Automatic Generation of General Speech User Interface

Dongyi Song

Department of Informatics, Media Informatics Group
Ludwig-Maximilians-Universität München
Munich, Germany
Dongyi.Song@ifi.lum.de

Abstract

This paper describes a novel approach to generating a general speech user interface to different applications by combining the existing speech user interfaces of the applications automatically. A general speech user interface enables the user to access different applications via speech simultaneously. One key issue of constructing such an interface is how different applications should be integrated. The approach represented in this paper integrates different applications into a general speech user interface by automatic merging their dialogue specifications into a unified dialogue specification, which provides necessary information for a multi-application supported dialogue system to enable the general speech user interface to these applications. In doing so, three issues in general speech user interfaces are addressed – transparent application switching, task sharing and information sharing.

1. Introduction

A general speech user interface allows the user to access information and services provided by different applications via speech simultaneously. Such an interface is usually provided by a multi-application supported dialogue system ([1, 2]). There are different architectures proposed for these dialogue systems. This paper addresses the issue of how to endow these dialogue systems with several applications.

The key issue of the deployment of a multi-application supported dialogue system to a set of applications to provide a general speech user interface is that different applications must be integrated together. This integration can take place at two different levels – dialogue manager level and dialogue specification level.

The open architecture of GALAXY-II [3] developed in the context of DARPA project is an example for integrating different applications at the dialogue manager level. Different applications are controlled by different dialogue managers individually and a Meta-dialogue manager is applied to manage all dialogue managers. In the provided general speech user interface, the user must switch the domain explicitly.

The architecture introduced in [1] is an example for integrating different applications at the level of dialogue specification. Each application is described by a Task-Description-Table (TDT), which provides the necessary information about the application for the dialogue manager. The dialogue manager can host different TDTs in runtime, so that different applications can be accessed. Domain switch is realized by the dialogue manager by means of activating the corresponding TDT. No extra Meta-dialogue manager is needed in this approach. However, at each time there is only one TDT active, so that only one application is accessible. In order to access a different application, the user still has to utter an explicit application switch command. A similar architecture can be found in [2] as well.

The approach to dialogue modeling for general speech user interfaces introduced in [4] improves the architecture of Lin [1] and Pakucs [2]. This approach brings different applications together into one dialogue system by arranging existing dialogue specifications of each application into an application hierarchy. Based on the description provided by the dialogue specification, the similarity between two applications is calculated. Afterwards, the applications are clustered in a binary tree with the most similar applications clustered under the same node in the tree. Given a user utterance, the similarity between the utterance and all applications are computed. If the similarity between the user utterance and a single application is beyond a predefined threshold, the desired application is determined. Otherwise the dialogue system navigates along the binary tree with clarification dialogues until the intended application is reached. Transparent application switching is enabled. However, interoperability among applications with respect to the task sharing problem is not solved in the work yet and is addressed as future work.

The general speech user interfaces provided by these existing approaches integrate the applications together into a set of applications without analyzing the dependencies between the applications, so that the interoperability between different applications cannot be considered appropriately.

This paper introduces a novel way to construct a general speech user interface based on existing speech user interfaces for different applications. Dialogue specifications of different applications will be merged into a unified dialogue specification. The dependencies between the integrated applications will be analyzed and represented appropriately in the general speech user interface. Based on the unified dialogue specification for different applications, a frame-based dialogue system [5] can provide the transparent access to all underlying applications.

Section 2 declares the requirements on the existing speech user interfaces, which can be integrated into the general speech user interface. Section 3 provides a formal definition for application specification in the existing single-application speech user interface. Section 4 proposes a novel algorithm for constructing a general speech user interface by merging existing dialogue specifications of different speech user interfaces. A dialogue example is illustrated in section 5. Section 6 summarizes and discusses open questions for further research.

2. Speech User Interface

A speech user interface is provided by a spoken dialogue system. If the dialogue system is capable of supporting access to different applications simultaneously, it is called a multi-application dialogue system. The purpose of this paper is to

propose a methodology to construct a general speech user interface by endowing multi-application dialogue system with appropriate application information automatically. This information is specified in a unified dialogue specification, which describes all underlying applications for the dialogue manager. This unified dialogue specification will be automatically generated based on existing dialogue specifications of different applications.

It is preserved, that all dialogue specifications of different applications are based on the same dialogue model and are thus supported by the same dialogue system. Since most spoken dialogue systems strive to support as many applications as possible, this preservation does not make evident restriction to the applications.

In order to support a general speech interface of arbitrary applications, the spoken dialogue system must be application-independent. In the system, the dialogue management mechanism is transparent to different applications. The system can be regarded as an aggregation of domain-independent dialogue management components and domain-specific knowledge. Porting the spoken dialogue system to new applications means to provide the corresponding dialogue specifications.

For the combination issue, the adopted dialogue specifications for defining the application-specific information in a dialogue system must be formal and declarative. An example is VoiceXML [7] or DIANEXML [6]. Because only declarative specifications can be compared and merged, native coding e.g. program codes implementation is not appropriate for these issues.

Among three main classes of dialogue modeling approaches [5], the frame-based dialogue modeling approach is best tailored to the requirements described above. In a frame-based approach, the algorithm for controlling the interaction between the user and the system is defined in the dialogue manager. The application information is specified in a dialogue specification as a set of frames. Each frame corresponds to a function provided by the application. The necessary information for executing the function is modeled as parameters of the frame. Deploying such a system to a specific application requires only the specification of the application using a set of frames.

The construction algorithm proposed in this paper is based on the assumption that a frame-based spoken dialogue system is used to support a set of applications, whose dialogue specifications are defined formally and declaratively.

3. Frame-based dialogue specification

The application specific information for a dialogue system is described in a dialogue specification. This section gives a formal definition for frame-based dialogue specifications, which abstracts different implementations.

In a frame-based approach an application is modeled as a set of frames.

$$A = \{F_1, F_2, ...F_n\}$$

There are different interpretations of a frame in different approaches. Abstractly, a frame has the following form:

$$F_i =< ID, FGr, \{PROMPT_1, ...\}, \{P_1, ..., P_m\}, Post >$$

$$P_i =< ID, PGr, Infer, \{PPrompt_1, ...\} >$$

In short, a frame consists of the following elements:

- *ID* – Identity of a frame
- *PROMPT* – Various prompts, which the system utters with respect to the frame in different situations, e.g. to confirm the user's intention to execute the frame.
- *FGr* – Key grammar (a context free grammar) defining all possible key expressions indicating a frame. E.g. the frame for "hotel reservation" will be indicated by the expressions "hotel reservation", "book a hotel", "reserve a hotel", etc.
- *POST* – Execution of the frame in the backend application and the corresponding information about the execution result
- *P* – Parameters of a frame modeling the necessary information required to execute the frame. Each parameter contains four elements:

1. *ID* – Identity of a parameter
2. *PGr* – A context-free grammar defining all possible natural user utterances for the parameter
3. *Infer* – An inference rule for inferring the parameter value based on the current dialogue state by the system [6]
4. *PPROMPT* – different prompts, which the system utters with respect to the parameter in different situation. E.g. query prompt, confirmation prompt, etc.

For example, the dialogue specification of an application for "flight reservation (AFR)" consisting of two functions –"flight reservation (FR)" and "check credit information (FCI)" can be modeled as $AFR = \{FR, FCI\}$. Table 1 illustrates the frame specification and table 2 gives an example for a parameter in this application.

For the combination purpose, another modeling example for a hotel reservation application (AHR) consisting of functions "hotel reservation (HR)" and "check credit card information (HCI)" is described in Table 3. The application is modeled as $AHR = \{HR, HCI\}$.

Due to the limited size of this paper, these applications as example here are simplified from the details such as ticket class, price, detailed location, etc. In practice, we have verified this method by building speech user interfaces for real industrial applications with it [8].

Table 1 Frame specification of AFR

	FR	FCI
ID	„flight_reservation“	„credit_information“
FGR	{“book an air ticket”, “reserve an air ticket”…}	{“credit card information“, “credit card“ …}
$Prompt_i$	“Do you want to book an air ticket?”	“Do you want to change your credit card information?”
$\{P_1 ... P_n\}$	{DEPARTURE_CITY, DESTINATION_CITY, DEPARTURE_DATE, DEPARTURE_TIME}	{LAST_NAME, FIRST_NAME}
POST	Reserve the ticket in backend system; Prompt “you ticket have been reserved.”	Get the credit card information from backend system; Prompt the user about it, i.e. “Your Visa card number is ….”

Table 2 Parameter example of AFR

	DEPATURE_CITY
ID	„DEPATURE_CITY“
PGr	{„Munich“, „Berlin“, „Frankfurt“…}
Infer	No inferrence rule
$PPrompt_i$	“Where do you want to fly from?”

Table 3 Frame specification of AHR

	HR	HCI
ID	“hotel_reservation”	“Credit_information”
FGR	{“book a hotel”, “reserve a room”…}	{“credit information”, “ask for credit card information”, …}
$Prompt_i$	“Do you want to book a hotel?”	“Do you want to check your credit card information?”
$\{P_1 ... P_n\}$	{CITY, DATE, DURATION}	{LAST_NAME, FIRST_NAME}
POST	Reserve the room in the hotel in backend system; Prompt “Thank you! The hotel has been reserved. ”	Get the credit card information from backend system; Prompt the user about it, i.e. “Your Visa card number is ….”

4. Constructing general speech user interface

Our approach to constructing a general speech user interface automatically based on the existing speech user interfaces of different applications is inspired by the idea that different frames of two applications can be merged together into one unified dialogue specification:

$$A_1 = \{F_{11}, F_{12}, ..., F_{1n}\} \quad A_2 = \{F_{21}, F_{22}, ..., F_{2n}\}$$

$$A_1 + A_2 = \{F_{11}, F_{12}, ..., F_{1n}, F_{21}, F_{22}, ..., F_{2n}\}$$

This basic principle can be applied to independent applications such as home intelligent environment and remote access to information database.

If two applications provide overlapped functions, which become all simultaneously active in the general speech user interface, it turns to be a problem for the dialogue manager to address the right function wished by the user. Also if two applications have some common information such as a user’s credit card information, the overlapped information should be shared by both applications in the general speech user interface. The next sections describe the different application relations and propose a solution for different scenarios.

4.1. Application Relations

Two applications can have different relations. An application is regarded as a set of frames and the essential information for a frame is the set of parameters of this frame. Therefore, to clarify the relations of two applications, we only have to look out the relations between the frames and parameters of the applications. We use the symbol $\approx$ to represent the similarity and the symbol $\neq$ to represent the independency of two frames or two parameters in advance.

4.1.1. Disjunctive applications

Two independent applications are referred as disjunctive. This is the case, if there are no similar frames and parameters in these applications. Such two applications are

independent from each other. The following form gives a definition for disjunctive applications:

$$A_1 \text{ and } A_2 \text{ are disjunctive } \leftrightarrow$$

$$\forall F_i \in A_1, F_j \in A_2 [F_i \neq F_j] \wedge$$

$$\forall p_k \in A_1, p_m \in A_2 [p_k \neq p_m]$$

For example, the applications "intelligent home environment" and "remote access to information database" are disjunctive applications. They do not provide any similar functions and do not have any shared information with each other.

4.1.2. *Functional overlap*

Two applications can provide common functions for the user, and we refer this situation as functional overlap between two applications.

We say two frames are similar, if they perform the same function for the user. Based on the formal definition of a dialogue specification, functional overlap can be defined in the following form:

$$A_1 \text{ and } A_2 \text{ have functional overlap } \leftrightarrow$$

$$\exists F_i \in A_1, F_j \in A_2 [F_i \approx F_j]$$

Two applications have functional overlap, if there is some frame in A1, which is similar as some frame in A2. In the examples introduced in section 3, AFR and AHR both provide the same function for "checking credit card information". The corresponding frames FCI (AFR) and HCI (AHR) are similar.

4.1.3. *Parameter Semantic Overlap*

Different applications can share common information with each other such as username, password, etc. We refer this situation as parameter semantic overlap.

We say two parameters are similar, if they refer to the same semantic concept and are related to each other. So semantic overlap can be defined in the following form:

$$A_1 \text{ and } A_2 \text{ has semantic overlap } \leftrightarrow$$

$$\exists P_i \in F_n, F_n \in A_1, P_j \in F_m, F_m \in A_2$$

$$[P_i \approx P_j \wedge F_n \neq F_m]$$

Two different applications have semantic overlap, if there is some parameter in application A1, which is similar to some parameter in application A2. And we do not consider the parameters in similar frames with respect to their similarity.

For example, the frame FR in application AFR and the frame HR in application AHR have two semantic overlaps - DESTINATION_CITY and CITY, DEPARTURE_DATE and DATE. The destination city of the flight is probably the accommodation city of the hotel. And the departure date of the flight is probably the hotel check in date, if the flight is always a short trip.

4.2. Application comparison

Before handling different application relations, the relation must be determined first. For applications represented as a set of frames, which consist of a set of parameters, the application relations can be determined based on the comparison of frames and parameters.

4.2.1. *Frame Comparison*

Considering different elements of a frame, the most important element, which can indicate the similarity of functions provided by two frames, is the key grammar. In a key grammar, all possible expressions (key phrases) indicating the function provided by the frame are defined. The key grammar distinguishes a frame from the other. A speech user interface should accept as many key phrases as possible, so that the user has as much flexibility as possible to express his/her intention. For example, for a frame "flight reservation", there are different possible key phrases such as "book a ticket", "make a reservation", etc. The speech user interface should be able to accept all these words, so that the user does not have to remember the "command" for the frame, and the dialogues are natural and intelligent. Therefore, we can assume all possible natural utterances representing a frame are defined in its key grammars. So we can make the following proposition:

If there is an identical key phrase defined in key grammars of two frames, the frames are similar. Otherwise, they are considered as independent frames.

This is a heuristics which may miss some intended similarities, but the heuristics has the nice property that it gets more prices if the quality of the key grammar is improved. And in general high qualified key grammars are required in a flexible and natural dialogue system.

The similarity judgment can be defined by the following form:

$$F_1 =< ID_1, FGr_1, \{PROMPT_{11}, ...\}, \{P_{11}, ..., P_{1n}\}, Post_1 >$$

$$F_2 =< ID_2, FGr_2, \{PROMPT_{21}, ...\}, \{P_{21}, ..., P_{2n}\}, Post_2 >$$

$$\exists w [w \in L(FGr_1) \wedge w \in L(FGr_2)] \leftrightarrow F_1 \approx F_2$$

L(G) represents the language defined by the grammar G. Section 4.2.3 will introduce the algorithm to compare two grammars to find their intersection.

4.2.2. *Parameter comparison*

A parameter defines its range of values with its parameter grammar. If two parameters refer to the same semantic and are related to each other, they must have at least one common value in their value ranges. Otherwise, they would never have the same value, so there is no chance for information sharing in two applications with respect to these parameters. This can be defined by the following form:

$$P_1 =< ID_1, PGr_1, Infer_1, \{PPrompt_{11}, ..., PPrompt_{1n}\} >$$

$$P_2 =< ID_2, PGr_2, Infer_2, \{PPrompt_{21}, ..., PPrompt_{2n}\} >$$

$$\exists w [w \in L(PGr_1) \wedge w \in L(PGr_2)] \leftrightarrow P_1 \approx P_2$$

If there is any common expression defined by the grammars of two parameters of two applications, it is *possible* for these two applications to share the corresponding information represented by the parameters. However, the parameters must not represent the same object if their grammars have an intersection. For example, a date grammar defines {Monday,…, Friday} , where as a person name grammar specifies {Miller, John,…, Friday}. This problem is due to the ambiguity of the natural language, which is not avoidable. There are two ways to solve this problem. To maintain the process automatic, a configuration file can be introduced to state the minimal rate of the intersection. For example, the common expressions must be at least 30% of all expressions defined by a grammar. The other way is to

involve the dialogue designer to ensure the right decision. The algorithm comparing the parameter grammars to find their intersection will be introduced in section 4.2.3.

4.2.3. Grammar Intersection

To determine the similarity of two frames and two parameters, their key grammars and parameter grammars will be compared with each other to find out if there is any common expression defined in the corresponding languages. The grammars used in spoken dialogue systems are theoretically context-free grammars. The intersection problem of context-free grammars is not decidable. [9] Fortunately, most grammars in dialogue systems only exploit the regular power of the context-free grammars. More particular, the expressions defined by key grammars are quite limited, so that in principle all expressions can be enumerated in acceptable computing time. Therefore, we propose the following algorithm to compare two grammars G1 and G2 used in the dialogue specification for indicating a frame:

1. Enumerate/Generate all expressions of a grammar G1
2. Parse each expression of L(G1) with the Parser P(G2) for the other grammar G2.
3. If any expression is accepted by the Parser P(G2), a similarity of the appropriate grammars is determined.

Others than key grammars, a parameter grammar can define a very large to unlimited amount of expressions. For example, a grammar defining any possible digit sequence is infinite and a grammar defining all possible dates can specify more than three hundreds expressions. For this issue, we first reduce and approximate a parameter grammar, so that the expressions generated by it can be counted in real time.

The reduction of a parameter grammar is based on the idea that each "non-terminal" in the grammar refers to the same semantic category. There are normally many enumerations for each semantic category. The combination of these enumerations causes the unbounded number of expressions in the corresponding language. We solve this problem by reducing the enumerations for one semantic category to only one representative enumeration.

E.g. the following grammar G1 defines natural language for date:

$DATE \rightarrow DAY \quad MONTH$

$DAY \rightarrow First \mid Second \mid ... \mid Thirty - First$

$MONTH \rightarrow January \mid February \mid ... \mid December$ In this example, a "DAY" expression ranges from "first" to "thirty-first" and consists of thirty-one enumerations. When comparing with another grammar G2 with the objective to find out if there is any common expression defined by G1 and G2, we reduce all enumerations for "DAY" to one single representative instance, which is also consisted in the vocabulary of G2.

For example, assume that both "first" and "second" are in vocabulary of G2. Then, if "first January" is not in L(G2), "second January" will not either. With respect to a single "non-terminal", when comparing with another grammar G, different enumerations of it cause always the same similarity result. This is similar as different transitions between two states in finite-state automata. The transitions can be various, but the destination state is always the same.

If none of the thirty-one enumerations are contained in the vocabulary of G2, all "DAY" productions will be removed. Since no expressions produced with "DAY" will be accepted by G2.

If a grammar is recursive (if there is a "non-terminal" A, so that $A \rightarrow^{*} \alpha A$), we approximate the recursion with finite loops. For flexibility, we introduce a recursion parameter in the approximation process to indicate the unfolding levels of rule application. So a recursive grammar can be approximated according to specific cases.

After reduction and approximation, the language generation of parameter grammars becomes moderate so the same algorithm used for key grammars can be applied.

An alternative method to compare two context-free grammars with respect to their intersection is to approximate the grammars with the finite-state automata [10] and determine their intersection based on the intersection of the corresponding finite-state automata [9].

In comparison, the used algorithm is much more simple and based on a thorough experiments on different grammars used in practical dialogue systems, the adopted algorithm has been proven to be efficient and sufficient.

4.3. Solution for different relations

4.3.1. Functional Overlap

To solve the functional overlaps, we propose to merge two similar frames into one frame of the general speech user interface. Because of the fact that it is not relevant for the user which backend application executes the function. He/She only cares the result. If the user asks for his/her credit card information, he/she does not care if the credit card information is provided by the "flight reservation system" or the "hotel reservation system", if we suppose that the user always uses the same credit card in both systems.

The merged frame is constructed by inheriting dialogue modeling elements from the original frames and choosing the appropriate backend application to perform the function.

$$merge(F_1, F_2) =< ID_i, FGr_i, \{PROMPT_{i1}, ...\}$$

$$\{P_{i1}, ..., P_{in}\}, merge(Post_1, Post_2) > i \in \{1,2\}$$

Since two similar frames perform the same function for the user, their dialogue modeling parts are almost identical. So we can take any frame's dialogue modeling part. But it is not always trivial how the postconditions should be merged together. The function can be performed by one of the applications or by both. For example, the frame FCI in AFR and HCI in AHR in the example, if it is allowed to have different credit cards for "hotel reservation" and "flight reservation", the merged "credit card information" frame should invoke both applications to provide both credit cards information, otherwise it suffices to perform the function by only one application. The right decision is always domain-specific, so actually a generic pattern for merging postconditions is not possible. To maintain the process automatic, a configuration file can be applied to state the default settings for combination. For example, if the function of the similar frames provides the user some information, both applications can be invoked to execute the frame. If the function is rather a transaction execution such as ticket reservation, only one arbitrary application will be invoked. Another solution for this problem is to involve the dialogue designer to make the right merging decision for postconditions.

4.3.2. Parameter Semantic Overlap

To handle the parameter semantic overlap, common semantic information can be shared by different applications. This can be achieved by specifying the corresponding inference rule of each parameter. In simple case, the related parameters always have identical values and this can be specified by defining mutual inference in the Infer elements of the corresponding parameters. For example, the Infer of CITY in HR will be specified as CITY=ARRIVAL_CITY and the DATE of HR will have an inference rule: DATE = DEPARTURE_DATE.

5. Dialogue Example

We have developed a prototype of a combination tool following the combination algorithm introduced in this paper. With this combination tool, we have combined AFR, AHR, which are dependent and two more independent applications – "word geography information" and "intelligent home environment" together. The general speech user interface to all applications supported by the dialogue system DIANE [6] provides a dialogue as the following example, which has been simplified due to the limited size of this paper:

S (System): How can I help you?
U (User): I want to fly from Munich to Berlin tomorrow morning at ten o'clock.
S: Flight 731 from Munich to Berlin, starting at ten o'clock on 14th September, Do you want to take this flight?
U: Yes.
S: Your ticket has been reserved.
U: Turn on the light in the kitchen.
S: The light in the kitchen is on.
U: I want to book a hotel.
S: Do you want to reserve a hotel on 14th September in Berlin?
(The system infers the date and city automatically.)
U: Yes.
S: The hotel has been booked.
U: Which credit card has been taken actually?
S: Visa card number 12345678.
(FCI and HCI are merged into one and only AFR was chosen to perform the function.)
U: Which city is the Capital of Germany?
S: The capital of Germany is Berlin.

6. Conclusion and Outlook

This paper described how to generate a general speech user interface for different applications based on their existing speech user interfaces automatically. With the approach proposed in this paper an intelligent general speech user interface can be generated for different applications. This interface improves the rapid provisioning of speech user interfaces through its intelligent nature.

Compared with existing multi-domain or multi-application dialogue systems, the novelty of the approach proposed in this paper is the idea of automatic merging different dialogue specifications at the function layer thus enabling transparent access to different applications with a normal frame-based dialogue system. The advantage gained by this approach is an acceptable solution for integrating different related applications.

The approach has been implemented based on the DIANE speech dialogue system [6] and is fully operational. Case studies with real industrial speech dialogues from the area of telecommunication system configuration have been carried out. Based on a configuration file, automatic merging of non-trivial SmartWeb applications has been achieved (http://smartweb.dfki.de).

To extend the current work, an evaluation of the combination tool can be carried out for verifying the feasibility of the combination approach. Furthermore, user tests of the generated speech user interface can be carried out for verifying the usability improvement.

7. Acknowledgements

This work was partly funded by the German Federal Ministry of Education and Research (BMBF) under grant 01IMD01K. The author is solely responsible for the view expressed in this paper. The author wishes to thank Prof. Heinrich Hussmann, Dr. Hans-Ulrich Block and Jürgen Totzke for their kind support during the research and Siemens for funding.

8. References

[1] Lin, YC, Chiang, TH, Wang, HM, Peng, CM and Chang, CH., "The design of a multi-domain mandarin Chinese spoken dialogue system", *In Proc. ICSLP1998, pages 41-44*, 1998

[2] Pakucs B., "Towards Dynamic Multi-Domain Dialogue Processing", *In: Proc. EUROSPEECH2003 pp 741-744*, 2003

[3] Seneff, S., Lau, R. and Polifroni, J., "Organization, communication, and control in the GALAXY-II conversational system", *In Proc. Eurospeech '99, pp. 1271--1274*, Budapest, 1999

[4] Bui, T., Zwiers, J., Nijholt, A. and Poel, M., "Generic dialogue modeling for multi-application dialogue systems", *In 2nd Joint Workshop on Multimodal Interaction and Related Machine Learning Algorithms*, July 13 2005, Edinburgh, UK, pp. 12, 2005

[5] McTear, M., "Spoken dialogue technology: enabling the conversational interface", *ACM Computing Surveys, Vol. 34(1): 90-169*, 2002.

[6] Block, H.U., Caspari, R. and Schachtl, S., "Callable manuals - access to product documentation via voice", *In: Wahlster, W. (ed.) Special Journal Issue Conversational User Interface. it - Information Technology 46 (2004) 6, München, Oldenbourg Wissenschaftsverlag (ISSN 1611-2776): 299-304*, Munich, 2004

[7] McGlashan, S., Burnett, D., Carter, J., Tryphonas, S., Ferrans, J., Hunt, A., Lucas, B. and Porter. B., "Voice extensible markup language (voicexml) version 2.0", *Technical report, World Wide Web Consortium (W3C)*, Feb., http://www.w3.org/TR/voicexml20/, 2003

[8] Schilling, A., "Modeling of Speech Dialog Applications", *Student Project*, Ludwig-Maximilians-Universität, Germany, 2005

[9] Hopcroft J.E. and Ullman J.D., "Introduction to Automata Theory, Languages, and Computation", *Addison Wesley*, 1979

[10] Nederhof M. J., "Practical experiments with regular approximation of context-free languages", *Computational Linguistics, v.26 n.1, p.17-44*, March 2000