the developed platform. The Web service-based platform facilitates the development and deployment of context-aware, integrated mobile speech and data applications. The demo application is a context-aware application providing tourists with interesting information and services of the visited country like dynamic navigation, meeting friends and information about interesting places. These points of interest are discovered by using the profile of the user which can be imported or the user can describe his interests. An interactive map is used to display static points of interests like tourist attractions, restaurants and dynamic points of interests like the location of friends, services like calling the friend, reserving a table at a restaurant or retrieving detailed information.

A mobile location-aware, multi-device museum guide is UbiCicero [22]. The guide provides the user multimedia information like videos, audios and graphical maps regarding the artworks. The mobile guide is a PDA equipped with a RFID (Radio Frequency Identification) reader and the museum artworks are fitted with physical tags that makes possible that the guide can localise the position of the user. Then it updates the displayed museum map and alerts the user that it has detected artworks nearby with interesting information. Artwork and room descriptions are dynamically created by a TTS module, because of limited memory on PDAs, audio files would require more physical storage and additionally text can easily be managed, because it has not to be read and recorded to create an audio file. Additionally the guide logs information about the user during his visit, e.g. the time he spent by listening to an artwork description, to estimate user preferences and to provide the user personalized information and suggestions regarding the next artworks to visit. To support and even enrich the user's experience the guide provides educational games for individuals or groups of players.

For extending the functionality of the mobile device especially while playing a game, the guide can exploit a larger screen of a stationary device. This resolution to augment a PDA with a surrounding screen or other computing devices near the mobile user is as well the subject of different papers, presented in chapter 3.4. Another multimodal guide for a cultural heritage is presented in [2]. Users can inform themselves about the historical wood ceiling at the University of Palermo. They can navigate a virtual representation of the medieval paintings and interact with the multimodal interface using speech or ordinary visual input devices. On the Bundesgartenschau 2005, a horticultural show that took place in Munich, visitors could rent the so-called BUGAbutler [25], a GPS-PDA that showed interesting places and navigated the user through the 330 hectare area of the show. It showed via display and by voice the current place of the user, offered information about the surrounding plants and objects, informed about actual events and could route the user to a venue. In addition the user could experience with geocaching, a paper chase supported per satellite.

Some applications use additional input modes to voice and text, for example gestures and handwriting. SmartKom [57] is a multimodal interface that uses an anthropomorphic personalized interface agent which uses speech and gestures for the interaction with the user. A part of the SmartKom project is the SmartKom Companion [7] which can be used while walking around and while driving a car where it is plugged into a docking station. Switching between these different mobile environments is easy, because the interaction with the system can be continued seamlessly. The main functionalities of the system are navigation and location-based information like planning a route, getting and presenting information about points of interest and guidance and monitoring of the user. Car-specific services are route planning and parking place reservation, while for pedestrians there are other places of interests relevant like historical buildings and museums or shops.

3.3.2. Working Context and Social Awareness

There are some projects providing solutions to the challenge of an everywhere and everytime messaging system [63]. Such a system should give the user the ability to send and receive messages everywhere and at any time. The requirements of such systems are minimizing interruption, while ensuring delivery of important messages timely, adaptation to the user's behaviour, location awareness and unobtrusive user interfaces. CLUES [37] is a dynamic message filter. It uses a number of sources of information about working relationships to infer which are relevant messages and to prioritise them. CLUES extracts meaningful items from calendar entries and other information sources to use them to find messages that are important for the user and to

categorize and prioritise the messages. Nomadic Radio [59], an interface that manages voice and text-based messages on a wearable computing platform, uses CLUES. The user has two modes for interacting with the system: navigating among messages and asynchronous notification of newly arrived messages. Content filtering is used to prioritise messages and based on the user's recent activities and the context of the environment the system determines the user's interruptability. The system operates primarily audio-only. Email, voicemail, hourly news broadcasts, and calendar events are automatically downloaded to a wearable audio device. The user has a wearable audio device to give spoken commands and the system uses synthetic speech to present textual messages such as emails and audio streams for voicemail and broadcast news. MailCall [38] is another messaging system in a non-visual environment. It also categorizes and prioritises messages by importance with CLUES and supports random access to the messages using speech recognition and synthesis. The system is used only via telephone.

Muñoz et al. presents a study on instant messaging in a hospital [44]. The user can specify contextual information for messages with his handheld system that has an interface for instant messaging. The study discovers four critical contextual elements: location, delivery time, role reliance and artifact location and states. A message can be relevant only for a specific location. A message is not sent immediately, the user can specify the delivery time. Messages are often addressed to special roles and not to particular individuals. For supporting timely delivery of pertinent information the relevant artifacts are monitored. Awarenex [71] is another interface for mobile devices that integrates awareness information and instant messaging. And it has also a speech interface to provide access over a telephone. The primary components of Awarenex are Contact List and Contact Locator. The Contact List shows a list of all users including awareness information like where that user is, whether he is logged in and using Awarenex, has been idle or is engaged, which means that he is involved in any communication activity. So this list helps to determine whether a person is available for contact. Clicking on an entry of the Contact List opens the Contact Locator which shows detailed awareness information to help the user to determine the best way to contact that person and it presents the relevant communication resources like instant messaging or email.

AwarePhone [3] provides social awareness among clinical professions. Social awareness relies on knowing the current work context of the co-workers which is important when initiating a conversation with another person. Especially the cooperation between clinical professions benefits from social awareness. A nurse needs to notify a doctor without disturbing him and a young doctor wants to contact a more experienced doctor and needs to know who is available and where he can find him. AwarePhone has

two main interfaces: the contact list and the message list. The contact list shows the user's list of contact persons. Three context cues are associated to each person describing the personal status, the current activity and location. When the user clicks on an entry of the contact list he can choose if he wants to phone the person or to leave a prioritized written message. The message list displays incoming and active messages in a prioritized order. With this list the user can read, reply and delete the messages. Additionally the paper describes the design and implementation of a framework for developing applications providing social awareness also for other mobile co-workers than medical personnel in hospitals.

The Klinikum Saarbrücken in Germany has started a pilot project in 2005 with the RFID technology [53]. In the admission ward each patient gets a wristlet with an integrated RFID chip, which includes the number of the patient. Physicians and nursing staff can read out the number with the aid of Tablet PCs or PDAs and get access via WLAN to the protected database to retrieve all details about the patient's record including the given medicine and their dosage, or further data that are important risk circumstances like allergies. The possibility to identify patients and their records easier and faster intensifies medical care and makes rationing and dosage of drugs more secure. Furthermore patients have the possibility to get specific personal information via infoterminals, like blood pressure, weight, therapy appointments and the end of hospitalization or general information about medical conditions, diagnosis and therapies.

3.3.3. Environment Context and User Situation

Djinn [32] is a multi-modal interaction framework to model interfaces for home environment based on speech and vision. It collects information from cameras, sensors, devices, appliances, etc to exploit the residential environment context. The user can control and communicate with his home via the PDA or telephone using basically voice commands. For example he can use a PDA to set the oven into a specific cooking mode, the GUI displays the items like temperature, duration and recipe and the user can set them all by one spoken command. [77] presents a mobile multimodal system which aids a user in finding out product information and product comparison information while shopping. The user can interact with the Mobile ShopAssist using speech, handwriting and gestures. Gestures can be performed either off-device like picking up a product from the shelf, so that product information can be retrieved through the RFID-tag or directly on the mobile device like pointing on an object displayed on the screen.

[31] describes an architecture for developing and executing mobile multimodal applications using multimodal interfaces and contextual information processing in

synergy. Contextual information, such as available modality, user situation and device information is used to decide which output devices are to be bound best into the actual user interface.

3.3.4. Collaborative Working Environments

In the past most mobile context-aware applications were developed only for single and independent users who are interacting with the system and they often used only spatial context information. New emerging team forms [70] in collaborative working scenarios can also take advantage of context-aware applications but they need more information of the environment than location to support them in their team interactions.

[12] presents a team characteristics model that defines the significant indicators for classifying emerging teams into Nimble, Virtual and Mobile Teams. The model combines related characteristics into the following five views: Spatial, Organizational, Project, Human Interaction and Service View. This model also structures team requirements in collaborative working environments into four categories: Management, Interactions, Information and Technology Requirements. The required features are implemented with Web services which are performing activities [61]. An activity is defined as everything people do in a collaborative working environment. The user can organize work and team collaboration according to these activities. An activity may be a basic activity or a composition of sub-activities. Moreover activities can be remodeled, sub-activities can be added or removed during runtime and sub-activities can be assigned to different team members. Providing such a flexible and reusable composition of services addresses the requirements of ad hoc collaboration.

Virtual teams mainly require ad hoc processes where the control flow between activities cannot be defined before execution and therefore the process cannot be modeled in advance. Team members of virtual teams require information on all work activities of all team members for a certain project. Process-aware tools are required to support virtual teams in ad hoc as well as collaborative processes. [16] discusses concept, design and implementation issues for process-aware collaboration systems and presents such a system called Caramba, which supports virtual teams on the Internet. Caramba users do not need to model processes in advance to achieve process-awareness. Users of Caramba can coordinate the activities of a process that is not based on a process template with Organizational Objects. Where Objects like Persons, Roles, Groups, Skills, Units, Organization, Tasks and Documents and their relationship are used to model organizational structures and responsibilities. Or virtual team members may use the Process Modeler component if it is able to model a process template. Caramba

supports modeled process templates, ad hoc activities and a combination of both.

A user of a collaborative working environment may be a member of many teams and is working on multiple tasks, projects and activities at the same time and he may want to collaborate while on the move. Because such users handle a lot of information filtering of content is very important for them. Information is relevant at a certain time and depends on the user's current situation and his context. Moreover amount and level of detail of the information is depending on time, situation and context. Using granular context [13] allows retrieving information in the relevant level of detail and therefore reduces the amount of unnecessary information exchanged which furthermore reduces network bandwidth and computing power which is especially important for mobile users who are using small devices and laptops. Granular context is modeled in a hierarchical form from the most generic information to the most detailed information. For example the highest level of a location information is country, and then city, street, floor and the lowest part is the room. For team members working at different location a higher level location information is needed such as city and street. Whereas for team members who are in the same building but in different rooms the relevant location is more detailed, at room level.

Besides spatial information like the geographical distribution of the team members, organizational information about team hierarchies, project management information about resources and artifacts and human interactions in collaboration are used to establish the team context [60]. Sequence and type of reoccurring human interactions in collaboration are described with interaction patterns. The interaction context defines the scope of the interaction and the different roles in a team. Team-awareness can be achieved by using context information as location, availability and current workload of the team members and considering the time-schedule of planned work and the deadlines of the projects. Such context information can be used to retrieve for example the team member who is able to process a certain request due to his availability, nearest location, no pending tasks and so on.

The inContext project [72] is a pervasive and collaborative working environment especially for all emerging team forms. inContext supports collaborative teamwork with common services like calendar, instant messaging, task and document management and with diverse other services to support emerging teamwork and autonomic capabilities. Different types of collaboration services, especially Web services that are loosely coupled and can be easily composed and adapted to different teams are integrated. Context related to teams and users, their activities and environment is modeled with ontologies. The inContext context model consists of reusable existing ontologies for modeling persons, addresses, geo-spatial context and so on and of five

new developed core ontologies: Location, Resource, Activity, Team and Action Context Ontology. The Context Management component collects, aggregates and provides context information for context-aware service adaptation. Within this scientific research activities and especially modeling the relations of team members with their work activities and the used resources are emphasized as the key component for retrieving context-awareness in collaborative working environments. Additionally the inContext platform uses meaningful patterns from observed interactions to enrich context information and for selecting and ranking services. The intensive use of context information and interaction patterns to adapt collaboration services to the changes and requirements of teams and environment reduces the necessity of human intervention.

3.4. Composite Device Computing

An interesting idea to overcome the limitations of small devices like PDAs is presented in [52]: a Composite Device Computing Environment. CDCE provides an infrastructure to augment a small mobile device with the available surrounding computing resources. The framework provides mechanisms for exploiting and interacting with “the world that surrounds the mobile user”. It describes this framework that supports users of small screen devices with a communication network infrastructure for seeking surrounding devices, to overcome small client constraints, to retrieve rich contents and access diverse services in conjunction with these other devices. Another CDCE is presented in [64]. An UPnP (Universal Plug and Play) implementation allows users to access larger screen devices like PCs, laptops, monitors, TV sets, public terminals, etc and to remotely control those with the PDA to extend the capabilities of the small screen device to provide access to rich multimedia context and services without content reduction. The key is that the devices incorporate and that computing tasks are outsourced so that each device does what it is best suited for. A possible application domain is in a hospital, as presented in [51]. Each doctor is equipped with a PDA and has wireless access to the patient information system. If he visits a patient his PDA detects the presence of possible other devices like a TV, his PDA communicates with the gateway server, x-ray images are transmitted to the TV for viewing and the doctor can use his PDA to annotate a region of the x-ray. So the PDA builds, with other devices, a convenient infrastructure for the doctor so that he can access, view, interact and collaborate upon the multimedia information.

3.5. Multimodal Architecture

A project at the Forschungszentrum Telekommunikation Wien is MONA – Mobile multimOdal Next-generation Applications [42]. This research project deals with the problem that developers had to adapt the user interfaces of their applications so that a user is able to access the application from different devices. It presents a solution for developing and deploying device- and modality-independent applications that combine a graphical user interface (GUI) with a voice user interface (VUI), enabling the user to use text and speech as input and output modality on different devices. A MONA developer specifies his user interface on an abstract level with an editor especially developed for the project, based on a special developed UIML (User Interface Markup Language). An application developer of this project is not concerned with device specific issues. He provides a single implementation of the user interface and the MONA presentation server renders a multi-modal user interface for each device which is accessing the application. For supporting other devices the application needs no update. In the context of the MONA project two prototype multimodal applications were developed, one in the business and one in the entertainment domain. The MONA Quiz is a multi-user quiz in the style of "Who wants to be a millionaire?" The players interact in real-time; each player is represented by an avatar which expresses his different moods depending on the current game situation. Additionally the quiz provides a multimodal chat for user-to-user communication where sending and receiving messages can be done visually or by voice. The MONA Server automatically translates spoken words to written text for GUI-only users and vice versa text-messages are translated to voice by TTS (text-to-speech). The second prototype application is a mobile multimodal unified messaging client that enables the user to administrate emails, SMS and voice messages. It addresses to a business user who wants to get an overview of new messages while he is not at his office. The application informs the user when a new message has arrived, he can read the text or listen to the synthesized audio output.

Another experimental platform for mobile information systems that supports the rapid prototyping of multi-channel, multi-modal, context-aware applications is presented in [4]. The platform consists of a Web publishing component, a cross-media server and a client controller. The client controller is responsible for the input and output handling with the user; he sends requests to the server and activates the appropriate output channel. The cross-media server delivers the information from the content database or the active content. The Web publishing component includes a context engine, which transforms the information to the appropriate format of the output channels. All information such as structure and presentation of Web documents, as well as the content, are represented as database objects which can be updated at runtime. The platform was used to develop a tourist information system for an international arts festival where interaction

was based on a combination of speech input-output and interactive paper. For supporting locator and navigation tasks the user also has a GPS device. Visitors of the festival can use the special interactive paper brochure containing a map and an event list with a digital pen to retrieve information about events and locations of the festival. They point with the pen on the map and the system initiates a voice dialogue via an earpiece to get additional information from the user on the one hand and on the other hand to give the desired information back to the user. Users can set a reminder for specific events and they can write short comments, which are stored in a database and can be requested by other users.

Fabbrizio et al. introduce a speech mashup architecture [11], a network-hosted application framework for mobile devices like Smart Phones, iPhone and BlackBerry that makes integrating Web content and speech processing easier. There is no need to install, configure or manage speech processing on the multimodal device. The framework consists of a speech mashup server for speech recognition and text-to-speech synthesis, a Web mashup or application server for additional service logic and a client running on a mobile device or a Web browser. The client application captures and relays speech or text to the speech mashup manager over an HTTP connection, which communicates with the speech server where the speech processing and the text-to-speech conversion takes place and relays text or spoken response back to the device. The speech mashup manager accepts and returns three data formats: XML, JSON and EMMA, the data format for the TTS engine is SSML.

3.6. Usability

In the literature some interesting studies compare voice with other modalities and examine the usability of voice in user interfaces. A study on information retrieval research is presented in [14]; it compares forming queries in written form with forming queries via voice to evaluate the feasibility of spoken queries for search purposes. The results of the study are that spoken queries are lengthier and contain more stop words, but the ease of speaking encourages people also to express more semantically. Unless as expected they found no significant differences on durations for formulating queries in spoken or written form. Something similar is evaluated in [46]. This work analysed if the absence of a visual display impairs usability in a gesture input system that is combined with speech output. The system provides information to patients in a hospital; the display shows a GUI presenting the available services and the gestures that invoke them. The result of the study is that there are no significant differences in the number of incorrect gestures made by participants using the system with or without the display, but users of the system with the display take longer than those using it without it. Both user groups the one with the GUI and the one without were given a training period practising the gestures, the one without the GUI had to remember the gestures, the other group were able to see the corresponding gestures on the GUI while they were asked to use the services. Again there were two groups, one with a training time of 5 minutes and the other one had 10 minutes for training. For those who could not see the visual display an additional training time decreased the number of incorrect gestures made. And on the other hand for the group that had a visual display, additional training time increased the number of incorrect gestures. This result was not expected. So the solution of this evaluation is that the presence of a GUI does not enlarge the count of incorrect gestures and the absence of a GUI did not lead to larger processing time, on the contrary the mean processing time was longer with the presence of a GUI, because the people were looking at the display. [50] presents an evaluation of unimodal and multimodal form-filling systems on desktop and PDAs. The study found synergies between speech and visual modes. Multimodal interfaces have shorter interaction times, visual feedback like GUI output and input modality choice is important. The results show that integrating visual output in spoken dialogue systems increases the efficiency significantly and that giving the user the possibility to choose his preferred input modality decreases interaction time significantly.

4. Framework

4.1. Scenario – Schedule Meetings & Calls

The most commonly needed services in the context of collaboration are calendar and address book applications for requesting and making appointments. On the basis of a team scheduling application we will demonstrate what kind of problem we want to solve within this approach.

Figure 10 illustrates the following scenario:
A project manager receives new information and data concerning a special project of one of his customers. Now he wants to call a meeting to inform all team members, discuss the details and plan the workflow. Using a common Web application he fills out a Web form with all data specifying the meeting, like costumer, project, start date and start time, duration, location and he selects the persons he wants to invite to the meeting. When he has finished, notifications are sent to all specified persons including the details of the appointment.

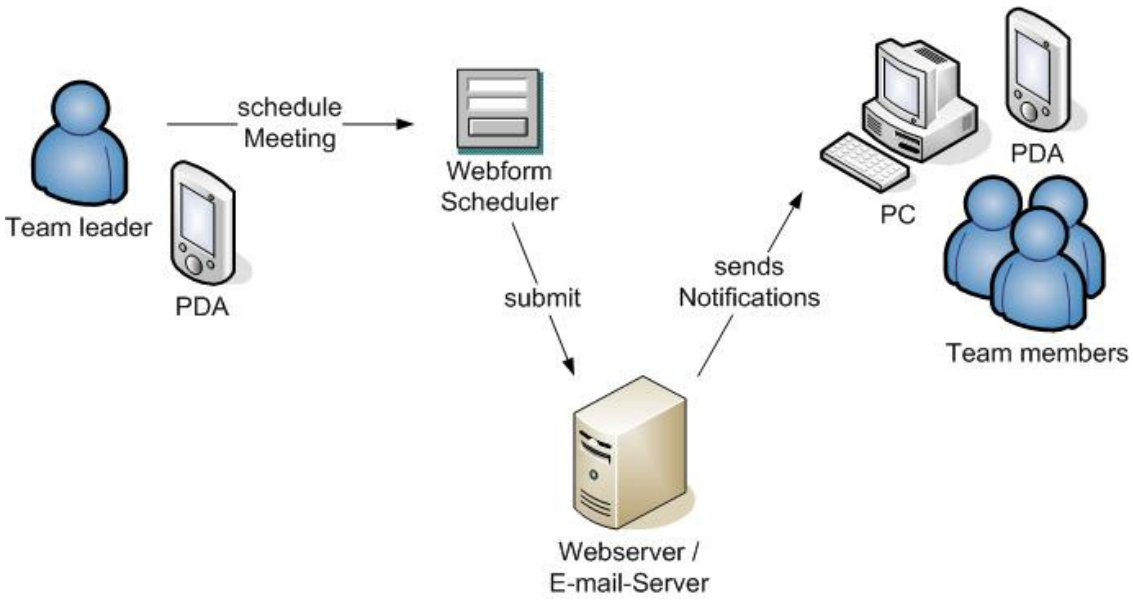

Figure 10: Basic scenario – team scheduling

If the project manager uses a mobile device with a small display and a pen for giving inputs he will encounter difficulties while filling out the form. Often he is included in many projects, so that the selective list of projects will be long and more than ever the list of

persons will be pretty large. Select boxes with a large list of options can soon be too large for a small display such as the screen of a mobile device. Scrolling down and selecting an item out of a long list of options is hard with a small screen resolution. Inserting date and time are as well difficult because the keypad of a mobile device like a PDA is small or it has even only a virtual keyboard which covers a part of the display and so reduces the already small screen.

A speech-enabled Web form gives the user the possibility to fill out the form by giving voice commands. For selecting an option of the list of projects or the list of persons, the user simply says the name he wants. If he wants to know which options are possible, he clicks on a button to hear them all and then he selects one of the options with his voice. But he has not to listen to the whole list of possible options, if he knows which one he wants to select, the reading will be stopped soon after his voice input. He can choose a start date and time by simply saying for example “seventh of September 2010 at five o’clock in the afternoon” or a similar phrase and the date and time fields will be filled out as usual with “2010-07-09 17:00”. A great advantage of a speech-enabled Web form is the possibility to fill out the whole form at once. The user can talk to the application in a whole sentence like in a dialog of a human-to-human communication. The application takes out the special words needed to fill-in the form and ignores the in-between phrases.

Regarding context data and relevance criteria in speech input and output will further improve the usability of such an application. The system can use context data to reduce the list of options of a select box, so that a user has not to listen to a long list of unwanted information. For example, the list of projects contains only those where the user is currently working on; the list of persons contains only those with whom the user had contact during the last week or only the team members of the user and his current projects. Location information of the device can be used to offer only that locations for a meeting in the Web form that are close to the current position of the user or the device, for example locations that are in the same city as the user currently. Furthermore location information can help to identify the kind of appointment, if it is a meeting or a conference call. If the user is far away or he is on the way then it can be assumed that the user wants to make a conference call instead of meeting the persons in real. Figure 11 illustrates the team scheduling scenario enlarged with context-based multimodal interaction.

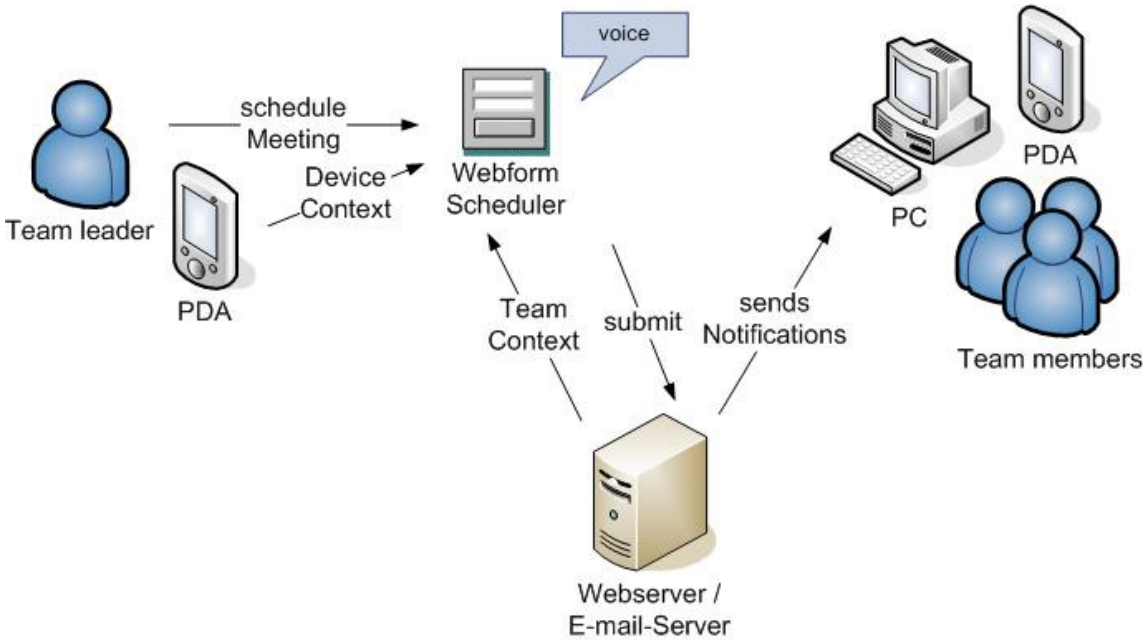

Figure 11: Team scheduling with context-based multimodal interaction

Integrating context and relevance information in multimodal application reduces long lists and complex grammar rules, so usability and speech recognition rates are improved, but also network bandwidth or battery consumption are reduced. Because there are two ways speech recognition can be integrated: either on the client or on the server.

Besides making these preselections we have to consider that the user has the possibility to enlarge or change the selection criteria. The user should have the possibility to specify these parameters or to get out the whole data list. For example, if the user plans a meeting concerning a project that does not occur in the list, because the user is not currently working on it, he needs the opportunity to enlarge the list and select another person. Or if the user wants to invite a person to a conference call who is not in the list because he is not actually in the same projects, but this person can be helpful with the current problem. Context information will be used to make recommendations to the user, but it should never be obligatory.

4.2. Design

We divided the context information into two parts: the scenario-specific and the device-specific part, see Figure 12. The scenario-specific data objects contain all relevant information about projects, tasks, teams and the team members and their location. The device-specific data contain details to identify the situation of the user and his device to decide which output mode will be recommended.

The scenario-specific objects are: project, team, task, member and location. A project can consist of many tasks and a task must be part of one specific project. A project is handled by one particular team. A team can have many members; a member can be assigned to a team. A member has a current situation and can be the assignee of many tasks. A task can be assigned to a member. A user is a special member and is always in a specified situation. We decided to use a simplified model, so that we can extend it in the future. A possible extension might be to allow that a project consists of more than one team, or we can restrict the relation between team and member, so that a member must be part of a team and a team must consist of at least one member respectively.

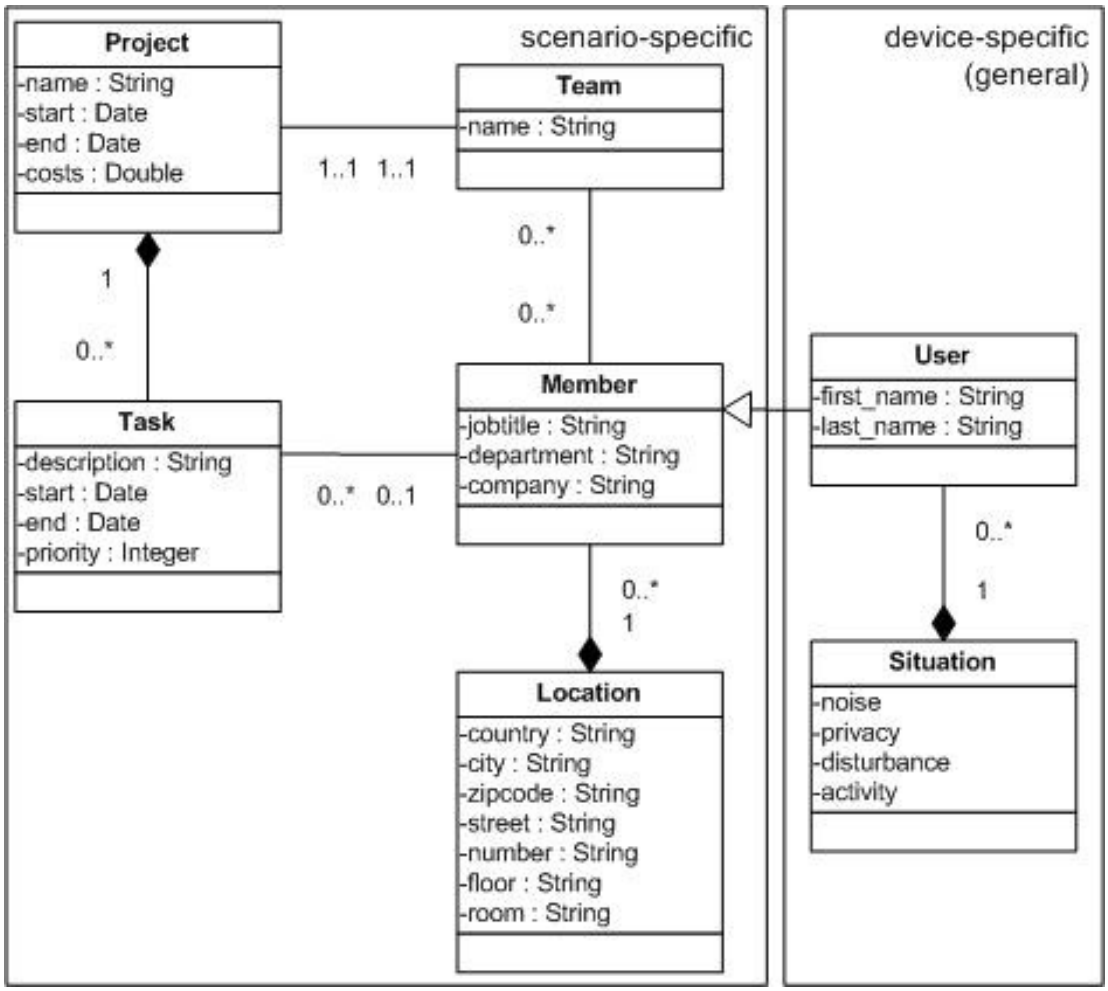

Figure 12: Diagram of context objects

The context of the device will be used to find the adequate modality, text or speech. The context object situation has four parameters: *noise*, *privacy*, *disturbance* and *activity*. To obtain the values of these parameters we can use different recognition methods: microphone, location data and other team scheduling context data or calendar information. The microphone of the PDA can be used to detect surrounding *noise*; the values can be loud or silent. The parameter *privacy* states if the user wants to keep his privacy, if he is at a public location or if he is at a private location like his office, so it can have the values public or private. The attribute *disturbance* means if there are other people around the user who can be disturbed or if he is alone, so the values are group or alone. *Activity* is a value representing if the application has the primary attention focus of the user or if the application has only the secondary attention focus of the user, for example if he is driving a car, so the values can be primary or secondary. This parameter can be achieved with calendar information of the user or his current location. Which modality is the best for the current situation depends on the values of these

attributes. The parameters, their possible values to decide between speech and text output and how they will be recognized are summarized in the following table:

Situation Parameter	Text out	Speech out	Recognition via
noise	loud	silent	microphone
privacy	public	private	information of location and tasks
disturbance	group	alone	number of members with same location
activity	primary attention focus	secondary attention focus	calendar information and location of the user

The best modality for the current situation depends on the values of the situation parameters, see the flowchart in Figure 13 which illustrates the decision-making of the best modality, text or speech. We decided to use a key-value model to define and store the context data. We use just simple 0 (=text) or 1(=voice) to decide between speech and text-only. So just if all parameters are set to 1 speech-output and speech-input will be used. In future extensions we can use a range of values instead of just 0 or 1 and decide which parameters have to be used strict and which can have a wider range to accept voice-output. For example, if a radio is playing in the office, noise will neither be silent nor loud and so it makes sense to use speech even if surrounding noise is a little bit loud.

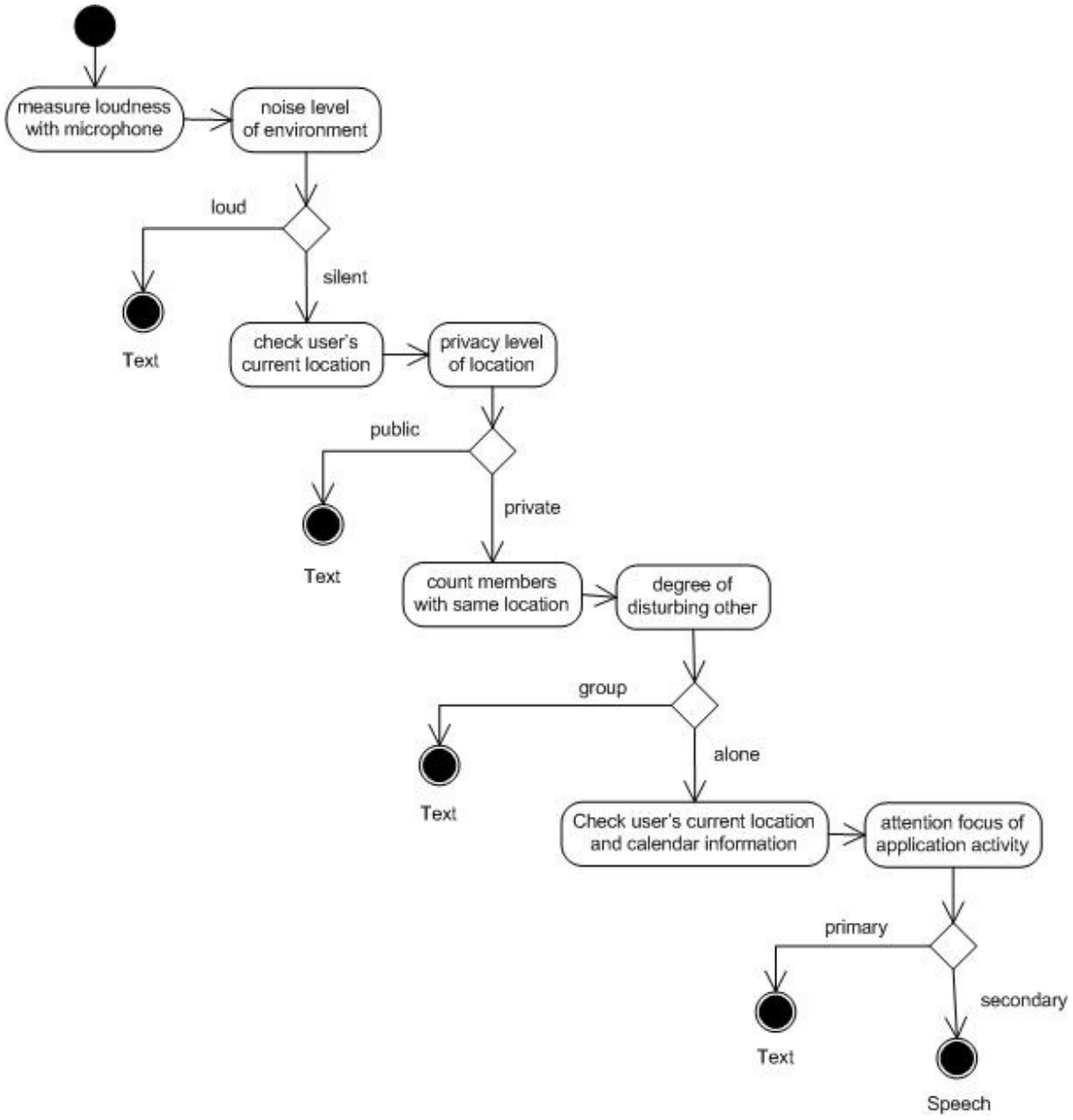

Figure 13: Flowchart of finding the best modality with the situation parameters

A use case is illustrated in Figure 14, a user can enter a new task or edit an existing task, which he chooses from the task list, and then the Scheduler updates the task list. The Scheduler can show a list of all tasks. If the user enters a new meeting or call, the Scheduler sends notifications to the user and all other persons that are concerned.

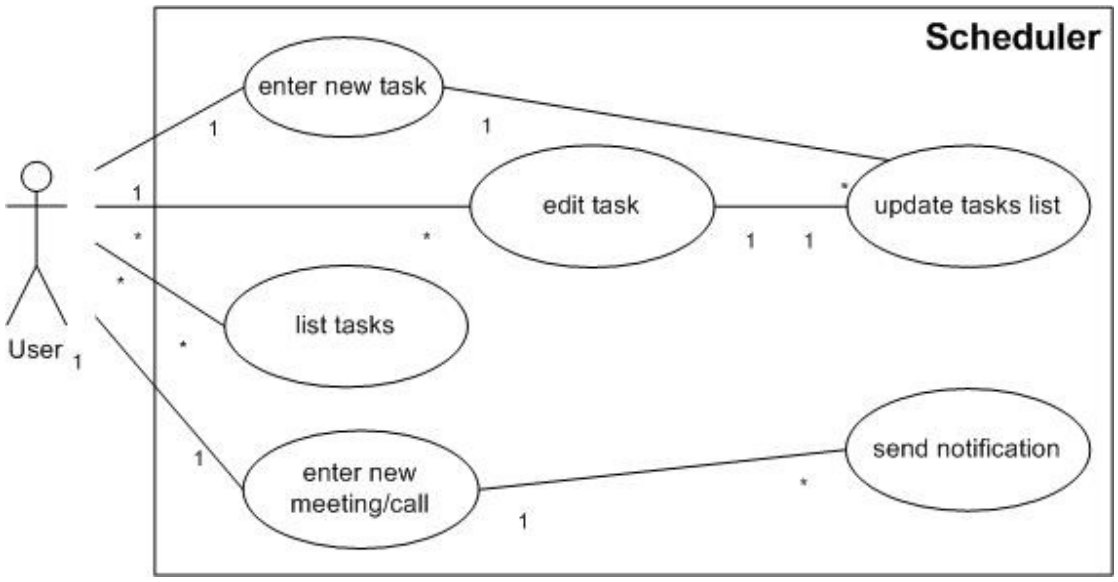

Figure 14: Use case

The basic components of our context-based multimodal team scheduling application are illustrated in Figure 15. On the one side there is a server hosting the team scheduling server application and the database containing all team, task and project data. On the other side is the PDA, the client, with a multimodal browser installed that displays visual and voice output and gets the input from the user. Furthermore the client has two applications running: the team scheduling client application and the device-context-data application. The Teamscheduler creates the multimodal Web forms including the scenario-specific context derived from the server and the device-specific context derived from the device-context-data application. Server and client communicate with SOAP messages. The client sends specific parameters via SOAP to the server, with these parameters the server can query the database, adapt the result to the context and then sends back the resulting context-aware data in a SOAP message to the client.

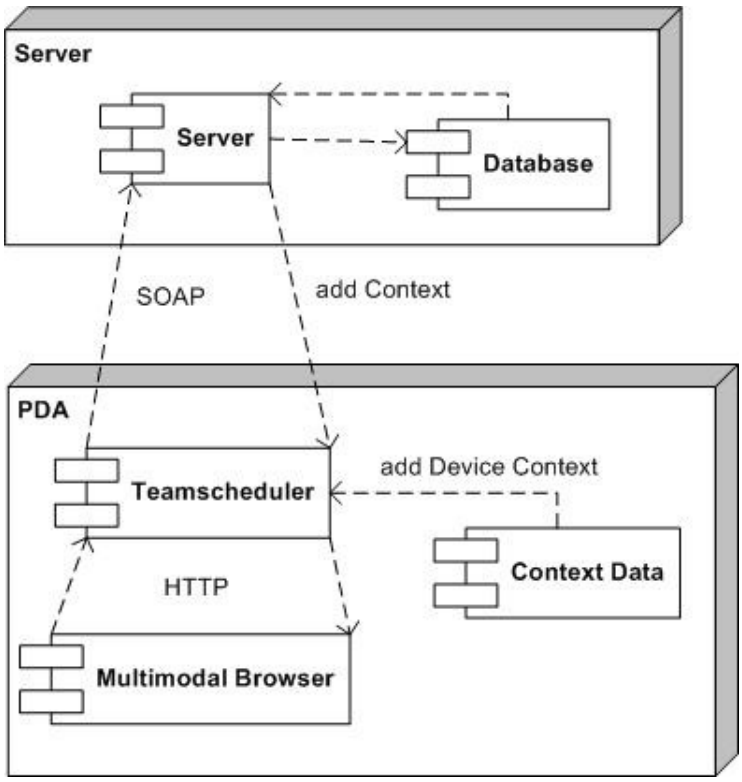

Figure 15: Basic component diagram

The sequence diagram in Figure 16 shows a few interactions of the objects of the team scheduling scenario. The first interaction starts with the user who questions the application for a list of all existing tasks by saying “show all tasks” or taping on the appropriate point on the GUI. Then the Browser sends the parameter “?list=tasks” with an HTTP request to the Teamscheduler which passes through the request unchanged to the Server within a SOAP message. The Server queries the Database and receives a list of all tasks in an array. Again over SOAP the Server communicates with the Teamscheduler and sends it the list of all tasks in an XML format. After receiving the data, the Teamscheduler checks the device context by querying the DeviceContext object that retrieves the situation parameters to check if activating the voice modality will be useful. This recommendation is sent to the Teamscheduler which sends HTML and if recommended VoiceXML output to the Browser. The second and third interaction in Figure 16 shows how context data will be used to achieve context awareness. Every time the user looks at the details of a specific task or edits a task the Teamscheduler gets a request from the Browser containing the id of the task for example “?show=task&id=xyz” as shown in the second interaction of Figure 16. Then the Teamscheduler adds the id of the requested task to a history element to save it. This history element is used to retrieve context aware data for the “editing a new meeting”-

form. If the Teamscheduler gets the request “?form=meeting” from the browser it sends to the Server a list with the task ids used lastly so that the Server can retrieve all data regarding these context, shown in the third interaction of Figure 16. More context-awareness for the Teamscheduler can be obtained. The list of projects can be adapted in a similar way, so that it shows only projects the user has lately viewed or edited. The list of persons can be adjusted to a list of those persons who had contact lately, who had been in meetings with him or are working in the same projects as the user. If the device has the possibility to get the current position of the user, for example with a GPS, this information can be used to adapt the selectable locations to present just meeting locations near to the user’s current location. The GPS position can be used to assume that the user wants to make a conference call instead of a meeting because his current position is far away or he is on the way.

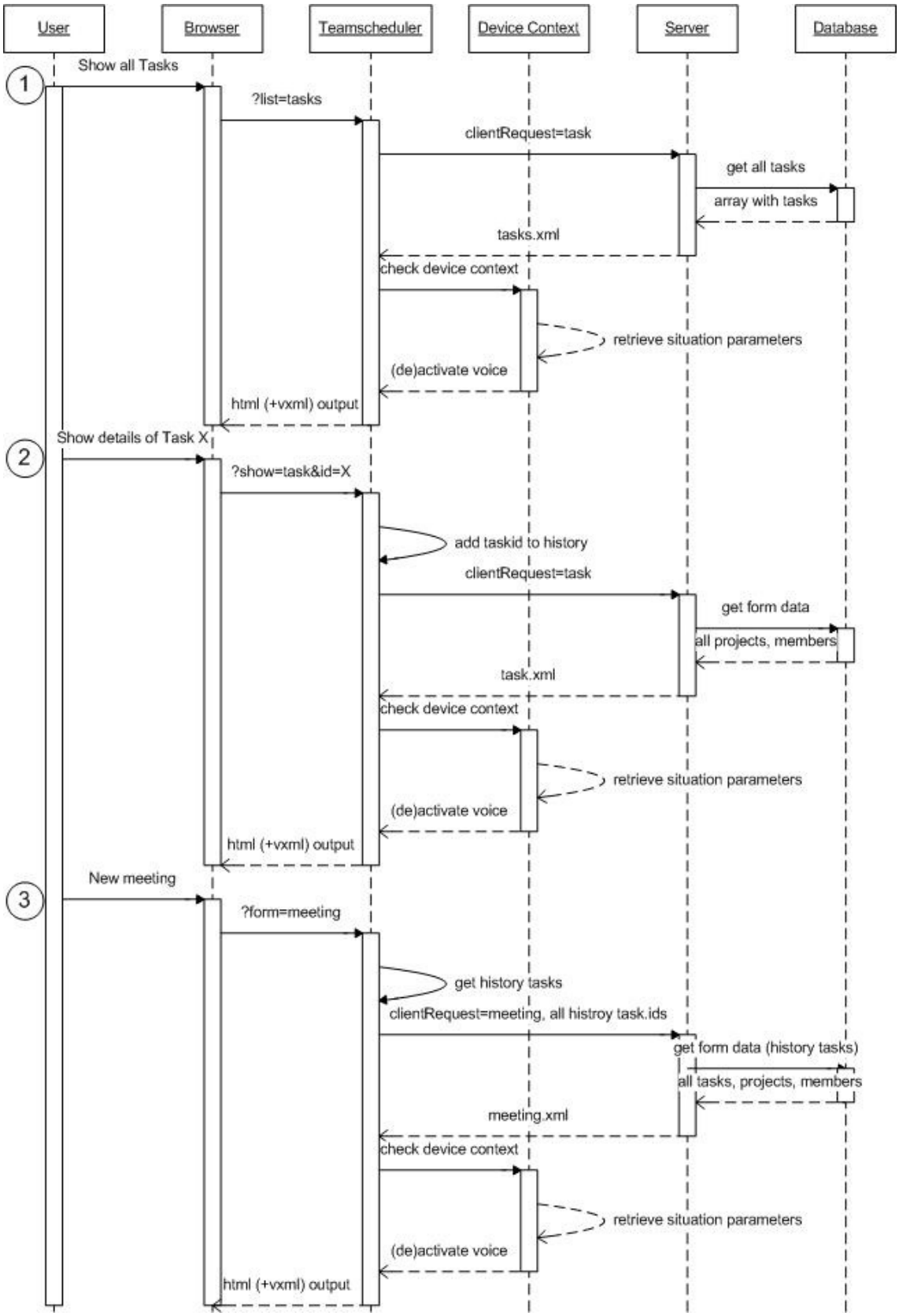

Figure 16: Sequence diagram

4.3. GUI and VUI Design

We decided to use X+V for implementing the multimodal user interface because it consists of existing XML compliant languages and the voice and the visual part can be designed and (re)used separately. First we designed the Graphical User Interface and implemented it in XHTML. The GUI consists of eight pages. The start page of the application shows the menu with all possible actions the user can do (see Figure 17): entering a new task, entering a new meeting, viewing a list of all existing tasks, viewing a list of all entered meetings and changing the configuration.

Figure 17: Graphical User Interface: Start page – Menu

On the list of all tasks each task (see Figure 18a) can be selected to show the details of the task and with click on an edit button (see Figure 18b) the user can change the properties of the task. The same applies to the list of meetings (see Figure 19a and b). Editing an existing task and entering a new task is one of the three different forms in the application (see Figure 18c). The user can give input for the following data of a task: description, project, start date, end date, priority and assignee. The description of the task is a text input field. The project of the task can be selected from a list box. Start and end date of the task are calendar dates. The priority of the task can be selected from a list box too; this select box contains the following five values: highest, high, middle, low and lowest. A multiple selection list box is used to assign the task to one or more persons, the assignees of the task.

a) shows the list of all tasks b) shows details of a task c) form to enter or edit a task

Figure 18: Graphical User Interface – Tasks

Another form is used to enter a new meeting and accordingly to edit an existing meeting (see Figure 19c). The following input parameter can be given for a meeting: project, task, meeting or call, invitees, start date, start time, duration, location, notification. Project and task can be selected again from a list box. If the meeting is where people come together at a certain place or if it is just a conference call can be defined with a radio button. One or more invitees for the meeting are selected using a multiple selection list box. Start date is a calendar date and start time represents the time of day when the meeting takes place. In the field duration the user can specify the duration of the meeting in hours and minutes. The location of the meeting can be selected out of a list box. If the application should send notifications to all invitees the user can specify this by clicking the correspondent checkbox of the form.

a) shows the list of all meetings

b) shows the details of a meeting

c) form to enter or edit a meeting

Figure 19: Graphical User Interface – Meetings

The third form of the application is to change the configuration of the lists, the user can decide which properties should be shown in the lists and the sorting of the lists (see Figure 20).

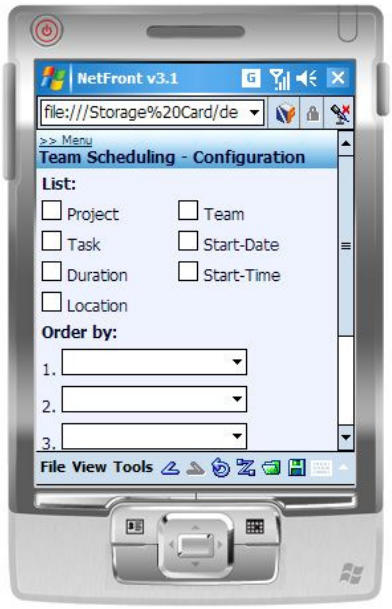

Figure 20: Graphical User Interface – Configuration

After designing the GUI and validating it is XML compliant, we designed the Voice User Interface with VoiceXML and JSGF grammars. We composed the voice part of meaningful prompts and reprompts in case of error, missing or misunderstanding input and with assistant hints. We enabled almost each graphical element of the XHTML pages for voice by entering VoiceXML snippets, correlating them with the visual part and creating JSGF grammars or using links to existing predefined grammar files like for example the date.jsgf representing calendar date and time. We decided that the dialog flow is always the same as the order of the XHTML elements in the GUI. All grammars besides the date grammar are simple grammars consisting of all possible values for the XHTML field combined with phrases like “the assignee is”, “the task is assigned to”, “the task is for”, “it is a task for” and so on. These grammars are all generated at runtime, we created just a few static ones for testing purpose. For calendar dates and time we used the existing predefined grammar date.jsgf contained in the Multimodal Toolkit of IBM’s WebSphere Studio framework. Additionally two static grammars are created: one used for skipping the input of a field or form and the other for the menu of the team scheduling application. The skip.jsgf grammar allows phrases like “go forward”, “go ahead” and “next” to skip the input of a field and give the attention to the next field. The grammar menu.jsgf holds the phrases for selecting the menu points. Then we designed a voice input field for each form that can be filled in at once when the user talks all input in one whole sentence. At the end of each form we built a dialog that reprompts all given input in a whole sentence followed by a question if the input is given correctly. So before the data is saved, the user has the possibility to correct his input by voice.

4.4. Implementation

The team scheduling scenario is implemented as classical client-server architecture, see Figure 21. The client runs on a mobile device in our case a PDA and the server is responsible for saving and loading the data from a database that holds the main data of the application. The client consists of a proxy application that exchanges data with the server application via SOAP messages. Furthermore the proxy on the client is responsible for the user interface and renders the visual and the voice part regarding the context information from the device.

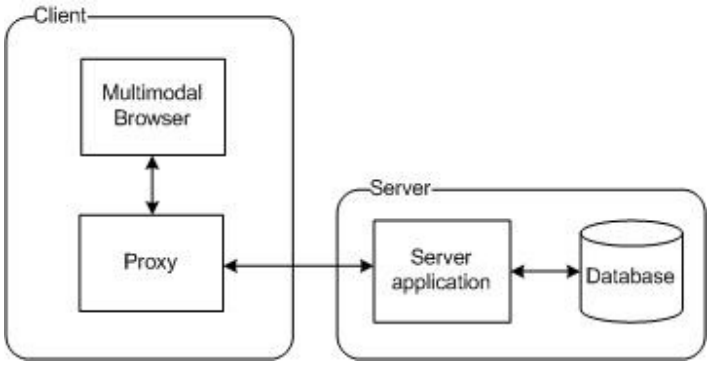

Figure 21: Software architecture

Our team scheduling scenario consists of two main components: a server and a client application, both written in Java and using the Eclipse IDE for development. The server part of the Teamscheduler consists again of two components: the server side application teamserver and the database. The server side application teamserver is a Java Web Service. All data is stored in a MySQL database. In our case the MySQL Server runs on the same computer as the Teamscheduler server application, but it might be separated. The team scheduling client application is written in Java too, but because it runs on a small device it is implemented as an OSGi bundle running within the OSGi framework Knopflerfish [33]. We are using the Java Virtual Machine J9 from IBM to run it on a PDA with Windows Mobile 2003 installed. On the client there are two bundles installed, the Teamscheduler bundle which uses a history context and the bundle which handles the device context to achieve context-awareness. Figure 22 shows the detailed component diagram.

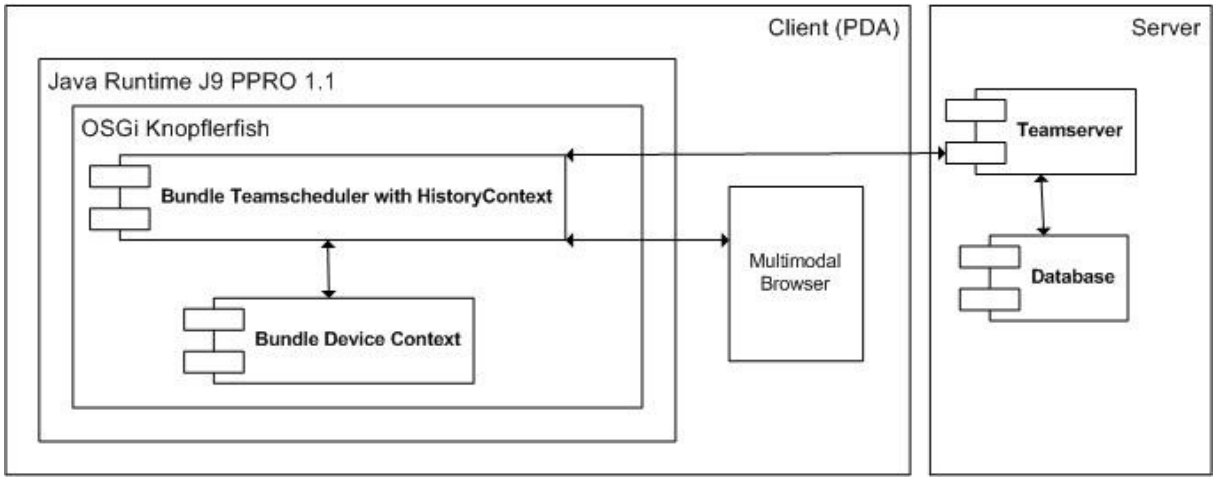

Figure 22: Detailed component diagram

4.4.1. Java on a Mobile Device

The Java 2 Micro Edition (J2ME) platform allows Java to be used on small and embedded devices. The J2ME specification defines just a small subset of the J2SE API and two configurations of the J2ME: Connected Limited Device Configuration (CLDC) and Connected Device Configuration (CDC). The CLDC is the smaller one and could be implemented on many different devices with limited memory, slow processors and having intermittent network connections like mobile phones and PDAs. Whereas CDC is more substantial because it includes a full-featured Java virtual machine, needs more memory, faster processors and greater network bandwidth. Both configurations support profiles: Personal and Foundation Profile are based on CDC, PDAP and Mobile Information Device Profile (MIDP) extends CLDC. MIDP is designed for very small devices like mobile phones and entry-level PDAs. It provides a complete Java runtime environment for handheld devices. Its functionalities are user interface, network connectivity, local data storage and application management. The Foundation profile based on the CDC provides most of the functionality of J2SE except for graphical support. The Personal Profile extends the Foundation profile with GUI support and is for high-end PDAs. The J9 JVM from IBM is J2ME compliant. The J9 is included in WebSphere Everyplace Micro Environment [79] and also part of WebSphere Device Developer [78]. We downloaded and installed a trial version of WebSphere Everyplace Micro Environment (WEME) 5.7.2 - Personal Profile 1.0 for Windows Mobile 2003 2nd Edition which includes a .cab file that can be executed on Windows Mobile for installing the J9 VM on the mobile device. The J9 runtime environment of these WEME contains Connected Device Configuration, Foundation and Personal Profile based technologies. CDC meets the requirements for OSGi container deployment and thus enables richer

Java applications on small devices. Also managed services can be deployed in OSGi containers in form of bundles.

4.4.2. OSGi Framework

We decided to use the OSGi container technology as a framework for our client application. The OSGi Alliance [48] has specified a Java-based service platform enabling the deployment of services over wide area networks to local networks and devices. The OSGi framework is the core component and it provides a standardized environment to applications. The applications and libraries are packed into bundles and share a single Java VM. The Framework manages the collaboration and the life cycle (install, resolve, start, stop, refresh, update, uninstall) of these bundles. The bundles can be remotely installed without restarts, they run unmodified on different hardware and software architectures. A large number of standard component interfaces for common functions like HTTP servers, configuration, logging, security, user administration, XML and more have been defined. All of them are implemented as bundles based on the Knopflerfish framework and can be run in this open source OSGi.

We use Knopflerfish as the framework for running our team scheduling client as an OSGi bundle. Knopflerfish is an open source implementation of the OSGi R4 framework specification. The OSGi Service Platform Release 4 specification defines a minimum execution environment that is a subset of Java 2 Micro Edition, the CDC configuration and the Foundation Profile. Knopflerfish can be installed using an installation wizard or by simply unpacking the jar file manually to a directory on the computer. It includes the OSGi runtime environment, the Knopflerfish framework and bundles, all Java source code files and a build system so that Knopflerfish may be rebuild locally and a documentation for users and developers. The framework can be started by running java on the framework.jar file in the osgi directory or simply execute the jar file within the file system because it is distributed as an executable jar file. After starting the framework a default set of bundles is started, including the Knopflerfish Desktop bundle and other useful bundles.

The framework can be controlled by xarg files. An xargs file sets the system properties, installs and starts the bundles. The framework can be further controlled by command line options and by setting system properties. When the framework is started at the first time it reads in the init.xargs file which includes all start options. All bundle data is stored in the default directory fwdir. The framework remembers its state from the previous start. It uses the restart.xargs file for restarting, that file includes no install options but sets system properties and uses the launch option for restarting the bundles. One can force a

restart with the initial options by removing the fwdir directory or by giving the framework.jar an adequate command line option. An xargs file must at least contain all the options needed for installing and managing the bundles even though an empty framework can be started too but nothing can be done in it.

We use the Eclipse IDE for developing the server part as well as for creating and developing bundles. Therefore we installed the Knopflerfish Eclipse Plugin so that we can define a J9 JRE under Eclipse for building, running and debugging a Java project for J9. The Plugin includes a special editor for .manifest files, using a graphical and a text view so that bundle.manifest files can be edited easily.

4.4.3. Multimodal Browser

Another component of the Teamscheduler client application is the Multimodal Browser which shows the GUI and VUI of the Teamscheduler and gathers input from the user by tapping as well as voice. There exist two Multimodal Browsers for Pocket PCs: Opera for Sharp Zaurus and NetFront by Access Systems for Pocket PC with Windows 2003. Both are enhanced with extensions that include the IBM speech recognition and text-to-speech technology. We used the NetFront v3.1 Browser where a trial-version is included in the Multimodal Toolkit of IBM. We installed the multimodal browser on the development PC for testing purpose. A trial version of the NetFront Browser for Pocket PC can be downloaded, it is a .cab file with a Device Installer that will be copied to the mobile device and can be executed on it directly.

Figure 23: Voice preference of NetFront multimodal browser

In the NetFront browser voice preferences can be set by selecting File -> Preferences respectively Tools -> Browser settings in the version for Pocket PC and then choosing the Voice tab, see Figure 23. All Voice features can be disabled but they are enabled by default. The Listening mode can be changed from Push-to-talk mode to Push-to-activate or Auto push to activate mode (see page 22 for a description of these listening modes). In the PC Version of NetFront a Push-to-talk key can be set, that is a keyboard key that can be used for activating the microphone for voice input. One of the following keys can be selected to be the Push-to-talk key: Scroll Lock, Ins, Shift, Ctrl, F8 or F12. Furthermore on the PC Version a Voice Log Level can be set to disabled, info, warning, verbose or severe. Via a checkbox it can be managed if a mouse click on the screen stops voice prompts. In the Pocket PC version of NetFront it is a stylus tab instead of the mouse click that can be enabled to be a cancelling feature. C3N stands for Command Control and Content Navigation which means that voice commands can be used to activate controls in the browser, these voice commands are:

- back, forward, home
- refresh
- page up, page down
- zoom in, zoom out, normal size
- show bookmarks
- show help and
- show voice commands.

A voice command must be preceded by a name which can be specified within the voice preference, it is “browser” by default. For example the user can say “browser, show bookmarks” to see a list of his bookmarks if the C3N option is enabled. “Browser show voice commands” will show a list of all voice commands that can be used. After making changes a restart of the browser is required to activate the changes.

4.4.4. Team Scheduling Server Application

The team scheduling server application consists of two components: the server and the database. A MySQL server is used for data storage in a MySQL database. In our case the MySQL server is running on the same computer as the team scheduling server application, but it can be separated to different computers. The database structure is nearly the same as the structure of the context objects which is described in Figure 12 on page 58.

The database consists of eight tables, Figure 24 shows the details of the database tables and their relations. The project, task, team, member, location and meeting tables hold the corresponding data. The table task_member stores the assignees of a task, it is an m:n relation between task and member. Accordingly meeting_member stores the invitees for a meeting, an m:n relation between meeting and member. The table project holds the unique name of a project, its start and end date and its possible costs. For each task is stored the description of the task, its start and end date, the priority that is a range from highest, high, middle, low to lowest and the project name the task is part of. The team table stores just the name of the team. For all members included in the team scheduling scenario is stored the first and last name, the job title, department and company where the member is working and his current location. This is the primary key of the table location that stores country, city, zip code, street, number, floor and room number representing the location where a member resides. For each meeting is stored: the project and the task it deals with, the form it has, which means if it is a meeting or call, start date and time, duration and location of the meeting and if notifications are sent to the invitees of the meeting. The invitees of a meeting are stored in the table meeting_member, each member can be invited to one or more meetings, a meeting can have one or more invitees. The table task_member stores the relation between member and task, each task can be assigned to one or more members and a member can be the assignee of one or more tasks.

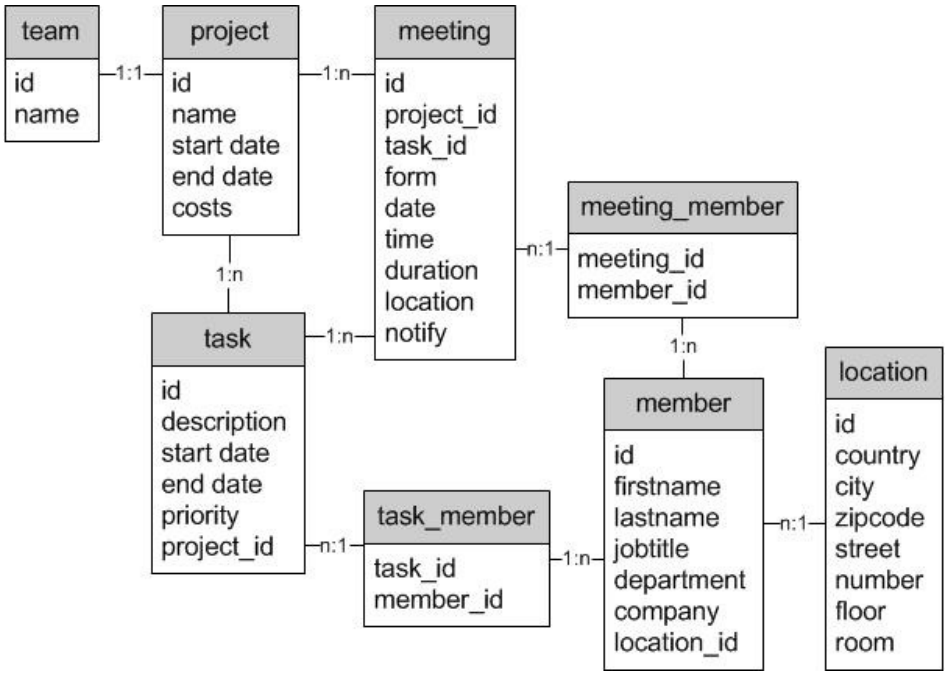

Figure 24: Database tables and their relations

The team scheduling server application consists of a package containing two Java classes. One that handles the database connection, queries the database and stores data to the database. The other part handles the communication with the client team scheduling application. It retrieves the request and sends back the required data within a SOAP message.

4.4.5. Team Scheduling Client Application

The team scheduling client application runs on a small device and uses an OSGi container as the application framework. We implemented two OSGi bundles: one is called teamscheduler.proxy and the other is called teamscheduler.devicecontext. Both are HTTP servlets running as bundles in the Knopflerfish framework. The bundle teamscheduler.devicecontext retrieves the values of the parameters of the context object, representing the situation of the device. These parameters are: noise, privacy, disturbance and activity. Details are described in Figure 12 on page 58.

Different methods can be used to detect the values. The parameter *noise* identifies the loudness of the surrounding environment and can be retrieved using the microphone of the PDA. The microphone takes an audio sample, which is the input of a function that calculates the root-mean-square (RMS), a measure of loudness. This RMS value is

compared with reference values representing several noise levels so that it can be mapped to a certain situation. For each reference situation a minimum and maximum value has been set. The reference situations are for example: a quiet environment like an office, a little bit louder like an office room where a few other persons are working, a louder room like a meeting room during a talk or a discussion and the loudest environment like a bar in the evening. Each reference value is assigned to a situation representing an environment of certain loudness from silent to loudest. These reference values are depending on the used device and therefore have to be adjusted if a different device will be used. The reference values can be retrieved using the root-mean-square function too. The sample code uses the .NET Compact Framework and can be downloaded from the Article of MSDN Magazine [8].

The situation parameter *privacy* represents if the user wants to keep his privacy, which means that using voice mode is most likely unwanted. This can be determined by his current location. If the user is at a private location like his home or his office he may prefer voice mode but if he is at a public location like a meeting room or in the subway he may want to keep his privacy and prefers text output and tapping for giving input. On the basis of the numbers of people with the same current location as the user the parameter *disturbance* can be set. If there are other people around him they may be disturbed when the user talks to the application respectively the Teamscheduler uses voice as output method. Whereas if the user is alone nobody can be bothered by using voice. The fourth parameter is *activity* where calendar information of the user and his current location is used to retrieve if the application has the primary or secondary attention focus of the user. If the application has just the secondary attention of the user because he is driving a car then using voice may be more useful and safer.

The situation parameters are describing the device context and are used to decide which the best suitable modality is: voice or text. Currently each parameter can have just the value 0 or 1 where 0 means text and 1 means voice is the best modality. In future extension a range of values may be used for each parameter so that the combination of them gives more possibilities for deciding if voice or text is the most suitable modality. At present the teamscheduler.proxy bundle reads out the device context parameters and if all parameters are set to 1 then it determines voice as the best suitable modality.

The main application on the client device is the teamscheduler.proxy bundle. It is implemented as an HTTP Servlet also. There are a few resources which are registered when the service is added. These are a stylesheet .css file and the static JSGF grammar files for date, menu and skip phrases. All other grammar files are dynamic and generated at runtime. The Teamscheduler servlet handles HTTP requests with GET and POST. Where GET request parameters are empty, list or id. If the parameter is empty

then it shows the menu and if the parameter is list the list of all tasks is shown. The parameter id is used to retrieve detailed data of a certain task so that a detailed view of a task can be shown. The POST request parameters are id or save. If the parameter id is empty, then an empty form for editing a new task is requested otherwise if the id is not empty the form is filled in with the data of the corresponding task. If the button save in the form was clicked, then the POST request parameter save is sent.

The taskHistory array holds the ids of five lastly used tasks and is used to achieve context-awareness. The id of a certain task is stored in this array whenever the user requests the details of a task or edits the parameters of a task. After a new task has been entered the id is also stored in the taskHistory array. If an empty meeting form is requested this history context array is used to set the task list box. For now we have just implemented this task-awareness, but this can further be extended with more context-awareness. For example the project list box can be adjusted in a similar way, so that it list projects the user has viewed or edited lately. The list of task assignees and meeting invitees can be adapted with using task and project information of the members to show just persons who are working at the same projects and tasks. Furthermore if we have a device with GPS the current location of the user can be used to retrieve just meeting locations that are near to the user and to identify if the user is far away or on the way so that the form of the meeting may be a conference call.

4.4.6. Deployment

We used three different environment configurations:

1. a PC with Windows XP running all parts of the team scheduling scenario,
2. the same PC for the server application and the database but the client application running in a Windows Mobile 5 Emulator Image for Pocket PC in Microsoft Device Emulator [10] and
3. the client application running on a HP iPAQ hx4700 PDA with Windows Mobile 2003 installed and again the PC with Windows XP hosts the database and the server application.

Running all parts on one Windows XP PC was used just for development, debugging and testing purpose because it is easier to handle and to trace all debugging messages and output parallel at runtime. On the Windows XP PC we used the Java SE Runtime Environment 1.6 and started Knopflerfish in the standard configuration. This includes the Desktop bundle which is a graphical overview of the OSGi framework so that managing bundles is easier. The multimodal browser NetFront looks just a little bit different than

the version for the PDA.

The deployment on the iPAQ and the Device Emulator is nearly the same. We specified in the Device Emulator a certain folder on the host as the shared folder for exchanging data between the host system and the Windows Mobile Emulator. For exchanging data with the iPAQ we used ActiveSync. As Java Runtime Environment we needed a Java Micro Edition which is especially for mobile devices. The J9 VM is a J2ME compliant that allows Java applications to run on small devices. We installed the J9 VM included in the trial version of WEME for Windows Mobile 2003. If the installer on the PC can establish an ActiveSync connection between the mobile device and the PC then it is installed on the mobile device too automatically. Manually it can be installed by copying the .cab file to the mobile device and executing it there. The .cab file includes the Device Installer of WEME. Then we installed the open source OSGi framework Knopflerfish with a minimal set of bundles to save memory and disk space. Additionally included are the bundles HTTP-Server and JSDK lib and of course our own implemented bundles teamscheduler.proxy and teamscheduler.devicecontext.

Instead of executing Knopflerfish and J9 from command line as on the Windows XP PC we need a shortcut file to run them on small mobile devices like the iPAQ or the Device Emulator. With a shortcut we can execute a program with parameters on Windows Mobile because it does not have a command line. A shortcut file has the extension .lnk and contains the command to be executed. It consists of a number that describes the length of the command line, followed by the # character and the actual command to be executed. In our case it looks like this:

```
255#"\Program Files\J9\PPRO11\bin\j9.exe" "-jcl:ppro11" "-cp"
"\Program Files\kf\framework.jar" org.knopflerfish.framework.Main
```

The command must be one single line and must not be more than 255 bytes long. The command with parameters can then be executed on the mobile device with the shortcut file. Figure 25 shows the output of the J9 console after calling the .lnk file that launches the Knopflerfish framework with the bundles of the team scheduling client application.

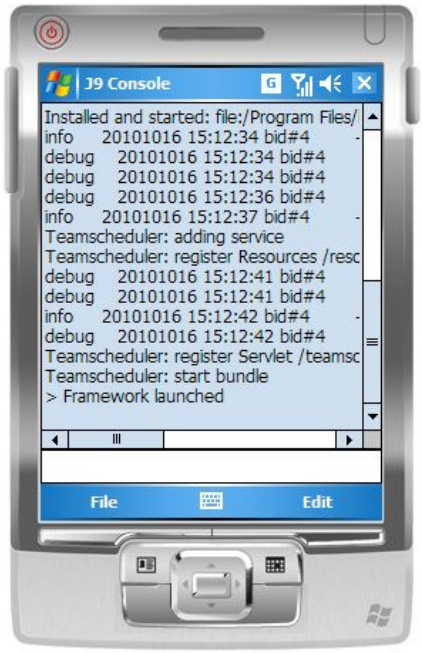

Figure 25: J9 console with Knopflerfish framework launched

The last step was to install the Multimodal Browser. We installed the NetFront Browser which is included in the Multimodal Tools package. There exists a special PDA version where the Device Installer is in a .cab file, which is transferred to the iPAQ or Device Emulator and installed by executing this cab file.

4.4.7. Deployment Problems

Unfortunately we had troubles with deploying the team scheduling client application on mobile devices, on an emulator as well as on the PDA. We have not yet found an emulator which runs all our team scheduling client components. We tried a few Windows Mobile 2003 Emulator Images for Pocket PCs that were available in Microsoft Visual Studio 2003. But in none of them we got any version of the IBM J9 JVM running. Furthermore we had troubles with memory and disc space, it was not possible to enlarge these parameters even though the emulator had this feature. Then we tried the Standalone Device Emulator with a Windows Mobile 5.0 Image. Therefore we used a trial version of WEME 6.1 which contained the J9 with CDC Foundation 1.1 and Personal Profile 1.1 for PDAs running Windows Mobile 5.0. The LIB and BIN directories of the J9 had to be copied manually to the emulator using a shared folder that could be defined in the Device Emulator. We got the J9 JVM running and also the open source OSGi Knopflerfish in a minimal configuration and with our team scheduling bundles. But then we had troubles with the NetFront multimodal browser. Most of the time the

NetFront browser did not perform correctly, sometimes it even did not open just a simple HTML file or it did not respond. But sometimes the browser rendered HTML and X+V files correctly. Unfortunately it was irreproducible under which configurations and requirements the NetFront browser will render files correctly without hanging up. Furthermore it was not possible to use a microphone with these emulators.

On the PDA the NetFront Browser worked correctly, we could use voice to fill in the multimodal examples which were installed with the multimodal browser. We had no troubles running J9 JVM and the Knopflerfish OSGi with the bundles of our team scheduling application. But running both the JVM with Knopflerfish and the NetFront browser caused problems because of limited memory space. Additionally we had difficulties setting up the WLAN configuration of the PDA but this is beyond the scope of this paper.

4.5. Evaluation

We evaluated the usefulness and usability of the team scheduling application in a user study. We asked different users with varying experiences in using mobile devices to participate. A few were novices and had never used a PDA while some had used Pocket PCs before, for navigation or calendar applications and a few were nearly experts in using mobile devices of various types. But none of them had ever used a multimodal application. First of all we gave them a little demonstration to explain the features of the application and how they can use the voice modality. Moreover each user was familiarized with the allowed grammar rules and utterances. Additionally they got some instructions about the tests they were expected to carry out: The user should create some new tasks and then a new meeting dealing with one of the formerly entered tasks. He should use voice or text or both as he may prefer.

A test of the team scheduling application should look like the following. After the user has started the application and has logged in, he first sees the Teamscheduler Menu (Figure 26) where he can decide what he wants to do: enter a new task or a new meeting or view a list of all existing tasks. He may use the voice button of the multimodal browser to give his voice input or use the pen and tap on the screen to select an item of the menu.

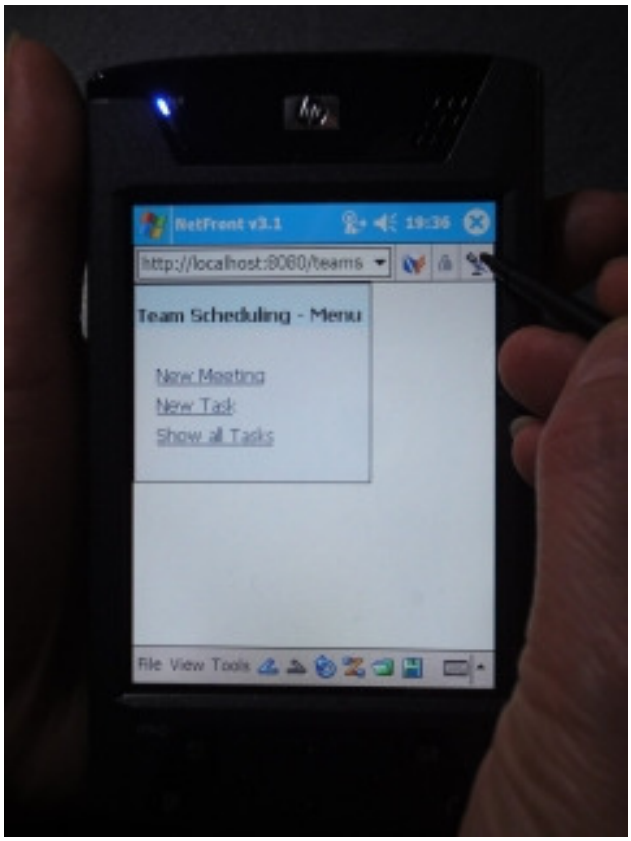

Figure 26: Teamscheduler – Menu

In the next test case the user should enter some new tasks. He can fill in the task form, see Figure 27, using voice or pen. He may use voice for all fields except the description of the task which is a text field with no grammar rules behind. If he uses voice, the dialog flow goes through the form from top to bottom while the active field is marked with a red border. If the speech input matches the grammar rules the field is filled in with the given input and the dialog flow proceeds with the next field. Additionally he may fill in the whole form with almost just one sentence. Furthermore he has the possibility to use help commands to hear all possible input values. If the speech input does not match the grammar rules he is asked to give the input again. At the end of the form the voice interface replays all given input and asks the user if this input is correct, then the user can do corrections or confirm the given input which is saved then.

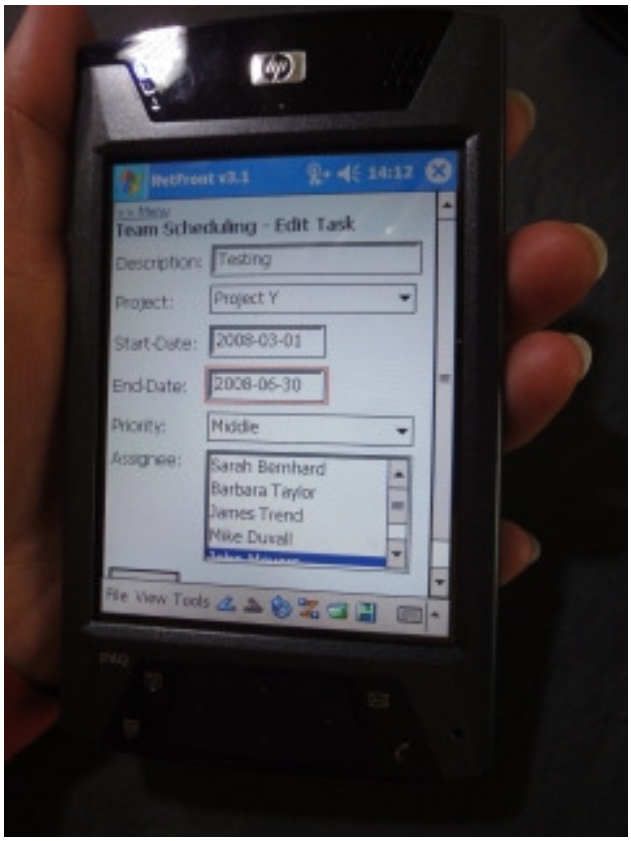

Figure 27: Teamscheduler – Edit Task

After having saved a task the application shows the list of all tasks, see Figure 28. On this graphical view the user can again select a task using voice or pen. If the user uses the voice button to give speech input the microphone icon of the multimodal browser displays a green circle instead of a red cross. This means the microphone is active and listening to input. There are three listening modes which can be configured in the multimodal browser, see page 22 for details.

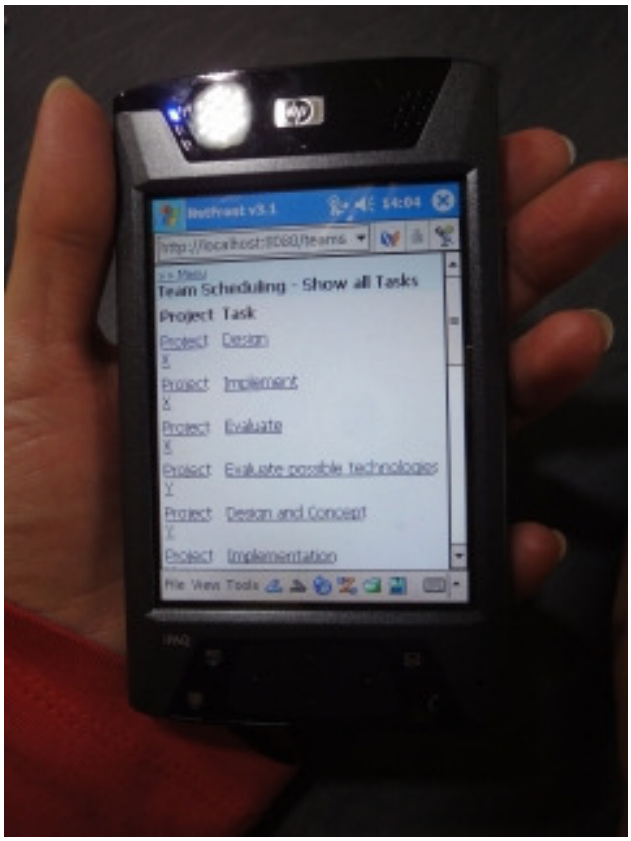

Figure 28: Teamscheduler – Show all Tasks

If the user selected a task out of the list of all tasks, the application shows the details of this task, see Figure 29. The parameters of the task are shown not only graphically but also played by voice. The user has the possibility to change the values of the task by giving a voice command like “change” or “edit”. Here he has as well the possibility to listen to all available speech commands when he uses the voice command “help”.

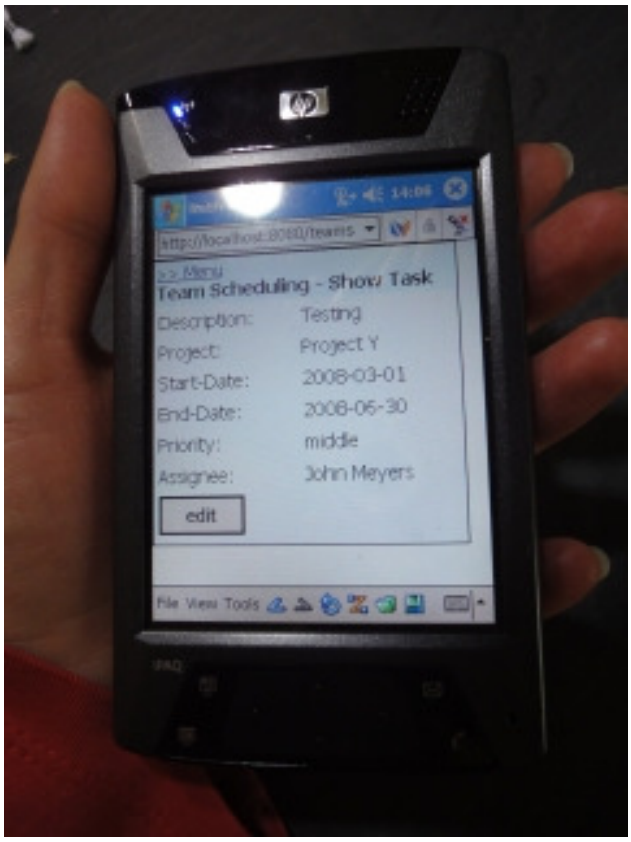

Figure 29: Teamscheduler – Show Task

After editing a few tasks the user was expected to edit a new meeting dealing with one of the tasks he had entered before. The new meeting form, see Figure 30, is adapted with context information. The application saved recently edited tasks and used these parameters to fill in the options of the list boxes of project and task and the option values of the multiple selection list box of invitees. Again the user can use his preferred input modality, voice or pen, to fill in the form. If he uses voice he has again the possibility to use just one sentence to specify all parameters of the meeting. As in the task edit form he can get help about the voice commands he may use. And if the speech recognition does not understand the given voice input he is asked to repeat his input as in the task form. Again all given input is repeated at the end of the form and the user can make corrections or confirm the speech input.

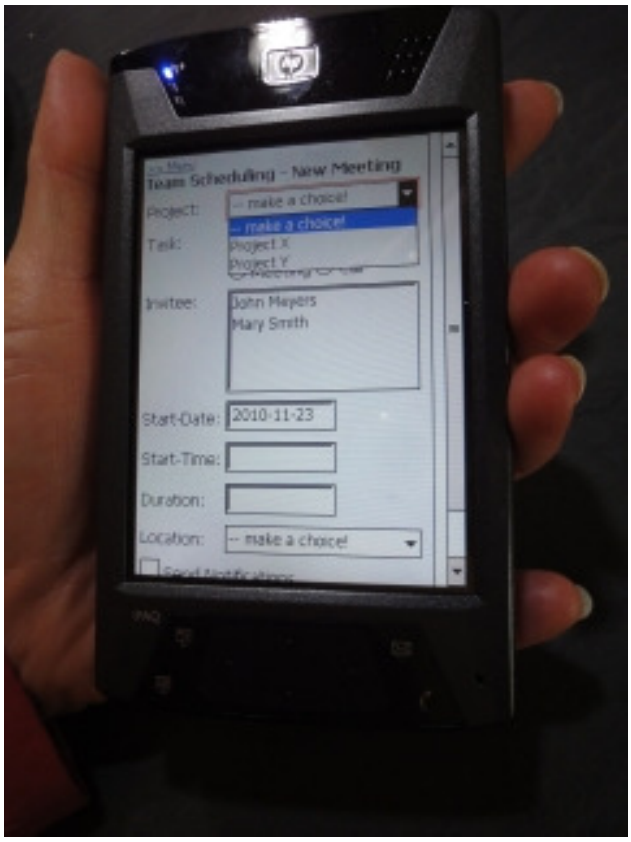

Figure 30: Teamscheduler – New Meeting

We observed the users while they were interacting with the application and they were interviewed afterwards. The user tests showed an overall good acceptance for the multimodal team scheduling application, but additionally outlined some limitations and problems. All users had sometimes difficulties with the speech recognition, few users had even great troubles being understood by the application. Despite these troubles with speech recognition accuracy most of the users preferred using voice. Mostly the users combined speech and pen inputs throughout the application. For selecting an element out of a list like the menu of the team scheduling application or the list of tasks most participants of our study used the pen instead of voice. Whereas filling in the form of a task or a meeting were mostly done by voice.

After completing the tasks the participants of the study were asked to express their satisfaction regarding the multimodal interaction in terms of usefulness and of speech recognition accuracy. In spite of some difficulties with the accuracy of the speech recognition nearly all participants declared that they prefer using speech. Almost 90 percent of the users liked using speech instead of pen and they considered that the voice interaction is faster than the pen-based. Only for simple interactions like selecting an item from the menu the pen is suggested as more suitable. All users liked that they

had the possibility to choose their preferred input modality. All users found especially the help messages and the error handling useful when speech recognition failed. Especially before the form is submitted when the application repeats the whole input and the user can correct it once more was outlined as helpful. Additionally filling out the whole form at once with one sentence was judged useful by the testers.

Also we asked the participants of our user study if they prefer using the voice part or the graphical part of the application, for output and for giving input. Only one user preferred using just the visual interface for input and output. He found it embarrassing to talk to a PDA. Additionally he outlined that he would prefer using mouse and keyboard because he feels more comfortable with these traditional devices. While one user suggested that he would like to use only voice mode. He stated that he is often on the way, for example, driving a car, and then he needs his hands and eyes free and would like to handle such a team scheduling application with his voice. While all other preferred using text and voice in combination. Mostly the users preferred multimodal interaction using speech and thought that they were more efficient with this kind of interface. Especially the visual feedback like output on the GUI was outlined as important and helpful.

Our evaluation took place in an office environment where it was rather very silent so that we had ideal conditions for accurate voice recognition. Further evaluation should be done in other environments with various noise levels to prove if ambient noise levels are a problem. Moreover this is needed to test if the application uses the adequate modality. The application uses the context of the device to decide which the best modality for the current situation is, voice or text. In a further user study it should be tested if the participants follow this suggestion. But for now we asked the users if they would prefer using voice in other situations too. Just one user requested pen-only interaction in a situation where other people are around him because he might not necessarily want others to know what he is doing and furthermore he did not want to disturb others. While most of the users would not mind if other people heard them talking and listening to a mobile device, because it is the same as using a mobile phone.

5. Conclusion and Outlook

This work deals with the problem that mobile collaboration systems running on mobile devices like PDAs have to overcome the limitations of small devices like screen resolution and unusual input facilities. Furthermore mobile users are often on the way and therefore prefer hands- and eyes-free interaction with the system. Using speech as input mode would improve such systems but having speech as the only input and output modality is not practical. There may be situations where the user would prefer using a stylus to give text input to keep his privacy or because of ambient noise. With a multimodal application the user has the possibility to interact with the interface using his preferred mode or the most appropriate mode for the current situation. For example he has the choice to use speech as the easier mode for providing input or he may prefer an input device such as a stylus, keypad or mouse to protect privacy or due to surrounding noise. Furthermore he may use voice, keyboard, mouse and pen in a synergistic way. The output can be audio as spoken prompts and playback, using text-to-speech or visually on graphical displays. Additionally a multimodal application simplifies using speech as an input modality by allowing the user to specify multiple pieces of the input with just one spoken sentence.

Users of mobile collaboration systems handle lots of information. Therefore speech output can be pretty long where the user has to listen a long time before he gets the data needed. And large data leads to large grammar rules which interferes speech recognition rates. For mobile workers it is important to get the appropriate information for a current place and time. Making speech output context sensitive will prevent overloading the user with unwanted information and context information can be used to improve speech grammars. Furthermore information of the current location and situation of the user can be used to find the adequate modality. In a noisy environment or when other people are around the user the best input and output mode will be the GUI using pen and display. Whereas speech will be the better modality for input and output when the user is driving a car and needs his hands and eyes free and cannot fully concentrate on the GUI. Enhancing a mobile collaboration system with multimodal interaction and additionally using context-based data for speech input and output will improve such a system considerably.

Within this thesis we implemented a team scheduling application for a PDA to show how context-based multimodal interaction can be used to improve mobile collaboration. With our team scheduling application a project manager can schedule meetings and tasks for

his team. A common Web application is enhanced with a speech-enabled Web form that gives the user the possibility to fill out the form by giving voice commands or combining text and voice input. The application consists of a server part and a client part. The teamserver application holds the database and serves all data requests. The client part runs on an HP iPAQ and consists of a multimodal browser and a Java application that is implemented as a bundle running in the Knopflerfish OSGi framework. Client and server communicate with SOAP messages. We used XHTML+Voice as the markup language to enable voice with VoiceXML snippets in an XHTML Web form. On the PDA we used the multimodal browser NetFront that can interact with multimodal applications written in XHTML+Voice. The browser handles the user input and the system output, speech and text. NetFront is enhanced with IBMs speech recognition and synthesis technology.

Most of the time speech recognition has functioned accurate and brought promising results. But the system configuration of the used mobile device, an HP iPAQ, has been insufficient. Limited memory space has led to troubles running both the JVM with Knopflerfish including our team scheduling client application and the multimodal browser NetFront. Sometimes it worked, but sometimes starting one application forced closing the other application because of the limited memory space. The decision when speech will be the best modality for the current situation depends on a few context parameters. We used just a simple key-value model where a value of 0 means text and a value of 1 means voice is the best mode. But just if all parameters are set to 1 speech-output and speech-input will be used. If just one parameter has the value 0 text is used as input and output mode. It would be better to use a range of values instead of just 0 or 1 and to decide which parameter has to be used strict and which can have a wider range to accept voice-output. In a further user study taking place at different places of diverse noise levels could be tested if the participants follow the suggestion of the best modality to find out which parameter can have what values.

Future extensions should implement additional context-awareness. At this time the team scheduling application uses just information about recently used tasks to adapt the list box of tasks. The project list box can be adjusted in a similar way, so that it lists projects the user has viewed or edited lately. The list of task assignees and meeting invitees can be adapted with using task and project information of the members to show just persons who are working in the same projects and tasks. Furthermore if we have a device with GPS the current location of the user can be used to retrieve just meeting locations that are close to the user and to identify if the user is far away or on the way so that the form of the meeting might be a conference call.

In our future work it would be interesting and make sense to enable the client application to run on a smart phone. Over the past years smart phones have replaced mobile

devices like the HP iPAQ in the area of PDAs. They have considerably higher performance and memory space. Additionally research has to be done in the area of speech technology, because the multimodal browser NetFront does not exist anymore and above all is not advanced anymore. The other XHTML+Voice enabled browser Opera does not support voice in the version for mobile devices. So our future work has to be done with other speech engines and maybe with another multimodal language for example with EMMA. This will be additionally advantageous because EMMA supports other input modes too. We currently support voice and text as possible input modalities, other modalities especially handwriting recognition will be an important additional input mode for the future. Pen or stylus can be used for handwriting, drawing and making notations.

We think that speech is the most convenient and natural way of interacting with other people and with the computer too. Multimodal interaction, a combination of speech and other input modalities, will be an important issue in the future. This work shows that there are other possible application domains besides navigation devices and reading email or text messages. In the future there might be many applications supporting multimodal interaction in challenging situations.

List of Figures

Figure 1: VoiceXML architecture 11
Figure 2: SALT architecture 15
Figure 3: Speech Server 16
Figure 4: A speech enabled Web form with X+V code segments 20
Figure 5: Input components of a multimodal system generating EMMA 23
Figure 6: Deployment of SALT applications using Visual Studio with SASDK 29
Figure 7: The Multimodal Toolkit for developing X+V applications 30
Figure 8: Basic steps to create an X+V application 31
Figure 9: An OpenVXML voice application 33
Figure 10: Basic scenario – team scheduling 54
Figure 11: Team scheduling with context-based multimodal interaction 56
Figure 12: Diagram of context objects 58
Figure 13: Flowchart of finding the best modality with the situation parameters 60
Figure 14: Use case 61
Figure 15: Basic component diagram 62
Figure 16: Sequence diagram 64
Figure 17: Graphical User Interface: Start page – Menu 65
Figure 18: Graphical User Interface – Tasks 66
Figure 19: Graphical User Interface – Meetings 67
Figure 20: Graphical User Interface – Configuration 67
Figure 21: Software architecture 69
Figure 22: Detailed component diagram 70
Figure 23: Voice preference of NetFront multimodal browser 73
Figure 24: Database tables and their relations 75
Figure 25: J9 console with Knopflerfish framework launched 79
Figure 26: Teamscheduler – Menu 82
Figure 27: Teamscheduler – Edit Task 83
Figure 28: Teamscheduler – Show all Tasks 84
Figure 29: Teamscheduler – Show Task 85
Figure 30: Teamscheduler – New Meeting 86

Bibliography

[1] Ancona, M.; Dodero, G.; Minuto, F.; Guida, M. and Gianuzzi, V.: Mobile computing in a hospital: the WARD-IN-HAND project. In Proceedings of the 2000 ACM Symposium on Applied Computing - Volume 2 (Como, Italy). J. Carroll, E. Daminani, H. Haddad, and D. Oppenheim, Eds. SAC '00. ACM Press, New York, NY, 554-556. DOI= http://doi.acm.org/10.1145/338407.338419

[2] Andolina, S.; Santangelo, A.; Cannella, M.; Gentile, A.; Agnello, F.; Villa, B.: Multimodal virtual navigation of a cultural heritage site: The medieval ceiling of Steri in Palermo. Human System Interactions, 2009. HSI '09. 2nd Conference on 21-23 May 2009 Page(s):562 – 567. Digital Object Identifier 10.1109/HSI.2009.5091039

[3] Bardram, J. E. and Hansen, T. R. : The AWARE architecture: supporting context-mediated social awareness in mobile cooperation. In Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work (Chicago, Illinois, USA, November 06 - 10, 2004). CSCW '04. ACM Press, New York, NY, 192-201. DOI= http://doi.acm.org/10.1145/1031607.1031639

[4] Belotti, R.; Decurtins, C.; Norrie, M. C.; Signer, B. and Vukelja, L: Experimental platform for mobile information systems. In Proceedings of the 11th Annual international Conference on Mobile Computing and Networking (Cologne, Germany, August 28 - September 02, 2005). MobiCom '05. ACM Press, New York, NY, 258-269. DOI= http://doi.acm.org/10.1145/1080829.1080856

[5] Bolelli, L.; Cai, G.; Wang, H.,;Mortazavi, B.; Rauschert, I.; Fuhrmann, S.; Sharma, R. and MacEachren, A.: Multimodal interaction for distributed collaboration. In Proceedings of the 6th international Conference on Multimodal interfaces (State College, PA, USA, October 13 - 15, 2004). ICMI '04. ACM Press, New York, NY, 327-328. DOI= http://doi.acm.org/10.1145/1027933.1027990

[6] Bouyer, A.; Chuffart, F.; Courval, L: The Design of a Multimodal Platform: Experimentation of Record & Replay. Advances in Computer-Human Interactions, 2009. ACHI '09. Second International Conferences on 1-7 Feb. 2009 Page(s):1 - 6 Digital Object Identifier 10.1109/ACHI.2009.27

[7] Bühler, D.; Minker, W.: Mobile Multimodality – Design and Development of the SmartKom Companion. In Journal of Sol-Gel Science and Technology, Volume 8, Issue 2, Jun 2005, Pages 193 - 202, DOI 10.1007/s10971-005-2170-y, URL http://dx.doi.org/10.1007/s10971-005-2170-y

[8] Chris Mitchell, Anpassen der Klingellautstärke an Umgebungsgeräusche, 1.8.2010, http://msdn.microsoft.com/de-de/magazine/cc163341.aspx

[9] Deng, L.; Wang, Y.; Wang, K.; Acero, A.; Hon, H.; Droppo, J. ; Boulis, C. ; Mahajan, M.; Huang, X.D.: Speech and Language Processing for Multimodal Human-Computer Interaction. The Journal of VLSI Signal Processing, Volume 36, Issue 2 - 3, Feb 2004, Pages 161 - 187, DOI 10.1023/B:VLSI.0000015095.19623.73, URL http://dx.doi.org/10.1023/B:VLSI.0000015095.19623.73

[10] Device Emulator wth Windows Mobile 5.0, 1.8.2010, http://www.microsoft.com/downloads/en/details.aspx?FamilyId=C62D54A5-183A-4A1E-A7E2-CC500ED1F19A&displaylang=en

[11] Di Fabbrizio, G.; Okken, T. and Wilpon, J. G.: A speech mashup framework for multimodal mobile services. In Proceedings of the 2009 international Conference on Multimodal interfaces (Cambridge, Massachusetts, USA, November 02 - 04, 2009). ICMI-MLMI '09. ACM, New York, NY, 71-78. DOI= http://doi.acm.org/10.1145/1647314.1647329

[12] Dorn, C.; Schall, D.; Gombotz, R.; Dustdar, S.: A View-Based Analysis of Distributed and Mobile Teams. WETICE 2007: 198-203

[13] Dorn, C.;Schall, D.; and Dustdar S.: Granular context in collaborative mobile environments. In International Workshop on Context-Aware Mobile Systems (CAMS'06), Montpellier, France. Springer.

[14] Du, H.; Crestani, F.: Spoken versus Written Queries for Mobile Information Access. Lecture Notes in Computer Science, Volume 2954, Jan 2004, Pages 67 – 78

[15] Dusan, S.; Gadbois, G.J.; Flanagan, J.: Multimodal interaction on PDA's integrating speech and pen inputs. Eurospeech (2003).

[16] Dustdar, S.: Caramba - A Process-Aware Collaboration System Supporting Ad hoc and Collaborative Processes in Virtual Teams. In Distributed and Parallel Databases, vol. 15, no. 1, pp. 45–66, January 2004.

[17] Eclipse Voice Tools Project, 29.8.2010, http://eclipse.org/vtp/

[18] Eclipse, 1.9.2010, http://www.eclipse.org

[19] EMMA, 20.5.2010, http://www.w3.org/TR/emma/

[20] Esbjörnsson, M.: From ethnography on infrastructure management to initial user feedback on PlaceMemo. In Personal Ubiquitous Comput. 10, 4 (Mar. 2006), 195-204. DOI= http://dx.doi.org/10.1007/s00779-005-0041-8

[21] eVALUES, 1. 7. 2008, http://evalues.moviquity.com/

[22] Ghiani, G.; Paterno, F.; Santoro, C.; Spano, L. D.: UbiCicero: A location-aware, multi-device museum guide. Interacting with Computers, Volume 21, Issue 4,

August 2009, Pages 288-303, ISSN 0953-5438, DOI: 10.1016/j.intcom.2009.06.001. http://www.sciencedirect.com/science/article/B6V0D-4WGBFKG-1/2/36207800e2ab702fabc2d48e9a1121fd

[23] Goose, S.; Newman, M.; Schmidt, C.; Hue, L.: Enhancing Web accessibility via the Vox Portal and a Web-hosted dynamic HTMLVoxML converter. Computer Networks 33(1-6): 583-592 (2000)

[24] Goose, S.; Wanning, H.; Schneider, G.: Mobile Reality: A PDA-Based Multimodal Framework Synchronizing a Hybrid Tracking Solution with 3D Graphics and Location-Sensitive Speech Interaction. In Lecture Notes in Computer Science, Volume 2498, Jan 2002, Page 33

[25] GPS-Wegbegleitung für die Bundesgartenschau, 1. 7. 2008, http://www.heise.de/newsticker/meldung/58648

[26] IBM Multimodal, 1. 7. 2008, http://www-306.ibm.com/software/pervasive/multimodal/

[27] IBM WebSphere development tools, 1. 7. 2008, http://www.ibm.com/developerworks/websphere/zones/studio/

[28] JSGF Specification, 1.7.2010, http://www.w3.org/TR/jsgf/

[29] Kadous, M. W.; Sammut, C. : InCA: A Mobile Conversational Agent. Lecture Notes in Computer Science, Volume 3157, Jan 2004, Pages 644 – 653

[30] Kane, S. K.; Jayant, C.; Wobbrock, J. O. and Ladner, R. E.: Freedom to roam: a study of mobile device adoption and accessibility for people with visual and motor disabilities. In Proceedings of the 11th international ACM SIGACCESS Conference on Computers and Accessibility (Pittsburgh, Pennsylvania, USA, October 25 - 28, 2009). Assets '09. ACM, New York, NY, 115-122. DOI= http://doi.acm.org/10.1145/1639642.1639663

[31] Kernchen, R.; Boda, P.P.; Moessner, K.; Mrohs, B.; Boussard, M.; Giuliani, G.: Multimodal user interfaces for context-aware mobile applications. Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005. IEEE 16th International Symposium on Volume 4, 11-14 Sept. 2005 Page(s):2268 - 2273 Vol. 4 Digital Object Identifier 10.1109/PIMRC.2005.1651849

[32] Kleindienst, J.; Macek, T.; Seredi, L.; Sedivy, J.: Interaction framework for home environment using speech and vision. Image and Vision Computing, Volume 25, Issue 12, The age of human computer interaction, 3 December 2007, Pages 1836-1847, ISSN 0262-8856, DOI: 10.1016/j.imavis.2006.04.026.

http://www.sciencedirect.com/science/article/B6V09-4M2WP0D-2/2/04fbbea472fa82a7a0371bb62ce5e129

[33] Knopflerfish, 1.7.2008, http://www.knopflerfish.org/

[34] Mamykina, L.; Goose, S.; Hedqvist, D.; Beard, D. V.: CareView: analyzing nursing narratives for temporal trends. CHI Extended Abstracts 2004: 1147-1150

[35] Mantoro, T. and Johnson, C.: Location history in a low-cost context awareness environment. In Proceedings of the Australasian information Security Workshop Conference on ACSW Frontiers 2003 - Volume 21 (Adelaide, Australia). C. Johnson, P. Montague, and C. Steketee, Eds. ACM International Conference Proceeding Series, vol. 34. Australian Computer Society, Darlinghurst, Australia, 153-158.

[36] Marmasse, N.; Schmandt, C.: Location-Aware Information Delivery with ComMotion. Lecture Notes in Computer Science, Volume 1927, Jan 2000, Page 157

[37] Marx, M. and Schmandt, C.: CLUES: dynamic personalized message filtering. In Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work (Boston, Massachusetts, United States, November 16 - 20, 1996). M. S. Ackerman, Ed. CSCW '96. ACM Press, New York, NY, 113-121. DOI= http://doi.acm.org/10.1145/240080.240230

[38] Marx, M. and Schmandt, C.: MailCall: message presentation and navigation in a nonvisual environment. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Common Ground (Vancouver, British Columbia, Canada, April 13 - 18, 1996). M. J. Tauber, Ed. CHI '96. ACM Press, New York, NY, 165-172. DOI= http://doi.acm.org/10.1145/238386.238467

[39] Meng, H.; Ching, P. C.,;Chan, S. F.; Wong, Y. F. and Chan, C. C.: ISIS: an adaptive, trilingual conversational system with interleaving interaction and delegation dialogs. In ACM Trans. Comput.-Hum. Interact. 11, 3 (Sep. 2004), 268-299. DOI= http://doi.acm.org/10.1145/1017494.1017497

[40] Microsoft Speech Application SDK, MSDN Library, 1. 7. 2008, http://msdn.microsoft.com/en-us/library/ms986944

[41] Microsoft Speech Server, 1. 7. 2008, http://www.microsoft.com/speech/default.mspx

[42] MONA - Mobile Multimodal Next Generation Applications, 20. 5. 2010, http://mona.ftw.at

[43] Multimodal Tools Project for Eclipse, November 2006, http://www.alphaworks.ibm.com/tech/mmtp

[44] Muñoz, M. A.; Rodríguez, M.; Favela, J.; Martinez-Garcia, A. I. and González, V. M.: Context-Aware Mobile Communication in Hospitals. In Computer 36, 9 (Sep. 2003), 38-46. DOI= http://dx.doi.org/10.1109/MC.2003.1231193

[45] NetFront Browser for Pocket PC, Access Company, 1.11.2006, http://www.access-us-inc.com/Prod_NetFront_nf_xhtml.html

[46] O'Neill, E.; Kaenampornpan, M.; Kostakos, V.; Warr, A.; Woodgate, D.: Can we do without GUIs? Gesture and speech interaction with a patient information system. Personal and Ubiquitous Computing, Jan 2006, Pages 1 - 15, DOI 10.1007/s00779-005-0048-1, URL http://dx.doi.org/10.1007/s00779-005-0048-1

[47] Opera Browser V 7.55, 1.11.2006, http://www.opera.com/products/verticals/multimodal/index.dml

[48] OSGi Alliance, 1.11.2010, http://www.osgi.org

[49] Paek, T.; Thiesson, B.; Ju, Y. and Lee, B.: Search Vox: leveraging multimodal refinement and partial knowledge for mobile voice search. In Proceedings of the 21st Annual ACM Symposium on User interface Software and Technology (Monterey, CA, USA, October 19 - 22, 2008). UIST '08. ACM, New York, NY, 141-150. DOI= http://doi.acm.org/10.1145/1449715.1449738

[50] Perakakis, M.; Potamianos, A.: A Study in Efficiency and Modality Usage in Multimodal Form Filling Systems. Audio, Speech, and Language Processing, IEEE Transactions on Volume 16, Issue 6, Aug. 2008 Page(s):1194 – 1206. Digital Object Identifier 10.1109/TASL.2008.2001389

[51] Pham, Thai-Lai; Schneider, Georg; Goose, Stuart: A situated computing framework for mobile and ubiquitous multimedia access using small screen and composite devices. ACM Multimedia 2000: 323-331

[52] Pham, Thai-Lai; Schneider, Georg; Goose, Stuart: Exploiting Location-Based Composite Devices to Support and Facilitate Situated Ubiquitous Computing. HUC 2000: 143-156

[53] Pilotprojekt am Klinikum Saarbrücken, 1. 7. 2008, http://www.heise.de/newsticker/meldung/58777

[54] Pokraev, S.; Koolwaaij, J.; van Setten, M.; Broens, T.; Costa, P.D.; Wibbels, M.; Ebben, P.; Strating, P.: Service platform for rapid development and deployment of context-aware, mobile applications, Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on , vol., no.pp.- 646, 11-15 July 2005

[55] Pospischil, G.; Umlauft, M. and Michlmayr, E: Designing LoL@, a Mobile Tourist Guide for UMTS. In Proceedings of the 4th international Symposium on Mobile

Human-Computer interaction (September 18 - 20, 2002). F. Paternò, Ed. Lecture Notes In Computer Science, vol. 2411. Springer-Verlag, London, 140-154.

[56] Rea, S. Morgan: Building Intelligent .NET Applications, Addison-Wesley, ISBN 0-321-24626-8

[57] Reithinger, N., Alexandersson, J., Becker, T., Blocher, A., Engel, R., Löckelt, M., Müller, J., Pfleger, N., Poller, P., Streit, M., and Tschernomas, V. 2003. SmartKom: adaptive and flexible multimodal access to multiple applications. In Proceedings of the 5th international Conference on Multimodal interfaces (Vancouver, British Columbia, Canada, November 05 - 07, 2003). ICMI '03. ACM Press, New York, NY, 101-108. DOI= http://doi.acm.org/10.1145/958432.958454

[58] SALT, 1. 7. 2008, http://www.saltforum.org/

[59] Sawhney, N. and Schmandt, C.: Nomadic radio: speech and audio interaction for contextual messaging in nomadic environments. ACM Trans. Comput.-Hum. Interact. 7, 3 (Sep. 2000), 353-383. DOI= http://doi.acm.org/10.1145/355324.355327

[60] Schall, D.; Dorn, C.; Dustdar, S. (2008): Context-aware Mobile Computing. In Encyclopedia of Wireless and Mobile Communications, CRC Press

[61] Schall, D.; Gombotz, R.; Dorn, C.; Dustdar, S.: Human Interactions in Dynamic Environments through Mobile Web Services. ICWS 2007: 912-919

[62] Schmandt, C.; Lee, K. H.; Kim, J. and Ackerman, M.: Impromptu: managing networked audio applications for mobile users. In Proceedings of the 2nd international Conference on Mobile Systems, Applications, and Services (Boston, MA, USA, June 06 - 09, 2004). MobiSys '04. ACM Press, New York, NY, 59-69. DOI= http://doi.acm.org/10.1145/990064.990074

[63] Schmandt, C.; Marmasse, N.; Marti, S.; Sawhney, N.; Wheeler, S.: Everywhere messaging. IBM Systems Journal 2000; 39(3/4)

[64] Schneider, G.; Hoymann, C.; Goose, S.: Adhoc Personal Ubiquitous Multimedia Services Via Upnp. ICME 2001

[65] Semantic Interpretation for Speech Recognition, 1.7.2008, http://www.w3.org/TR/semantic-interpretation/

[66] Semantic Markup Language Reference, 1.8.2010, http://msdn.microsoft.com/en-us/library/ff394926.aspx

[67] SHARE Project, 20.5.2010, http://www.ist-share.org

[68] Speech Recognition Grammar Specification, 1.7.2008, http://www.w3.org/TR/speech-grammar/

[69] Speech Synthesis Markup Language (SSML), 1.7.2008, http://www.w3.org/TR/speech-synthesis/

[70] T. van Do, I. Jørstad, S. Dustdar. Mobile Multimedia Collaborative Services. In Handbook of Research on Mobile Multimedia, Edited by Ismail Khalil Ibrahim, Idea Group Publishing, USA, 2006

[71] Tang, J. C.; Yankelovich, N.; Begole, J.; Van Kleek, M.; Li, F. and Bhalodia, J.: ConNexus to Awarenex: extending awareness to mobile users. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (Seattle, Washington, United States). CHI '01. ACM Press, New York, NY, 221-228. DOI= http://doi.acm.org/10.1145/365024.365105

[72] Truong, H.L.; Dustdar, S.; Baggio, D.; Corlosquet, S.; Dorn, C.; Giuliani, G.; Gombotz, R.; Hong, Y.; Kendal, P.; Melchiorre, C.; Moretzky, S.; Peray, S.; Polleres, A.; Reiff-Marganiec, S.; Schall, D.; Stringa, S.; Tilly, M.; Yu, H. Q.: inContext: A Pervasive and Collaborative Working Environment for Emerging Team Forms. SAINT 2008: 118-125

[73] VoiceXML Forum, 20. 5. 2010, http://www.voicexml.org/

[74] VoiceXML, 20. 5. 2010, http://www.w3.org/TR/voicexml20/

[75] W3C “Voice Browser” Working Group, 20.5.2010, http://www.w3.org/Voice/

[76] W3C Multimodal Interaction Framework, 1.7.2010, http://www.w3.org/TR/mmi-framework/

[77] Wasinger, R.; Krüger, A.; Jacobs, O.: Integrating Intra and Extra Gestures into a Mobile and Multimodal Shopping Assistant. In Lecture Notes in Computer Science, Volume 3468, Jan 2005, Pages 297 - 314, DOI 10.1007/11428572_18, URL http://dx.doi.org/10.1007/11428572_18

[78] WebSphere Device Developer 5.7.1, 1.7.2008, http://www.ibm.com/software/wireless/wsdd/

[79] WebSphere Everyplace Micro Environment, 1.7.2008, http://www-306.ibm.com/software/wireless/weme/

[80] XHTML, 20. 5. 2010, http://www.w3.org/TR/xhtml1/

[81] XHTML+Voice Programmer’s Guide, IBM WebSphere Multimodal Toolkit

[82] XHTML+Voice, 1. 7. 2008, http://www.voicexml.org/specs/multimodal/x+v/12/

[83] XML Events, 20. 5. 2010, http://www.w3.org/TR/xml-events/