Voice-based generic UPnP Control Point

Andreas Bobek, Hendrik Bohn, Frank Golatowski

Andreas Bobek, Hendrik Bohn, Frank Golatowski, Institute of Applied Microelectronics and Computer Science, University of Rostock, Rostock, Germany, e-mail : (andreas.bobek, hendrik.bohn, frank.golatowski)@etechnik.uni-rostock.de

***Abstract*—A concept for a generic voice based Universal Plug and Play (UPnP) control point (client) is presented in this paper which can be used to operate services on remote devices. Generic control points are a powerful alternative to special control points which are tailored to their special purposes. By using such a generic control point it is possible to control each UPnP device via voice without the need of manual adaptations. Starting from the analysis of the UPnP Device Architecture (UDA) and examination of the possibilities of voice supporting technologies we will show the feasibility of a generic voice based UPnP control point realization. The generic user interface is based on the standardized UPnP device and service description. Two different architectures will be introduced (coupled and decoupled solution) and their advantages, disadvantages and application will be discussed. Furthermore, we demonstrate that voice based configuration increases the user friendliness of operating a control point.**

***Index Terms*—UPnP, VoiceXML, voice user interfaces, generic user interfaces.**

I. Introduction

In June 1999, the *Universal Plug and Play* (UPnP) Forum [1] was founded by Microsoft to set up a standard for integrating devices and services in a network area and to define their usage. UPnP is based on a client-server-concept where UPnP *devices* (servers) offer their services to a network and UPnP *control points* (clients) can be utilized to find and use them.

The UPnP specification describes applied protocols and their appropriate use for interaction between devices, services and control points. The user interface is not defined. Although graphical implementations (e.g. Universal Control Point by Siemens [2]) exist some application fields require voice based user interface (e.g. for handicapped users or users on the move).

Most UPnP control points are purpose-built for particular applications (*special control points*). For instance, a control point to operate a heater could be adapted to its special function. A graphical throttle could adjust the temperature and a digital display could show current and desired temperature. If a voice interface is attached, a temperature change could be initiated by the question: “Do you wish to change the temperature?”

Special control points are tailored to their special purposes. Unfortunately special control points must be redesigned for each application and only operate in a very specific scope. *Generic control points* offer more flexibility. They are designed for a number of applications. The Universal Control Point [2], developed by Siemens, is such a generic control point for graphical user interfaces. Information from the UPnP device and service descriptions are presented in a textual format. The state of the device being operated can be changed using a keyboard.

Speicher describes the *Voice Service Browser* which can be used to search and execute services [3]. A special voice interface attached to each service is the precondition for service access. The target platform is the *Java Enhanced Service Architecture* (JESA) [4].

The *ICrafter Framework* [5], *Personal Universal Controller* [6], *EXACT* [7] oder *XWeb* [8] offer voice user interfaces to operate services but either require additional abstract service or user interface descriptions in *HyperText Markup Language* (HTML) [9], *Voice Extensible Markup Language* (VoiceXML) [10] or other native XML dialects, or are not able to create user interface generically. Furthermore, mentioned technologies do not address selection and search of services [3].

In this paper, we introduce a voice based generic UPnP control point. In Chapter II necessary technologies are introduced. Requirements for our voice based generic UPnP control point are discussed in Chapter III. The architecture is presented in Chapter IV whereby two approaches are considered. In Chapter V possibilities for user interface realization are discussed.

II. Key Technologies

Below we introduce VoiceXML, the term VoiceXML Browser and UPnP as the used basic technology.

A. VoiceXML & VoiceXML Browser

VoiceXML is a standardized XML based description language designed for creating audio dialogs. VoiceXML elements therefore feature instructions as speech recognition, recording of spoken input, synthesizing speech, digitizing audio and recognition of input via *Dual-tone multi-frequency* (DTMF) [11]. They are interpretable and executable by a *VoiceXML Browser*.

VoiceXML is build on existing web technologies and their underlying concepts. XML is a key technology of VoiceXML. Thereby, available tools and software components (e.g. XML parsers and XML editors) as well as developers which are familiar with other XML dialects are supported.

VoiceXML documents are linked together by using the *Uniform Resource Identifiers* (URI) [12]. URI is a standardized technology to reference resources within a network. User input is carried out by web forms similar to HTML. Therefore, forms contain input fields which are edited by the user and are sent back to a server.

For a better illustration, VoiceXML browser and HTML browser are comparable due to similar concepts and roles within architectures (fig. 1).

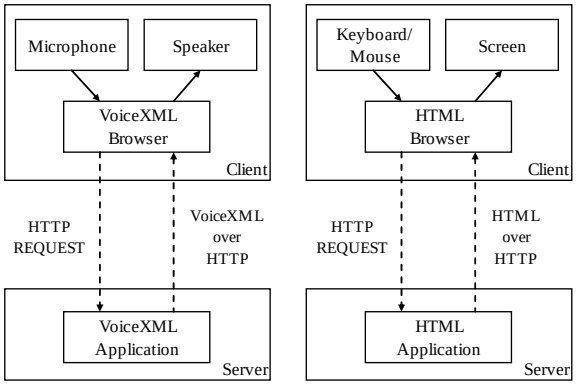

Fig.1 VoiceXML browser and HTML browser integrated into the client-server architecture

Both architectures are based on the client-server concept. The browser and a hardware interface controlling the browser are located at client side. In case of a HTML browser, the hardware consists of a keyboard/mouse for input and a screen for output. In case of a VoiceXML browser, input is realized by a microphone and output by speakers. Telephonic control is possible alternatively. In the latter case, other architectures are feasible (e.g. using the VoiceXML Browser as a server component).

Resources for the browser (data output/display and scripts for server-side generated documents) are located at server side. In case of HTML, resources are static HTML pages, images, videos, flash animations and scripts for dynamic creation of HTML pages. VoiceXML server resources are static VoiceXML files, digital audio data (*audios*), instructions for speech recognition (*grammars*) and scripts likewise. The same script technologies are applicable for creating VoiceXML documents as for creating HTML pages (e.g. *Common Gateway Interface* (CGI) [13] scripts, *Active Server Pages* (ASP) [14] and *Java Server Pages* (JSP) [15]).

In both architectures data is transmitted from client to server and back using the *Hypertext Transfer Protocol* (HTTP) [16].

Since transmission protocols and technologies for dynamic generation of HTML/VoiceXML are the same, web servers for HTML applications are also usable for VoiceXML applications.

Fig. 2 illustrates the design of a VoiceXML browser. The *VoiceXML Interpreter* is the central component of the browser. It interprets VoiceXML files and communicates with the *Speech Recognition Engine*, *Text-to-speech Engine*, *Voice Recorder* and *Audio Player*.

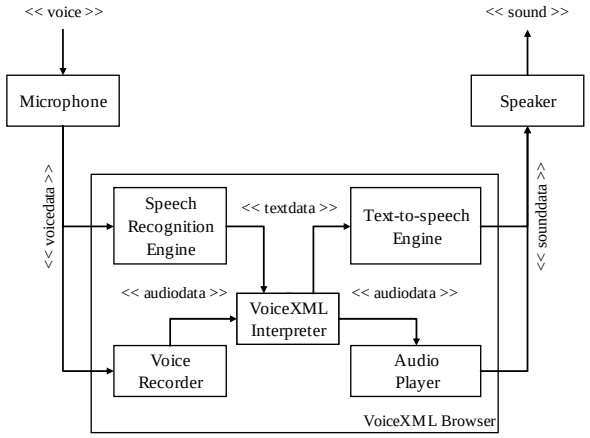

Fig.2 Components and data flow of a VoiceXML browser

B. UPnP

UPnP supports ad-hoc networking of devices and services by defining their announcement, discovery and usage [17]. Programming languages and transmission media are not assumed. Only protocols and interface are specified instead. Device and service descriptions as well as messages controlling a device (*ControlMessages*) are formatted in *Extensible Markup Language* (XML) [18].

UPnP defines three communicating components. The *device* (server) is a logical unit and may contain other devices. Devices are called *root device* if they are not contained in other devices. Devices are uniquely characterized by properties like model number, model name, vendor etc. Devices offer *services* which operate as interface to a device. A service offers methods (*actions*) which can be invoked by *control points* (clients). Furthermore, a service manages its state in variables (*state variables*). Control points search for services, invoke their actions and request state variables. The terms ‘device’ and ‘control point’ stand for logical components (not physical), since a device can also act as a control point.

The UPnP specification is divided into six phases: addressing, discovery, description, control, eventing and presentation.

Addressing - In order to participate in networking a device needs a unique IP address. Therefore it has to lookup the *Dynamic Host Configuration Protocol* (DHCP) server (default) or has to assign an IP address per *AutoIP* to itself [19]. AutoIP is a standardized technique for dynamic allocation of IP addresses if a DHCP server is not available.

Discovery - The *Simple Service Discovery Protocol* (SSDP) is used for discovery [20]. Discovery messages comprise announcements of integration and removing devices and services as well as discovery of services. When entering a UPnP network a device sends messages containing information about itself, its containing devices and services (*Advertisements*). Control points may save and use them.

A device has to resend advertisements periodically, because of their limited validity duration (lease concept). To find a device or a service a control point sends out search

discovery messages. Devices that match the query respond directly.

SSDP features searching for devices and services. The query could relate to all reachable devices and services, only to root devices or to devices and services with a specific type (predefined by UPnP forum or self-defined).

Description - A control point requests the device description if it is interested in a device. The description consists of device specific information, a list of all containing devices and information about the services. A control point requests the service description if it is interested in a service. The description specifies actions and state variables. Descriptions are sent in XML format.

Control - Control messages are used for invoking actions und requesting state variables. They are transmitted via the *Simple Object Access Protocol* (SOAP) [21]. As a result of invoking actions, the service responds by sending an updated list of changed state variables.

Eventing - A control point can be registered at a service to be informed about state changes. The service sends event messages to all registered control points after changing the value of a state variable. Event messages are transmitted via the *Generic Event Notification Protocol* (GENA) [22].

Presentation - A device may offer a special *Uniform Resource Locator* (URL) [23], which represents an adapted user interface to control the device. Thereupon, the control point can connect to the URL and display the specific HTML page in a browser.

III. Required Functions of generic voice-based UPnP Control Point

This chapter describes necessary functions of a generic voice-based UPnP control point.

Generic control points are a powerful alternative to special control points which are tailored to their special purposes. By using a generic control point it is possible to control each UPnP device via voice without the need of manual adaptations. An existing realization of a generic voice-based UPnP control point is not known to the authors.

The design of a generic UPnP control point results from the needs of a capable user interface and have to meet the complete *UPnP Device Architecture* (UDA) [17]. This is in contrast to a specific control point as described in Chapter I.

According to an analysis of the UDA we define functions necessary to manage a generic control point .

The functions are grouped into message management and timeout management. Table 1 gives an overview of all possible messages that pass a control point. Thus, the control point is responsible for evaluating all incoming messages, forwarding them to the user interface and, if necessary, replying them. Furthermore, the control point has to accept input data from the user interface and analyzes it.

Timeout management includes asynchronous searching of devices and services (discovery timeout) as well as updating the device cache. A device that does not resend their advertisements after expiration could not be reached anymore. Both, updating the device cache and sending a message to the user interface are necessary (alive timeout).

The implementation of the mentioned requirements is independent from the kind of user interface. In contrast, the preparation of messages for the user depends on the user interface.

Tab.1 Incoming and outgoing messages related to an UPnP control point

Outgoing Message	*Incoming Message*	*asyn.*	*syn.*	*Protocol*
DiscoveryReq	DeviceDiscoveryResp ServiceDiscoveryResp	x		HTTPU
DeviceDescriptionReq	DeviceDescriptionResp		x	HTTP
ServiceDescriptionReq	ServiceDescriptionResp		x	HTTP
-	DeviceAdvertisementAlive	x		HTTPU
-	ServiceAdvertisementAlive	x		HTTPU
-	DeviceAdvertisementByeBye	x		HTTPU
-	ServiceAdvertisementByeBye	x		HTTPU
VariableReq	VariableResp		x	SOAP
ActionReq	ActionResp		x	SOAP
EventSubscriptionReq	EventSubscriptionResp		x	HTTP
EventReSubscriptionReq	EventReSubscriptionResp		x	HTTP
UnsubscriptionReq	UnsubscriptionResp		x	HTTP
EventMessageResp	EventMessageReq		x	HTTP

IV. Architecture

Below, we present two architectures for a voice-based generic UPnP control point: The *coupled* and *decoupled solution* (fig. 4). Advantages and disadvantages are discussed with regard to possible fields of application and implementation. Both solutions are designed modularly and also use a modular designed UPnP control point and VoiceAPI. Therefore, the common used components are introduced at first.

A. UPnP Control Point (CP)

The UPnP control point component is the realization of the UDA. The components *UDP*, *TCP*, *SSDP*, *GENA*, *HTTP* and *SOAP* realize according protocols. They accept incoming packets and forward them to higher components. *Discovery*, *Eventing*, *Control* and *Description* parse incoming messages and push them through an interface to the *Control Point Manager* (CPM), fig. 3. The CPM manages devices and services in a *Device Container* and in a *Service Container* respectively. Only devices and services are considered the CPM is interested in. The device and service instances contain information extracted from their descriptions. The CPM is not part of the UPnP control point, since the application on top of the CPM decides which devices and services have to be cached and which events have to be subscribed to.

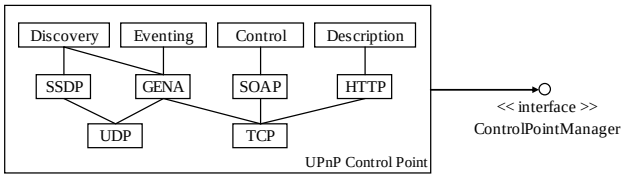

Fig.3 Components of the UPnP control point

B. VoiceAPI

The *Voice Application Programming Interface* (VoiceAPI) is responsible for speech recognition (*Speech Recognition Engine*), synthesizing of speech (*Text-to-*

speech Engine), recording of audio data (*Voice Recorder*) and play back of existing audio data (*Audio Player*). IBM offers such a VoiceAPI, called WebSphere Voice Server SDK [24].

C. Coupled architecture

The term *coupled* is used because the application has direct access to the VoiceAPI. The VoiceAPI recognizes spoken words and informs the coupled voice application via interfaces. Contrary, the voice application can use the VoiceAPI for speech synthesis. Communication between *Coupled Voice Application*, UPnP control point and VoiceAPI is based on programming language specific messages.

D. Decoupled architecture

In comparison to the coupled architecture communication with the voice components inside the decoupled architecture is based on request-response cycles using VoiceXML over HTTP. Similar to the coupled application communication between *Decoupled Voice Application* and UPnP control point is based on programming language specific messages.

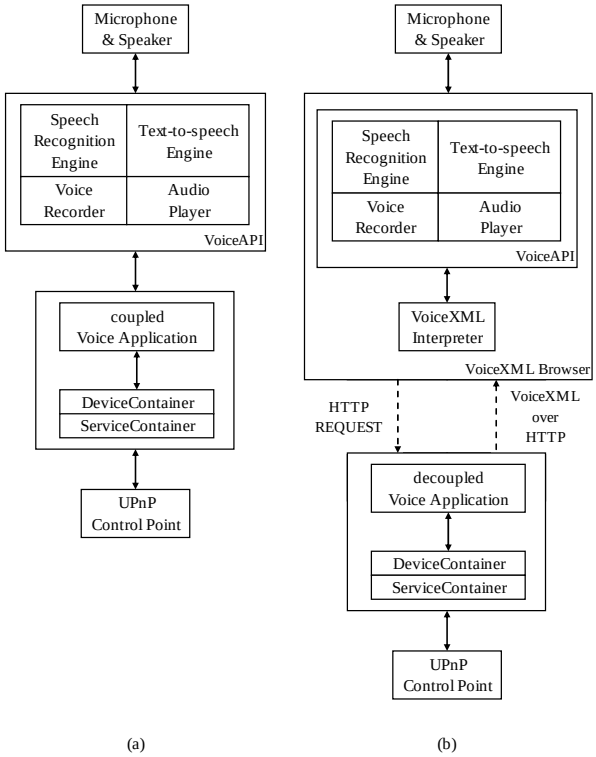

Fig.4 Coupled (a) and decoupled (b) architecture

Every communication process between VoiceXML browser and voice application has to be initiated by the VoiceXML browser (client) as a request to the voice application (server) (fig. 4b). For this purpose VoiceXML files contain *grammars*, which specify expected words and phrases. Grammars are associated with link elements within VoiceXML files. A link contains a URL which refers to the voice application. The browser connects to that URL as soon as it recovers a match between spoken input and one of the grammars. The voice application identifies the request reason based on the URL. Parameters can be assigned to the URL for simplification. The voice application responds with a new VoiceXML file.

The VoiceXML browser consists of the VoiceAPI (as in the coupled architecture) and the VoiceXML interpreter (responsible for executing VoiceXML files). So, the decoupled architecture implements the client-server concept introduced in Chapter II.

E. Comparison

The coupled solution is similar to a local running application. Both use native communication methods which are based on programming language specific messages. The decoupled solution is similar to a HTML web application (server) which is controlled by a HTML browser (client). Both use the client-server model and communicate via request-response cycles. Communication has always to be initiated by the client (request).

State changes of an application that requires an update of the graphical user interface are immediately executed by control components of the local running application. But it is not possible for a web server application to inform its browser about state changes until the next request-response cycle.

The same applies for the voice browser in the decoupled architecture. For instance, if the value of a subscribed state variable changes, the device sends an appropriate *EventMessageRequest* to the control point. But the control point is not able to directly inform the voice browser which could inform the user by synthesize corresponding speech. That problem applies to all asynchronous incoming messages and all synchronous incoming request messages (table 1).

There is a workaround regarding that problem. The browser sends check requests to the server to request periodic state changes even when no input is generated by the user.

The workaround is achieved by a time controlled reload. A META tag is used for that purpose in a HTML browser. In a VoiceXML browser the *noinput* event handler could be set up. The handler is called by the browser if no user input is available after a certain time.

The coupled architecture implies libraries and hardware for input and output on the same device. Therefore, user and control point are located at the same position. For this scenario the user can use a local UPnP network which is available close to the user. Even an unknown network is possible, where the control point has to allocate an IP address dynamically.

In the decoupled architecture data is transmitted via HTTP (over IP). Therefore, VoiceXML browser/user and control point could be located at different positions. In this scenario an UPnP network could be controlled remotely. Since UPnP was designed for local networks it is important where the control point is located, but not the user. In the latter case the IP address has to be known to the VoiceXML browser (or user), because it is required to create a connection to the decoupled voice application.

Both architectures cover different fields of application. The coupled application could be completely implemented into a portable device. Combined with *Bluetooth* [25], *Wireless Local-Area Network* (WLAN) [26] or another medium it could be used in any local network to search for UPnP devices and services and to use them.

The control point of the decoupled architecture could be located in a house or in a car. It is an inherent part of the local network and has it own static IP address. The user can control services of its house or car, even in absence.

Beside presented architectures there is another one possible. VoiceAPI, Voice application (coupled or decoupled) and UPnP control point could be located at a server in the local target network. Furthermore, a telephonic interface to the server is available. In that case the entire control point could be remotely controlled by a telephone. This architecture is a mixed alternative of the coupled and decoupled architecture and combines their advantages. The control point can be controlled locally and remotely. Furthermore, the VoiceXML based implementation of the user interface could be avoided.

V. User Interface

One of the main problems in developing voice-based generic UPnP control points is the design of user interfaces. Compared to a graphical interface the voice interface is limited to its possibilities regarding input and output. Using a voice-based application only the sequential play back is meaningful. Although a parallel play back is technically possible, the user is not able to notice multiple simultaneously spoken information. Normally, graphical output is parallel output.

A further difference is the temporal availability of information. Acoustic output is only available at the moment of play back. Graphical output is shown until the display changes.

Following questions result from designing the user interface (see table 1): How can a search request be verbalized (DiscoveryRequest)? How can a device be chosen to query state variables (VariableRequest) or to invoke actions (ActionRequest)? How can events be subscribed to (EventSubscriptionRequest)? How can devices and services be described which are discovered by a generic control point (DiscoveryResponse)? How can the generic voice-based control point inform about the availability of new and the removal of known devices (AdvertisementAlive, AdvertisementByeBye)? How can it notify changes of state variables (EventMessageRequest)?

A UPnP device is uniquely identifiable by its *Universally-unique Identifier* (UUID). It can be found in the *Unique Device Name* (UDN) entry in the device description. Another possibility is making use of the *friendly name*, but it is not unique. The friendly name contains a short and descriptive name of the device.

UDN and *friendly name* are mandatory information within a device description. A user could use the friendly name to access a device. For that purpose the friendly names of all known devices have to be listed as options within the grammar element. If two devices got the same friendly name the control point could suggest a new by selecting from a predefined device name pool or the user specifies a name by *configuration*. The possibilities of configuration are described below.

The UPnP discovery protocol specifies searching for devices and services (see Chapter IIB). Searching for root devices as well as searching for all reachable devices and services could be implemented by providing simple keywords, accordingly. Searching for certain types of devices or services is more difficult. It is possible to query types defined by the UPnP forum. Therefore, the user must be provided with a list containing all known types. This list is a static list and has to be updated manually.

It is not possible to search for vendor-defined types if using a voice-based generic control point. An overall searching could be used instead.

The possibilities of naming devices can be adapted to the names of actions and state variables. Their names are mandatory information within the device description. Therefore, the user could be provided with a list to choose from, if she/he wants to invoke an action or query a state variable.

As mentioned in Chapter IIB, actions may contain parameters. The parameter values have to be specified before they can be executed. A parameter entry consists of a parameter type and optionally of a predefined value and range. This information can be requested by the user at any time or at parameter specification (on demand), respectively. The UPnP forum specified 24 predefined parameter types which can be categorized in four groups: numerical (12), time-based (5), char-based (2) types and remaining types (boolean, bin.base64, bin.hex, uri, uuid).

Spoken input of the user has to be converted into a value according to the parameter type. Using VoiceXML the predefined input field types could be used: for numerical types the field type number or digit, for time-based date and time, for boolean boolean and for char-based, bin.hex, url and uuid the phonetic alphabet (introduced in the next chapter) could be used.

Solely, there is not an appropriate input possibility for the type bin.base64. Since this type is based on binary data, we can not find a scenario how binary data could be entered directly, neither in a graphical nor a voice-based generic control point.

We assume that binary data being given indirectly in all cases, for instance as a file handle. The source of such a file handle is not assumable; therefore that problem is not solvable generically.

The bin.base64 type is the one and only flaw in designing a voice-based generic UPnP control point.

Instead of entering an UUID phonetically, it can be entered indirectly by using a device name or service name. This methodology is equivalent to a reference transfer in object oriented programming languages.

The speech conversion of state change messages is simply done by reading out the names of according device, service, state variable and the new value in dependence of the

type.

A. Speech-to-text and phonetical alphabet

Normally, it is assumed that the user has to choose words or phrases from a given static or dynamic list when developing a voice-controlled system. However, for specifying parameter values in actions such a list can not be provided. Parameters of the type *string* are exceptions, when their description contains an *allowedValueList* element.

There are two possibilities to solve this problem. Using a *speech-to-text engine*, speech can be converted into text. Although this feature is provided by some components available on the market (e.g. iCommunicator [27]), it is not part of the VoiceXML specification. The second possibility is an alphanumerical alphabet which is used such as the phonetic alphabet (alpha, beta, charlie...). This special kind of alphabet has the advantage to be implemented in most languages and to be well known. Furthermore, it could be used in VoiceXML´s grammars. However, it takes more time specifying a word because a whole word stands for each letter.

B. Configuration

The above listed solutions are enough to realize a voice-based generic UPnP control point. Unfortunately, there is a lack of user friendliness. That is why we suggest extending the control point by an optional *configuration* component. It is possible to specify adapted names for devices and services using the configuration component. The input effort for the user can be reduced significantly by using an alias name for actions together with parameters and predefined values.

A device can be stored in a database by saving its configuration parameters and its UUID. If the device is in offline mode the configuration settings are kept.

A control point can also be configured to adapt the response to events such as the availability of new devices or removal of known devices. In active mode the control point speaks directly to the user, in passive mode it waits until the user requests.

VI. Conclusion

Starting from listing requirements in Chapter III we have shown that it is possible to realize an UPnP control point voice-based and generically. Our concept works without special adoption of the services. The basics of our approach are the UPnP device descriptions.

The sole exception for a generic solution are input and output based on bin.base64 type.

Configuration in combination with the phonetical alphabet can be used to optimize the user interface and to work together with the VoiceXML technology.

The coupled and decoupled architectures were introduced. Both architectures cover the need of a local and a remote control of UPnP networks which have local scope, normally.

VII. Acknowledgment

This work was supported by the SIRENA project in the framework of the European premier cooperative R&D program ”ITEA”.

VIII. References

[1] Universal Plug and Play Forum. http://www.upnp.org.
[2] Siemens Plug & Play Technologies. http://www.plug-n-play-technologies.com.
[3] Sebastian Speicher, *Entwicklung und prototypische Implementierung eines Konzeptes zur Sprachsteuerung von spontan vernetzten mobilen Geräten*, diploma thesis, University Rostock, Rostock, 2003.
[4] Stephan Preuss, *Java Enhanced Service Architecture*, Technical Report, University Rostock, Rostock, 2003.
[5] Ponnekanti, R. Shankar, Brian Lee, Armando Fox, Pat Hanrahan and Terry Winograd, "ICrafter: A Service Framework for Ubiquitous Computing Environments" Lecture Notes in *Computer Science*, 2201, 2001, pg. 56–75.
[6] Nichols, Jeffrey, Brad A. Myers, Michael Higgins, Joseph Hughes, Thomas K. Harris, Roni Rosenfeld and Mathilde Pignol, "Generating remote control interfaces for complex appliances", *Proceedings of the 15th annual ACM symposium on User interface software and technology*, ACM Press, 2002, pg. 161–170.
[7] Alexander Yates, Oren Etzioni and Daniel Weld, "A reliable natural language interface to household appliances", *Proceedings of the 2003 international conference on Intelligent user interfaces*, ACM Press, 2003, pg. 189–196.
[8] Dan R. Olsen, Jr., Sean Jefferies, Travis Nielsen, William Moyes and Paul Fredrickson, "Cross-modal interaction using XWeb", *Proceedings of the 13th annual ACM symposium on User interface software and technology*, ACM Press, 2000, pg. 191–200.
[9] Hypertext Markup Language 4.0 Specification, April 1998. http://www.w3.org/TR/REC-html40/.
[10] Voice Extensible Markup Language. http://www.w3.org/TR/2004/REC-voicexml20-20040316/.
[11] Dual-tone multi-frequency, ITU-T Recommendation Q.23.
[12] T. Berners-Lee, MIT/LCS, R. Fielding, U.C. Irvine, L. Masinter, Xerox Corporation, *Uniform Resource Identifier (URI): Generic Syntax*, Network Working Group, Request for Comments 2396, 1998.
[13] Shishir Gundavaram, *CGI Programming on the World Wide Web*, O´Reilly UK, 1996.
[14] Francis, Fedorov, Harrison, Homer, Murphy, Sussman, Smith and Wood, *Professional Active Server Pages 2.0*, Wrox Press Ltd, 1999
[15] Marty Hall, Larry Brown, *Core Servlets and JavaServer Pages*, Prentice Hall PTR, 2003.
[16] R. Fielding, UC Irvine, J. Gettys, Compaq/W3C, J. Mogul, Compaq, H. Frystyk, W3C/MIT, L. Masinter, Xerox, P. Leach, Microsoft, T. Berners-Lee, W3C/MIT, *Hypertext Transfer Protocol -- HTTP/1.1*, Network Working Group, Request for Comments 2616, 1999.
[17] UPnP Device Architecture 1.0, December 2003.
[18] David Hunter, Kurt Cagle, Chris Dix, *Beginning XML*, Wrox Press, 2001.
[19] R. Droms, Bucknell University, *Dynamic Host Configuration Protocol*, Network Working Group, Request for Comments 2131.
[20] Yaron Y. Goland, Ting Cai, Paul Leach, Ye Gu, Microsoft Corporation, Shivaun Albright, Hewlett-Packard Company, *Simple Service Discovery Protocol/1.0*, Internet Engineering Task Force, Internet Draft, 1999, 2000.
[21] Simple Object Access Protocol, http://www.w3.org/TR/soap/.
[22] J. Cohen, S. Aggarwal, Y. Y. Goland, *General Event Notification Architecture*, Internet Draft, 2000.
[23] T. Berners-Lee, CERN, L. Masinter, Xerox Corporation, M. McCahill, University of Minnesota, *Uniform Resource Locator*, Networking Group, Request for Comments 1738, 1994.
[24] IBM WebSphere Voice Server SDK, http://www-306.ibm.com/software/pervasive/voice_toolkit/.
[25] Bluetooth Special Interest Group, http://www.bluetooth.org.
[26] Wireless local-area network, IEEE 802.11.
[27] 1450, Inc., iCommunicator V.4.0, http://www.myicommunicator.com.