GRAMMAR-BASED DIALOGUE MANAGEMENT TECHNIQUES

Václav Matoušek, Tomáš Nestorovič

University of West Bohemia in Pilsen, Faculty of Applied Sciences
{matousek | nestorov}@kiv.zcu.cz

Abstract: This article is focused on some of spoken dialogue management techniques, in particular on those widely well known as grammar-based ones. Furthermore, as this article shows, even dealing with such more simple dialogue management techniques, the resulting dialogue manager can be capable to cope with relatively advanced phenomena, as for example the cross-references to historically spoken entities. This article is divided into two parts. In the first one, all three techniques mentioned above are overviewed and compared to each other. The rest of the article describes a dialogue manager, currently being developed at our department as a part of an experimental navigation system. Especially, it is focused on the crucial propositions and background ideas like the structure of manager's internal model of a world (static and dynamic frames and bindings between them) and structure of a dialogue history (history of computer and user's utterances and spoken entities).

1 Introduction

Dialogue management is conceived in machine reasoning, in particular, finding the best machine utterance as a response to previous user's ones and moreover keeping the discussed task in consistency with domain possibilities. A wide variety of methods has been evolved, embodying and regarding different complexity and usability as well. These methods are commonly divided in several groups and even if the division is not standardized in any way, it always involves grammar- and plan-based methods and methods for collaboration [1]. The rest of this article is focused purely on the group mentioned very first. Our explicit research aim is to develop a portable mixed-initiative domain-independent (multimodal) dialogue manager coming along with a user-friendly domain dialogue editor. The manager will be a part of a virtual navigation system. Our motivation for this domain is the elimination (or at least repression) of the driver's need to look at the graphical display of its car navigation during the ride (traffic safety should not be threatened). However, in order to test the manager capabilities properly, it is planned to employ it still in another domain – the train timetable. But this second dialogue system is currently not our primary focus – it is the manager accomplishment and its successful application to car navigation domain.

In the following, we shortly describe grammar-based management approaches, their advantages as well as drawbacks and, as next, we move the attention to the approach of our dialogue manager, its overview and detailed descriptions of currently existing capabilities. The explanation is augmented with some examples for better understanding and demonstration as well.

2 Grammar-Based Management

Grammar-based dialogue management group contains a lot of approaches of different complexity, among which one can count in the state-based methods (sometimes also referred to as finite state-based) and all of frame-based variations (most generally, those employing flat and hierarchically nested frames). According to [2], all approaches involved may be considered as equivalent, and, moreover, transformable to a finite state automaton using Schank's planning script.

2.1 Finite State-Based Approach

This approach is based on existing formalism – a finite state automaton. Therefore, it is not unusually referred to as a transitional network approach [3], because it can be thought of as a weighted directed graph, where every state (node) represents a system utterance. The transition to another state (node) is conditioned with a corresponding user's utterance matching one of edge values coming out from a current state. From a developer's point of view, a lot of integrated environments have been evolved – a very well known one of them is the Rapid Application Developer, a part of CSLU Toolkit [4], [5], which enables the dialogue to be created using simply dragging and dropping icons on the screen.

An essential advantage of this kind of management is its simplicity and a highly straightforward design capability. However, on the other hand, there are standing attributes like the lack of flexibility and hard applicability to other domains [1]-[3], [6]. Additionally, trying to get over the inflexibility, a state explosion may arise. A developer also encounters an unpleasant situation when getting a confirmation process involved – generally, every information needs to be confirmed by the user separately. Moreover, he is not given the possibility to correct himself (after a misrecognized user input, the system generally moves to another state). As a solution to this pitfall, a special key-word, for example "Back", may be considered – then, corrections can be achieved using an "undo" operation [7].

Regarding its shadow sites, the finite state approach has a very constrained area of applicability. According to [8], it is best suited to "applications in which the interaction is well defined and can be structured as a sequential form-filling task or a tree, preferably of yes/no or short answer questions."

2.2 Frame-Based Approach

As seen above, the pure state-based management is very restrictive one because of all its disadvantages coming along. The frame-based management reflects most of them and provides solutions. Here, the basic construction asset is a frame (sometimes also referred to as entity, topic or template, etc.) consisting of a set of slots. For controlling the dialogue flow, the system needs to select one of empty (generally unsatisfactorily filled) slots. For example, in VoiceXML, a XML-based language for creating voice response application, such algorithm is called the Form Interpretation Algorithm (FIA) [9]. To get the user aware of what slot the system has chosen a prompt attached to that slot needs to be sent to an output module. Therefore, the purpose of frames is to cumulate the information gathered from the user. Traditionally, a slot is assigned a set of event handlers instructing what actions the system needs to carry out when certain situations arise during the conversation. Back in VoiceXML, such events are "no-match" (the user's response is entirely out of acceptable utterances) or "no-input" (the user kept silent for a certain period of time).

Employing the frame-based management, the dialogue becomes more flexible in comparison to the previous approach because the possibility to take initiative during the discussion is held not only by the system but, instead, it is distributed between both partners [8] – the so-called mixed initiative. The scenario of mixed initiative dialogues is nearly the same in every case. At the beginning, the user makes a suggestion what he/she would like to talk about. However, a complete demand is provided very seldom or is not recognized properly, which implies the reason why the system takes the initiative and asks the user additional questions to obtain the missing necessary information.

A wide variety of frame types has been developed. The original idea of flat frames (VoiceXML) has been overcome with hierarchical (or nested) frames. Moreover, other case-based approaches emerge, for all of them let us remind the E-Forms present in WHEELS [10]. According to [3], the frame-based management is often involved in information retrieval systems – traveling, financial or timetable services. Still, because of simplicity, its pure version cannot be used in more complicated tasks [11].

3 Dialogue Manager Approach Description

The dialogue manager being currently developed at our department derives from its previous multimodal version [12] employing pure flat frames. Conceptually, this previous version was evaluated to have context and history as weakest parts, too much simple approach was the reason. The manager was applied in an experimental car navigation system domain, as well as the upcoming will. The reason why we have decided to remake it is that it did not seem to provide algorithms strong enough for a generally wider spectrum of collaborative tasks – a language model of a flat frame-based system cannot provide a necessary flexibility, because the user to be able to refer a desired system frame (task) must utter a whole particular phrase, which happens seldom, implying in the final, the novice user to have to follow exactly predefined utterances. This way, the conversation reduces to state-based model when going through the menus. However, more common is an incremental demand (as our observations show), where users rather than to express the whole command, try to explore the system step by step, usually beginning with a sentence containing a key verb ("navigate", for instance).
Currently, the new dialogue manager can cope with disambiguation and history creation and exploitation. However, it still lacks some core functionalities (as for example confirmations, corrections and subdialogues dealing). These will be accomplished very quickly, however, we have firstly focused our attention to the modules where the ancestor version of the manager seemed to have drawbacks – the context and the history modules. Moreover, instead of flat, hierarchical (nested) frames were necessary the dialogue manager to be able to deal with, enabling the users' incremental exploration.

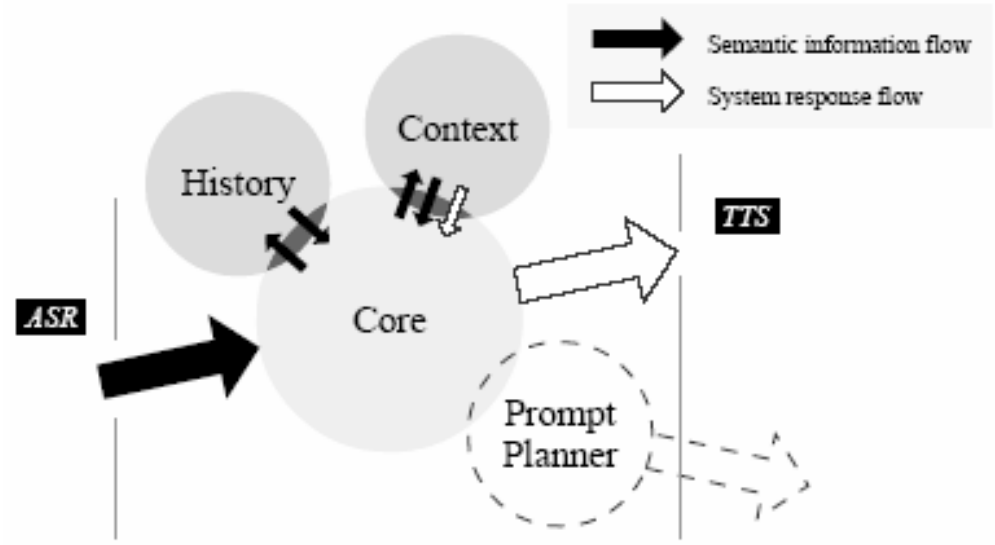

Figure 1 – The manager overall structure consists of three modules, the fourth is still to be accomplished

The manager current overall structure (see Fig. 1) consists of three modules:

- *Context* – a module maintaining a current model of a dialogue (for detailed description see below),
- *History* – a "memory" of a dialogue (see below),
- *Core* – main module directing both of previous ones, and interpreting current model of a dialogue.

The fourth, *Prompt Planner*, is still to be accomplished, and should enable the manager to produce more natural prompts employing common human language phenomena, as for example ellipsis. The manager repeatedly carries out three tasks:

- as soon as user's utterance semantic information is retrieved, it undertakes integration procedure (through History module into Context),
- when integrated, the Core initiates interpretation of current state of Context resulting in a system response, augmented with a response semantic information,

- finally, semantic information of the system response is integrated in the same way as the user's one.

This is manager's load in brief, more detailed description follows below.

3.1 Context Property

This section is rather than purely on the Context module focused on the context approach as a whole. As mentioned above, the context deals with hierarchical frames. This approach was chosen not only because it seems to be a promising way of frame-based management [10], but, additionally, because it enables the users to explore the system incrementally, thus allows more natural information representation in comparison to flat counterparts.

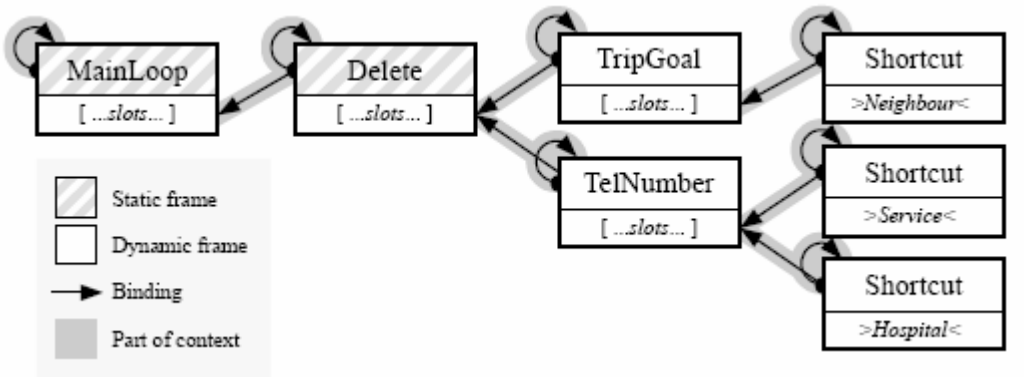

Figure 2 – A hypothetic context contains static frames MainLoop and Delete, dynamic frames TripGoal and TelNumber and three instances of another dynamic frame Shortcut containing different values

For the upcoming text, let us stick to the navigation domain and consider a hypothetic context containing a situation, where the driver wants to delete at once two addresses and one number stored under different shortcuts in the system (see Fig. 2). To make it possible, several types of frames must be defined – *MainLoop* (a top-frame constantly present in the system to ask the driver to begin a task), *Delete* (a frame asking for and maintaining what should be deleted and executing this demand making changes in the domain world), *TripGoal* and *TelNumber* (two frames asking for and maintaining system shortcuts), and *Shortcut* (a frame containing the information about a particular predefined system shortcut). According to the situation described, the TelNumber frame in the figure above consists of two Shortcut subframes. The frame-subframe relation is expressed using directed bindings. For the purpose of History module implementation simplicity, the context is made up of bindings only, implying every frame to be represented in it as a reflexive binding. The context contains a given frame if it contains its reflexive binding. In our approach, we define two general types of frames – dynamic and static, respectively. The first mentioned ones are expected to be used as information containers only (TripGoal, TelNumber and Shortcut), whereas the second ones are intended to be key frames and hold additional actions as well (MainLoop and Delete). Additionally, every frame contains a slot counter and a message queue, both are processed during the context interpretation.

In the context interpretation, it is necessary to carry out two essential operations: firstly, find an unsatisfactorily filled slot and evaluate its prompt, and secondly, integrate an incoming user's response semantic information. The finding problem is resolved very easily. Every frame is in the design phase assigned a priority (the nested a frame is, the lower priority it holds). The manager begins to process the highly prioritized frame queue containing a message. If a FRAME_INTERPRET message is popped, a slot addressed by the slot counter is evaluated. If it misses a value, then unsatisfactorily filled slot has been found and appropriate prompt is formulated. The same situation arises if it misses a subframe. If it contains a series of subframes, the searching problem is recursively transmitted to them.

Finally, even if the recursion did not find any slot and the message queue is empty, then the second highly prioritized frame undertakes this procedure. The manager ends the interpretation, if there is no frame with non-empty queue.
To demonstrate our integration problem resolution, let us focus back to the hypothetic situation of deleting some shortcuts. Now, consider that as soon as the system asks for deletion confirmation, the user augments his demand with "And the Cottage shortcut too, please" resulting in the ASR (Automatic Speech Recognition) to produce the following semantics:

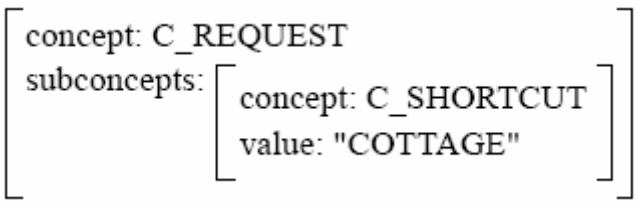

Figure 3 – The semantics for the sentence “And the Cottage shortcut too“

First in the integration process, the manager tries to transform the provided semantic information into a set of integration trees covering all meaningful hierarchical combinations of frames (currently existing in the context as well as the non-existing ones, i.e. entirely new). The process description is expressed in the following steps:

1. For each elementary semantic information find and maintain all possible paths through the frame hierarchy. Here, the frame Shortcut containing “Cottage” can be located either as a TripGoal subframe, or TelNumber subframe, i.e. two paths are found.
2. Merge groups of paths starting and ending in identical frames into one path. Here, both path start and end in identical frames, and are, therefore, merged in one, still regarding the choice between TripGoal and TelNumber.
3. Combine paths into a set of trees. Evaluate them according to different aspects of their nodes, like whether a particular frame is new or currently present in the context, static or dynamic, or whether the binding between related nodes is a part of a path to a slot, which prompt has been formulated as last, etc. Here, only one path exists, therefore only one tree is produced, and its evaluation is needless – there is no other one to compare it with (see the next step).
4. Select a tree having the highest evaluation. If there exist more than one, select the first of them (more sophistical strategy is still to be devised).

For our semantic information, this process results in the integration tree depicted in Figure 4:

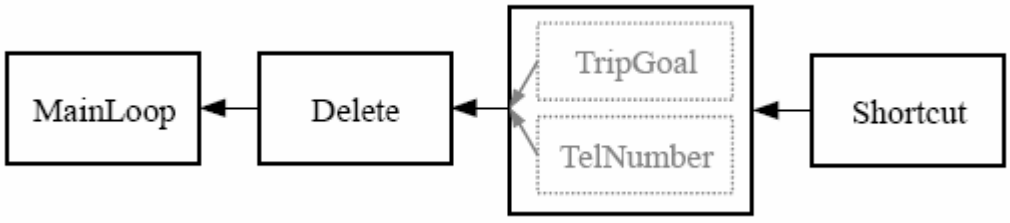

Figure 4 – The integration tree of user's sentence “And the Cottage shortcut too, please”. Its third node is made up of two subtrees

This tree now dictates the integration method. Starting in its root, the MainLoop and Delete frames – both currently exist in the context, therefore, neither of them will be recreated. In contrast, although three Shortcut frames exist there, a fourth will be created because it holds an unique value. However, its superframe (“third” node) is ambiguous in the tree, reflecting the location of ambiguity in user's utterance – as mentioned above, the shortcut Cottage may be conceived either as a trip goal shortcut, or a telephone number shortcut. Thus, the dialogue manager formulates a clarification question and binds the new Shortcut frame to an internal

auxiliary static Disambiguation frame. As soon as the user utters a resolution (for example “I meant a trip goal”), the Disambiguation frame has gathered all necessary information and interconnects both frames. As next, it disappears from the context.

3.2 History Property

In this section, the history approach will be presented – again instead of pure History module description. History structure (inspired by [13]) was designed with respect to an easy implementation of manager upcoming extensions (see "Future work" at the end).
Back in the context, we define a frame to be "*sealed*" If the following applies at once:

- the frame has filled and confirmed slot value (if any),
- every slot in the frame has acceptable amount of sealed subframes bound,
- bindings between the frame and all its sealed subframes are confirmed,
- there are no unsealed subframes bound.

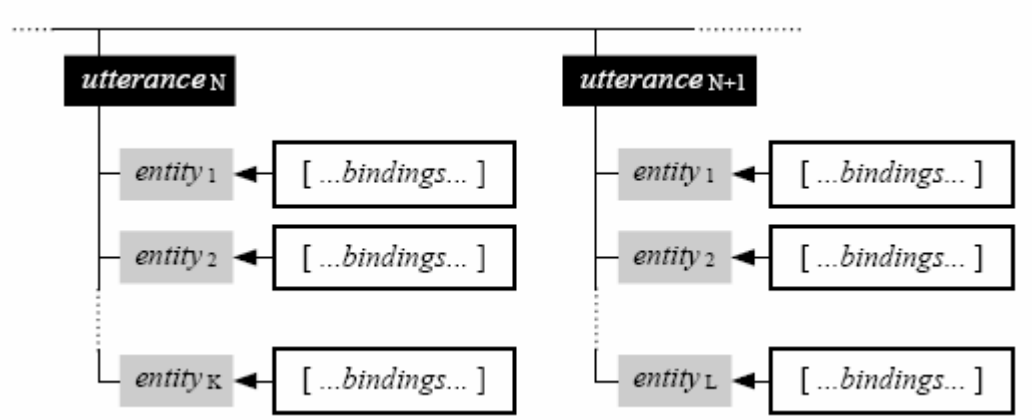

Figure 5 – The history structure consists of references to particular entities implying from user's or manager's utterances

We perceive the dialogue history as a storage of sealed frames, shortly *entities*. For example, one entity is a set of TelNumber and both Shortcut frames augmented with all interlaying bindings (see Fig. 5). As mentioned above, the context is made up of bindings only, benefiting in History module implementation simplicity. Here, the simplicity lies in entities made up of bindings only, as well (see forth).
Every time the Context module integrates incoming semantic information, the History module starts searching for newly emerged entities. If any found, it stores them ordered from the concrete to the general ones (from Shortcut to MainLoop, for example) in a new structure called *utterance*, which is initially empty. For all entities which it holds applies, that they imply from a particular user's utterance (which may be a confirmation, for example).
The inverse operation, reading from the history, occurs implicitly, which states for as soon as the ASR module provides a semantic information – any. The semantic information is transmitted to the History module. The History module treats it as an entity description and tries to find a match. If unsuccessfully, it sequentially begins to drill into the information structure and repeats the reading. Otherwise, if successfully, two operations are needed to be taken. Firstly, the original semantic information must be replaced with a particular entity semantic information *reconstruction*. Secondly, the History module needs to "remember" this successful reading. The information flow diagram (see Fig. 6) introduces more clarity in the semantic information processing.

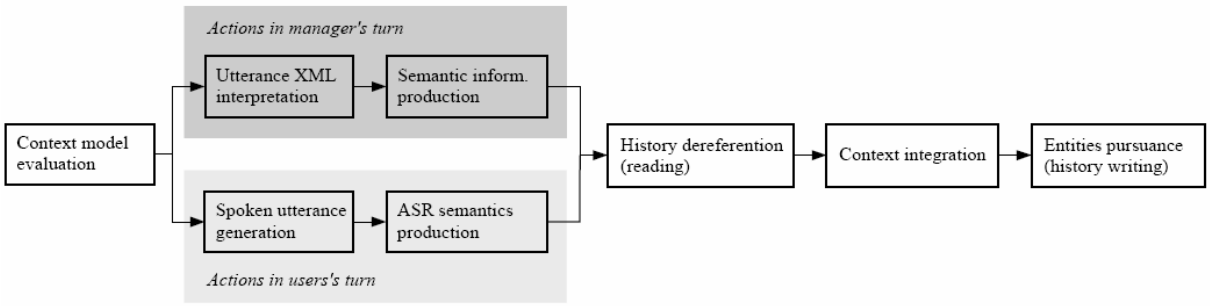

Figure 6 – Semantic information processing diagram branched for user's and manager utterances

The dialogue manager is prepared to deal with a "history shifting" as well, i.e., processing utterances similar to "What about the *previous* train?" (in a train timetable domain). It is achieved perceiving the actual context as the "history of right now" and exploiting the information in successful readings stack for the history searching continuation.
However, although the whole semantic information processing may feel as unwieldy and lacking flexibility (it does not transform the given semantic information into any internal structures), it provides enough robustness to make possible the system utterances to undertake the same way of dealing as the user's ones. In fact, the system prompt is tagged which helps to convert it into a semantic information, which is, in turn, confronted with the history reading, context integration and finally history writing, indeed. This is present because the system is not expected to only interpret (read) the current context information, but instead, it may introduce entirely new one as well (inferring from database etc.). Thus, both the user and the system are given the possibility to make changes in the context, which reflects the mixed initiative and the collaborative behavior, respectively.

4 Future Work

As mentioned above, the manager is not completed yet, instead, some functionalities in the core are missing. The correction and confirmation capabilities – both should be accomplished in the manner of disambiguation, i.e., "little" static frames not observable from the final developer's view. This approach seems to be clear and found an inspiration in McGlashan's goals [14]. However, more complicated challenge is expected to be the appropriate semantic information design, which the ASR module should produce. It is not clear whether one semantic system would be enough for both of them. Such information might look like:

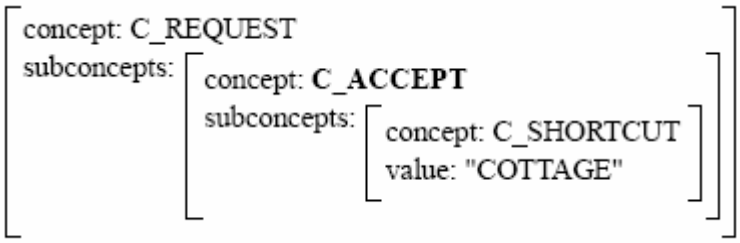

Figure 7 – One semantic system might cover both corrections and confirmations arising during dialogue

Last but not least, we want to augment the History module with enabling it to accept sets of entities instead of one entity at a time. The user would be offered the possibility to refer to a particular entity within a set by simply describing it, for example as "the second". However, another disambiguation problem has to be resolved – "the second" may refer either to an entity or a date (the second of May) [15].
Apart of missing functionalities, we also need to augment the manager with "old" existing capabilities of its ancestor [12]. Among them, the user's initiative restriction, achieved using

so called interaction modes, can be counted in. The manager switches to a more restrictive mode when notifying a stagnation of dialogue flow. The mode approach found a motivation in [8].
Currently, the manager is written in ECMA-Script (which satisfies the demand of portability), however, in the future, we would like to migrate to another platform. We have not decided about a particular one yet, our favorites are Java and Flash. We tend to the second one, not only because of its strong multimedia presentation capabilities, but easy – to create user interface design as well. On the other hand, we see the local in- and out- communication in Flash as the biggest drawback, which Java is free of.

5 Conclusion

We are on a long-term development hoping that our effort will result in a portable extendible domain-independent (multimodal) dialogue manager. In this article, we have presented its inner structure, context and history approaches, which are still not completed yet, however, seem to be on a good way to achieve our mentioned goal.

References

[1] Churcher, G. E., Atwell, E. S., Souter, C.: "Dialogue Management Systems: a Survey and Overview". University of Leeds, School of Computing Research Report, 1997.

[2] Wilks, Y., Catizone, R., Turunen, M.: "Dialogue management: State of the Art Papers". In: COMPANIONS Consortium, 2006. Available at: http://www.companionsproject.org/downloads/Companions_SoA2_Dialogue_Management.pdf.

[3] McTear, M. F.: "Modelling spoken dialogues with state transition diagrams: experiences with the CSLU toolkit". ICSLP, 1998, paper 0545.

[4] McTear, M. F.: "Using the CSLU Toolkit for practices in spoken dialogue technology". In: MATISSE, 1999, pp. 113-116.

[5] Sutton, S., Cole, R., de Villers, J., Schalkwyk, J., Vermuelen, P., Macon, M.: "Universal speech tools: The CSLU Toolkit". In: Proc.of the ICSLP, 1998, pp. 3221-3224, Sydney, Australia.

[6] Melichar, M.: "Template driven dialogue management approach in the framework of multimodal interaction". PhD. thesis proposal, EPFL Lausanne, 2005.

[7] Araki, M., Kaga, A., Nishimoto, T.: "Comparison of 'Go back' implementations in VoiceXML". In: Proc. of ISCA workshop on error handling in spoken dialogue systems, 2003, pp. 31-34.

[8] Levin, E., Narayanan, S., Pieraccini, R., Biatov, K., Bocchieri, E., Di Fabbrizio, G.: "The AT&T-DARPA communicator mixed-initiative spoken dialog system". In: ICSLP, 2000, vol.2, pp. 122-125.

[9] W3C, Voice Extensible Markup Language (VoiceXML), Version 2.0, 2004. Available at: http://www.w3.org/TR/voicexml20/.

[10] Cenek, P.: "Hybrid dialogue management in frame-based dialogue system exploiting VoiceXML". PhD. thesis, Masaryk University, Brno, 2004.

[11] Bui, T., H.: Multimodal Dialogue Management - State of the Art, CTIT Technical Report series No. 06-01, University of Twente (UT), Enschede, The Netherlands, 2006.

[12] Matoušek, V., Nestorovič, T.: "Hlasová komunikace s navigačním systémem automobilu". In: Proc. of Int. Conference "Navage", Prague, 2006.

[13] Zahradil, J., Müller, L., Jurčíček, F.: "Model světa hlasového dialogového systému". In: Proc. of Znalosti, 2003, pp. 404-409.

[14] McGlashan, S.: "Towards multimodal dialogue management". In: Proceedings of Twente Workshop on Language Technology, vol. 11, Enschede, The Netherlands, 1996.

[15] McGlashan, S., Fraser, N., Gilbert, N., Bilange, E., Heisterkapm, P., Youd, N.: "Dialogue management for telephone information systems". In: Proc. of the International Conference on Applied Language Processing, Trento, Italy, 1992.