

# A Frame-Based Dialogue Management Approach

Tomáš Nestorovič

University of West Bohemia in Pilsen, Department of Computer Science and Engineering  
Pilsen

nestorov@kiv.zcu.cz

## Abstract

*This paper is focused on our approach to hierarchical frame-based dialogue management. As we show, even when dealing with this simple technique, the manager exhibits complex skills, as for example acceptance of references to historical entities or maintenance of context causality (by utilizing application of a journaling system). Our research goal is to create a generic and easy-to-use manager. At the end of this paper, future work is outlined, as the manager is still under development.*

## 1. Introduction

Dialogue management focuses on machine reasoning, in particular, in finding the best machine response to a user's utterance on a basis of given circumstances within the dialogue. Since the beginning of research in this field, many approaches based on different backgrounds emerged: from finite state machines to intelligent agents and recently Markov models. However, we decided to follow the way of using frames in our approach. Not only due to the fact that it covers many tasks (regardless it is relatively simple technique), but moreover, because it seems to be a promising way of dialogue management [1].

The rest of the paper is divided as follows. First, we describe the term *frame* and summarize it briefly (section 2). Next, we move the attention to our approach and describe manager's context and history modules (section 3). Finally, planned future work is outlined and paper concluded (sections 4 and 5).

## 2. Overview of frame-based management

Frame-based management attempts to reflect most of the state-based approach disadvantages (inflexibility above all). Here the basic construction asset is a frame (sometimes also referred to as an entity, topic, template, etc.) consisting of a set of slots.

To control the dialogue flow the system needs to select one of empty (in general, unacceptably filled)

slots. To inform the user about which slot was chosen, an appropriate prompt needs to be uttered by the system. The prompt is usually attached to a slot and invoked as a reaction to the "value-needed" event. Traditionally, additional event handlers are assigned as well, instructing what actions the system needs to carry out when these events (situations) arise during the conversation. However, the main purpose of a frame still remains to accumulate information gathered from the user.

For a simple demonstration, we can refer to the VoiceXML. Its algorithm for a slot selection is called the Form Interpretation Algorithm (FIA, described in [2]), and its frame implementation distinguishes among several events – no-input, no-match, help-asked and "value-needed" (<prompt/> element).

A variety of frame types evolved during research. All of them are an extension of the classical flat technique. The wide well known one is the so-called E-Frame, employed in the WHEELS car system [3]. The extension lies in giving every slot a priority and selecting them accordingly in descent order. A complete overview of different approaches to frames may be seen in [1].

Under frame-based management, a dialogue gets more flexible – a possibility to exhibit initiative during the discussion is granted not only to the system, but instead it is distributed between both partners [4] (the so-called *mixed initiative*). The scenario is always the same: at the beginning, the user provides an incomplete demand (due to his/her unfamiliarity with the system or speech recognition errors). To satisfy the demand, the system takes the initiative over and elicits additional information. Therefore, the frame-based management is mainly accommodated in information retrieval systems (traveling, weather or timetable services) [5]. However, due to its relative simplicity, it still is hard to apply on domains with complex information structure [6].

## 3. Application of frames in our approach

Due to the fact that our dialogue management approach employs hierarchical extension to basic flat

frames, many algorithms solving particular issues are needed. They will be described in the next sections, however, now on to a top-level description.

The manager is divided into four collaborative modules (**Figure 1**). The *Context* module maintains information about the current dialogue (active frames and relations between them are stored here). The *History* module serves as a source of historical data – it provides a basis for dereferencing/disambiguating user's utterances (for example “*the previous train*”). The *Core* module controls the behaviour of both modules – it interprets the current state of the dialogue and coordinates information stream flows. Additionally, the Core produces CTS (Concept-to-Speech) utterances descriptions and feeds them into the *Prompt Planner* module. Here, we will apply natural speech paradigms to the descriptions – however, this module still remains unimplemented.

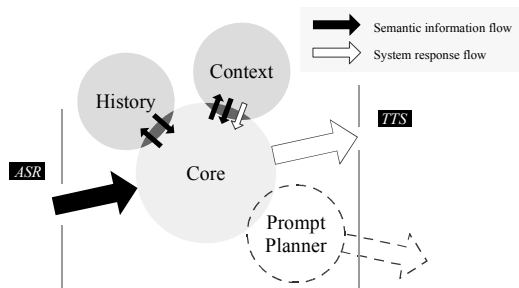


Figure 1. Manager topology and its information flows

From the top-level point of view, the manager loops in a cycle “system prompt – user's answer.” All related actions are depicted in **Figure 2**: the semantic information received from the ASR (Automatic Speech Recognition) module (1) needs to be disambiguated based on given dialogue history (2). Next, it is integrated into the current task context (3), and finally, the new context is interpreted and system prompt produced (4). Note that the system utterances follow the same way of processing as the user's ones do (5-8). The reason is that even the system may introduce new information that needs to be anchored within given context and recorded to history (“*The next train leaves at 15:00*”).

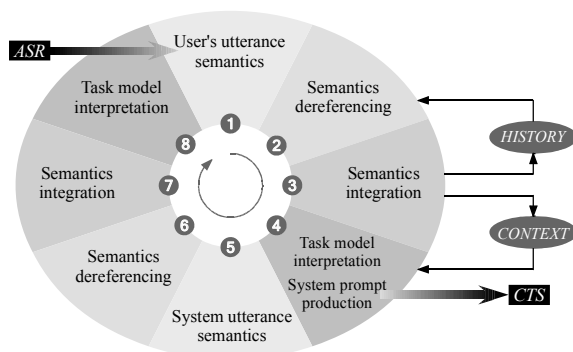


Figure 2. Action loop performed by the manager

The manager deals with several key situations which may arise during the conversation:

- introduction of a new concept by the user,
- corrections (not only of current concepts, but of relations between them as well),
- confirmations (of context fragments), and
- recalling information from the History module.

Solutions to these issues are the following. Every fragment of semantics is a priori supposed to either refer to historical data (d), or to introduce new information (a). Situations (b) and (c) are perceived as very similar ones – in particular, we deal with confirmation as with a special case of correction. Hence, the input semantics model gets more simpler as it is possible to represent both of them using the same semantic element:

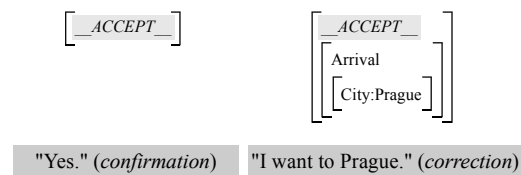


Figure 3. Confirmation is a special case of correction

### 3.1. Frames and relations

As mentioned above, both frames and relations between them are parts of the Context module. Our notion of a frame is quite “concept-like” since it may hold single domain information at most. Hence we design a frame to handle a specific concept type (for example *Time* concept). Additionally, our implementation of frame is equipped with a message queue containing demands for actions to be performed, and a journal for a (cascade) roll-back operation.

*Relations* express how active frames are bound to each other. Templates for possible relations are defined in the manager's editor environment, and in run-time they are constructed in accordance with these templates. Generally, the Context module contains two types of relations – *standard relations* (to maintain relevant bindings) and *disambiguation relations* (to express a detailed description of a particular frame).

Note that the Context consists of *relations only*, i.e., every frame is within the Context registered using a *registration relation* (a special case of the standard relation). We found this approach of Context very useful, as operations with historical entities get simpler; see below.

### 3.2. Semantics integration

Let us stick to the Context module description and skip the process of semantics dereferencing for now, we shall return to it later. Suppose that the input

semantics went through dereferencing and is about to be integrated into the current context model. The basic idea is a production and evaluation of all possible unification trees. The best evaluated one is then used as a template for the semantics integration. Let us demonstrate the algorithm and consider a simple fragment of a timetable domain (**Figure 4**). In the model, solid lines represent standard relation templates and the dotted ones disambiguation relation templates. Additionally, consider the user uttered “London” when the system asked for arrival city (**Figure 5** represents current context where grayed is a path to the currently interpreted Arrival frame).

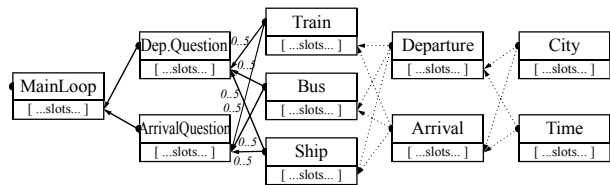


Figure 4. Timetable domain model

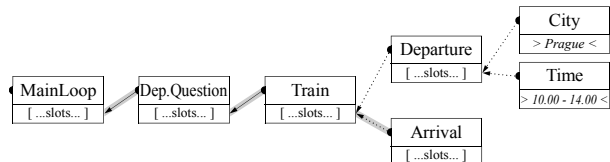


Figure 5. Current context model

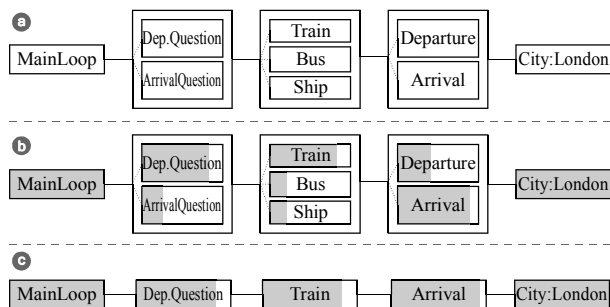


Figure 6. Integration process

The algorithm for integration is as follows.

1. Let  $F$  denote a set of active frames within the Context and  $D$  a set of frame templates within the domain model. Then for every *elemental* semantic information find a collection of all possible integration paths within  $F \times D$ .
2. Join “similar” paths together. Paths are similar if they end in the same elemental semantics. In case they differ in some part, these parts are made parallel sub-paths in the joined path. In our case, all paths are similar because we have only one elemental semantic information (London), see **Figure 6a**.
3. Build all possible trees upon joined paths and evaluate them (**Figure 6b**). There are

six criteria for evaluation, as for example whether a particular relation does exist or not, or whether a particular frame is on the path to the one interpreted as last.

4. Select the best evaluated tree. Try to perform implicit disambiguation according to evaluation (**Figure 6c**; the Train frame beats Bus and Ship – a train is being discussed in the current context, **Figure 5**). Next, process the disambiguated tree as a LISP program structure. The basic interpretation may be affected by system semantic element (correction, for instance), however, this is not the case in this example.

### 3.3. Dialogue stack

The manager maintains currently discussed “topics” in a form of a *stack*. This approach found an inspiration in Grosz and Sidner's framework [9]. However, in comparison to it, our stack *topics* are frames themselves, not abstract descriptors. There is another deflection: an absence of interruption detection, i.e. absence of a capability to detect a discussion topic shift – to make a change, the user is supposed to utter an explicit correction demand, for example “No, I want to get there by ship.” Therefore in our approach, the stack plays a role of a purely passive component of the manager, designed to collect

- newly emerged concepts (i.e. frames) in the discussion,
- frames with an updated content, and
- currently discussed frames.

Frames are stored in the stack as long as they meet at least one of the conditions above, otherwise they are popped out. To be more specific, a frame is popped out of the stack if

- user's correction affects its existence in the context,
- user's utterance does not answer a question asked by a system disambiguation process (see below), or
- frame is, from the system's point of view, no longer needed to be discussed, i.e., its interpretation is completed.

### 3.4. User's corrections

The ability to make corrections in a current model must be an essential part of every manager. It is due to ASR (Automatic Speech Recognition) errors arising during an interaction, as the ASR module serves as the weakest part of every dialogue system [12]. However, sentences like “I don't want Y, but X instead” or “X, not Y!” provide semantics distinguishable by the ASR only, but say nothing about user's intentions. It is the manager's task to guess them.

Our approach to this issue is a restriction to a last manipulation with a particular frame. We distinguish between two types of manipulations – *construction of frame*, and *its use as a super-frame* for another one (Train is a super-frame for Arrival). Hence, if Y has not been used as any super-frame until now, user's correction “I don't want Y” is perceived as a rejection of Y. Similarly, if the last manipulation was making Y a super-frame for Z, then by uttering the same sentence user is rejecting the relation between Y and Z, not the existence of Y itself.

Our current approach expects system's prompts to inform the user about recognized concepts as soon as possible. For example, instead of sentence “Which time do you want to leave?” a production of “Which time do you want to leave by train from Prague?” offers a user a possibility to make instant changes of transportation means or departure city. Better composed sentences help users to feel more confidently while interacting with a spoken language dialogue system [10].

The manager is able to infer an invalidation of related parts of the context on the basis of one particular change. This mechanism is called a *causality consistence mechanism* (its description follows). Using it, information dependent on changed fragment disappears from the context and the system is forced to re-elicite it. However, we would like to extend the current “correction/causality” mechanisms with the possibility of recovering last confirmed fragments. An open question remains whether this introduces rather more confusion than help.

### 3.5. Causality consistence mechanism

As mentioned above, the mechanism helps to keep the current context in causal consistence. We decided for a *distributed approach*, i.e., every part of the context (frame and relation) maintains its own agenda of what operations it was involved in. Compared to a centralized approach, the distributed one offers more flexibility regarding a roll-back.

We distinguish between three types of operations: information *reading* and *writing*, and *interpretation* of a frame. Entries of these operations are inserted into particular frame journals, and in a case of reading, also into journals of relations the operation covers as well. Information *changing* causes a roll-back of journals. For an illustration of a roll-back, let us stick to the timetable domain and consider a context fragment depicted in **Figure 7**. Here, the system uttered a particular transportation means in Q/3 (DepartureQuestion frame, slot 3; “The next ship from Delft to London leaves at 10:15”), and an additional back-end reading was performed in S/1 (Ship). Journal contents for each of frame are depicted in **Figure 8**. The figure also serves as a trace of the interpretation algorithm as time is

involved. Consider the user changed the City of departure from Delft to Oslo. Now, neither of the previous readings  $R_1$  and  $R_4$  is valid and the journal of the City of departure will be rolled-back up to  $R_4$ . However, the rolled-back fragment still remains stored in a REDO part of the journal. The same must be done with both readers (Ship and DepartureQuestion), temporarily losing the Arrival branch (during the reinterpretation it is recovered utilizing the REDO). As for the Departure, it remains unaffected. To keep track of what parts of the context were modified, notification messages with D/0, S/0 and Q/0 are sent to Departure, Ship and DepartureQuestion, respectively. By this, the model reaches the consistency and a new interpretation may begin. (Example continues.)

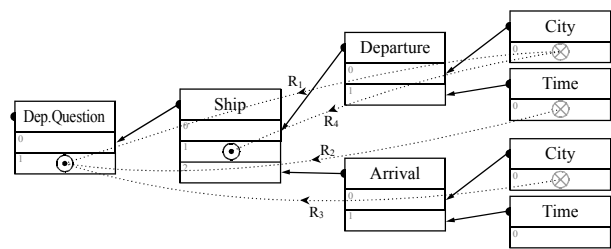


Figure 7. Readings within given context fragment

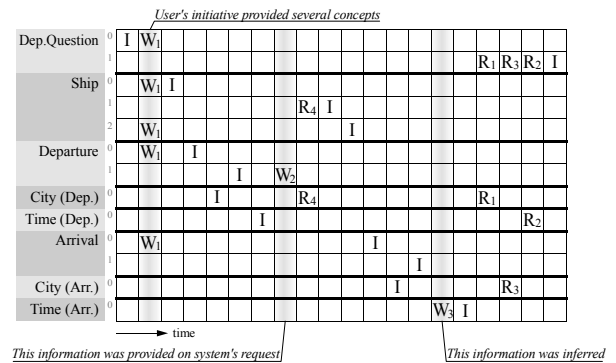


Figure 8. Content of frame journals in time;  $R$  represents a reading,  $W$  a writing and  $I$  an interpretation

### 3.6. Context interpretation

An essential goal of the interpretation is to search for newly emerged, updated or missing fragments of the context, and invoke their integration (see section 3.2), validation (proving they are true) or elicitation, respectively. As mentioned above, the behaviour of frames is modeled by message passing. To reach not only a flexible, but a collaborative interpretation environment as well, we additionally employ

- *the dialogue stack* (only the message queue of a frame on its top is processed as long as it is not empty), and
- *an interpretation token* (a frame which holds it may pass it over to one of its sub-

frames; the interpretation token is realized as a standard message).

In combination of both of these components the dialogue flow is managed on one side quite strictly, however, on the other hand it is still easily adaptable to new circumstances. For example, although the user provides new information (new frames are pushed onto the stack), from the manager's point of view, it may be *currently* irrelevant (until the frames do not hold the interpretation token, the manager *mostly* ignores them; mostly = except for necessary operations like disambiguation). If the information is currently really irrelevant, it will disappear from the stack (however, not from the context) and the dialogue will continue in accordance with manager's original plan. Note that this plan may be affected by user's corrections.

Let us continue and finish the “roll-back” example above. The interpretation starts with obtaining the messages, and hence, reinterpreting the City of departure. Next, it continues reevaluating S/1 and moves to S/2. Here, REDO part of the journal will be employed and the formerly lost branch recovered. Finally, the interpretation reaches Q/1 and a new prompt is generated – “*The next ship from Oslo to London leaves at 11:35.*” Note that the new time was obtained by searching in database, initiated by the DepartureQuestion frame.

### 3.7. History module

Now, as the Context module is described, let us return to the semantics dereferencing depicted in **Figure 2**. The structure of the History module consists of a series of previously used entities, similarly as proposed in [8]. We define an *entity* to be a set of relations (i.e., a fragment of context) which meet the following conditions.

- All information held in frames is confirmed.
- All standard relations are confirmed.
- Every frame content is acceptable (i.e., it does not need to be disambiguated any further).

The history is built automatically after semantics has been integrated. If a context fragment meets the conditions above, a set of entities based on this fragment is created. The process of generation starts with an entity containing the most concrete information and ends with the most general one – **Figure 9** demonstrates.

The dual operation, reading the history, is initiated implicitly, i.e., every incoming semantic unit is perceived as a reference to historical data. The process of dereferencing tries to take as big fragment of semantics as possible and match it against the most general historical entity found closest to the “present.” If a match is found, the entity is transformed into a semantics replacing the original

fragment in the input. However, the reading is a complex issue. For example in the reference “*the previous ship,*” first an entity expressing a ship must be found (in Figure 9 the above one), and once found, the “previous <entity>” must be dereferenced (in Figure 9 the below one). We approach this by introducing a *stack of pointers* to the history time line where successful dereferences were realized. Therefore, once the inner reference is resolved, the outer starts searching from the point the inner was satisfied either back or forward in the history (in our case marked with ♠ sign in Figure 9).

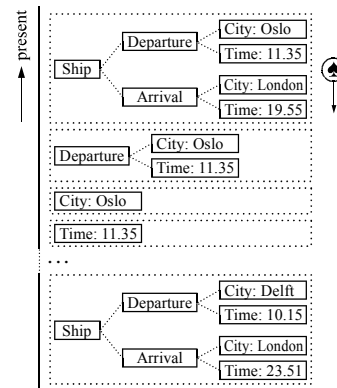


Figure 9. A fragment of History containing two “Ships”

### 3.8. Disambiguation process

Disambiguation is another key capability the manager must carry out. Our approach is inspired by McGlashan's one [11] and involves an ordered list of disambiguation questions related to a particular frame. The manager sequentially picks up questions to clear an ambiguous frame. For example, if a user wants an information about departure of a ship, there may be more than one ship in the database and it is not clear which one of them s/he meant. The proper list of sub-frames (holding the questions) is:

1. Departure.City
2. Arrival.City
3. Departure.Time
4. Arrival.Time

The order may be overridden (constrained) by items of a list related to any of super-frames (Departure-Question), however, this is currently only a vision and a matter of future, hence, it will not be detailed any further here. The current algorithm is as follows.

1. Build a database query and obtain results. If the number of results (*database instances*) exceeds the acceptable amount (for example 5 ships at most) continue in the disambiguation process. Otherwise, disambiguation is completed. The parameter relaxation process is currently omitted, however, we have a quite clear conception about it.
2. Pick up first yet *undiscussed* sub-frame in the list (Departure.City, for example).

Such frame must not have been provided by the user in the past. If all sub-frames in the list are discussed, the disambiguation process is completed.

3. Create artificially the frame selected in (2), and produce a question it holds. In fact, the frame needs to be created in order to be interpreted and the question produced.
4. Wait for the user's reaction and integrate its semantics.
5. After regaining the control, test if it was necessary to create the frame in (3). If not, remove it (it was unnecessary if the user did not answer the question, and hence, it now does not hold any information).
6. Repeat the process from step (1).

## 4. Future work

In this paper, we have omitted to describe a production of the system utterances. We currently employ a XML-based description of a sentence to first express the content itself, and second, to mark distinguished fragments. For example, “<q><concept ship><TheShipFrom><concept departure><r \_parent.#1/></concept><Leave/>...</concept></q>” is our current (simplified) description of “*The ship from Oslo leaves...*”. Hopefully, this approach will lead to the CTS output fed into the Prompt planner (**Figure 1**) which is currently not realized.

Also the manager lacks a confirmation process. However, because finding entities in the context depends on it, we simulate it on-the-fly. As a real solution, we propose an introduction of a special type of slot which question will be built with respect to information to be confirmed. This would enable the manager to automatically detect which fragments of context should be considered as believable after a user's positive answer is obtained.

## 5. Conclusion

The research goal we follow is to create a generic and easy-to-use dialogue manager. Our approach utilizes frames technique for context knowledge representation. In this paper, we presented and demonstrated broad scale of algorithms providing manager's particular capabilities. We found an inspiration for them in several of the cited sources. We adjusted well known approaches to fit our purposes of creating a manager with complex behaviour. In [1] we found a motivation for nested frames technique, from [2] we adjusted FIA to work recursively using message passing, [8] served us as a basis for historical entities processing we augmented with stack of pointers – another stack besides the one (partially) adopted from [9]; our disambiguation process is inspired by [11], however, we extended it to work in nested frames environment and will

continue on “reversible” version as well (enabling relaxation). Last but not least, we presented here our journaling system for keeping the context in causal and coherent state. Once the manager is finished, it will be applied in car navigation and timetable domains to thoroughly test its management skills. Its previous version [7] was applied in car navigation domain only. We expect to obtain far better results in this domain since the previous version employed flat frames only (extended with another features). Finally, we also would like to offer the manager as a free software on our website,<sup>1</sup> as the development seems to evolve promisingly towards our specified goal.

## References

- [1] P. Cenek, *Hybrid dialogue management in frame-based dialogue system exploiting VoiceXML*, Ph.D. thesis proposal, Masaryk University, Brno, 2004.
- [2] W3C, *Voice Extensible Markup Language, Version 2.0*, 2004. Available: <http://www.w3.org/TR/voicexml20/>
- [3] Meng H. *et al*, “Wheels: A Conversational System In The Automobile Classifieds Domain,” in *Proc. of ICSLP*, 1996, pp. 542-545.
- [4] Levin E. *et al*, “The AT&T-DARPA communicator mixed-initiative spoken dialog system,” in *Proc. of ICSLP*, 2000, vol.2, pp. 122-125.
- [5] McTear M. F., “Modeling spoken dialogues with state transition diagrams: experiences with the CSLU toolkit,” in *Proc. of ICSLP*, 1998, paper 0545.
- [6] 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.
- [7] Nestorovič T., Matoušek V., “Entwurf der Sprachkommunikation mit einem Car-navigationssystem und Ihre Implementation in der VoiceXML Sprache,” in *Proc. of Elektronische Sprachsignalverarbeitung*, 2006, pp 119-126.
- [8] Zahradil J., Müller L., and Jurčiček F., “Model světa hlasového dialogového systému,” in *Proc. of Znalosti*, Ostrava, 2003, pp. 404-409.
- [9] Grosz B. J., and Sidner C. L., “Attention, intention and the structure of discourse,” in *Computational Linguistics*, 12(3):175-204.
- [10] Yankelovich N., “How do users know what to say?,” in *Interactions*, vol 3, 1996, pp. 32-43.
- [11] McGlashan S., “Towards Multimodal Dialogue Management,” in *Proc. of Twente Workshop on Language Technology* 11, pp. 1-10.
- [12] Gustafson J., *Developing Multimodal Spoken Dialogue Systems - Empirical Studies of Spoken Human-Computer Interaction*, Ph.D. thesis, KTH, Department of Speech, Music and Hearing, 2002.

<sup>1</sup> <http://liks.fav.zcu.cz>