

Towards Flexible Dialogue Management Using Frames

Tomáš Nestorovič

University of West Bohemia in Pilsen
Univerzitní 22, 30614 Pilsen, Czech Republic
nestorov@kiv.zcu.cz

Abstract. This article is focused on our approach to a dialogue management using frames. As we show, even when dealing with this simple technique, the manager is able to provide a complex behaviour, as for example maintenance of context causality. Our research goal is to create a domain-independent dialogue manager accompanied with an easy-to-use dialogue flow editor. At the end of this paper, future work is outlined, as the manager is still under development.

1 Introduction

Dialogue management focuses on finding the best machine response to a user's (spoken) input. Many approaches to this issue emerged. They are based on different backgrounds ranging from finite state machines to intelligent agents. However, we decided to follow a way of using frames. Our aim is to implement and test a generic manager equipped with (relatively) uncommon way of keeping track of coherence of changing circumstances within a dialogue. This approach is based on application of a journaling system to the construction of frames. Last but not least, we also would like to offer the manager as a free software on our website¹ as the development progresses promisingly towards the specified goal.

In the rest of the paper, we first describe the term frame and try to summarize it in short. Next, we move to our particular approaches and describe manager's context and history modules. The paper is concluded with the planned future work.

2 Frame-Based Dialogue Management in Brief

Frame-based management attempts to reflect most of the disadvantages exhibited by the state-based approach (inflexibility above all). Here the basic construction asset is a *frame* (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 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, defining actions to be carried out when these events arise. However,

¹ <http://liks.fav.zcu.cz>

the main purpose of a frame still remains to accumulating information gathered from the user. A variety of frame types evolved during research – [1] provides an overview.

Under frame-based management, a dialogue gets more flexible – a possibility to exhibit initiative during the discussion is granted not only to the system itself, but instead it is distributed between both partners [2] (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 complete the demand, the system takes the initiative over and elicits additional information. Therefore, the frame-based management is mainly involved in information retrieval systems (traveling, weather or timetable services) [7].

3 An Example of Frame-Based Technique

Due to the fact that our dialogue management approach employs hierarchical extension to basic flat frames, many of 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 (Fig. 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) utterance descriptions and feeds them into the *Prompt planer* module. Here, we will add natural speech paradigms into descriptions – however, this module still remains unimplemented.

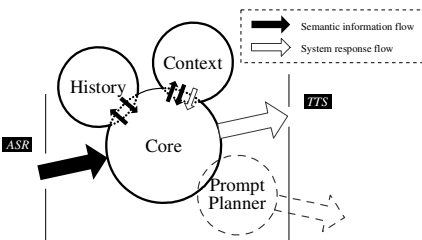


Fig. 1. Dialogue manager modules topology

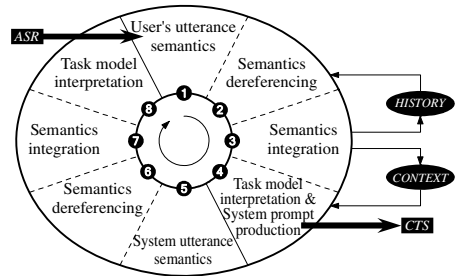


Fig. 2. Action loop performed by the manager

From the top-level point of view, the manager loops in a cycle “system prompt – user's answer.” All related actions are depicted in Fig. 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"). Additionally, the manager deals with several key situations arising during the conversation:

- introduction of a new concept by the user,
- corrections (of both current concepts and relations between them),
- confirmations (of context fragments), and
- recalling information from the History module.

Solutions to these issues are the following. Every incoming semantics fragment 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 – in particular, we deal with confirmation as with a special case of correction. Hence, the input semantics model get more simple as it is possible to represent both of them using the same element (Fig. 3).

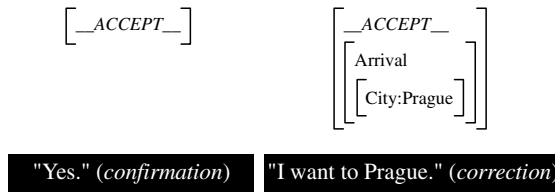


Fig. 3. Semantic information for confirmation (on the left) and correction

3.1 Frames and Relations between Them

As mentioned above, both frames and relations 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). A frame is additionally equipped with a message queue containing demands for actions to be performed on this frame, and a journal for a roll-back operation.

Relations express how active frames are bound to each other. Templates for possible relations are defined within the manager's editor environment, and in run-time they are constructed in accordance with these templates. The Context module contains two types of relations – *standard relations* (to maintain relevant bindings) and *disambiguation relations* (to express a detailed description of a frame). An example of a proposed domain structure may be seen in Fig. 4.

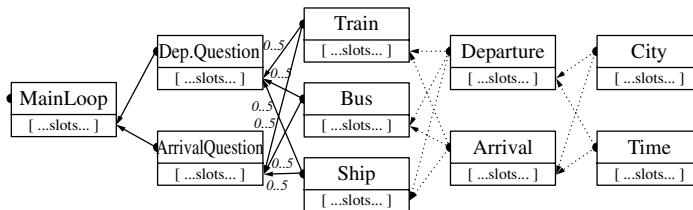


Fig. 4. Proposed structure for a timetable domain. Solid lines represent standard relation templates, whereas the dotted ones are disambiguation relation templates.

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

3.2 Integrating the Semantic Information into the Context

Let us stick to the Context module description and skip the process of semantics dereferencing (see Fig. 2), 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. Here, the basic idea is a production and evaluation of all possible unification trees. Only the best evaluated one is then used as a template for the semantics integration. The complete algorithm is described in the following four steps.

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 parallelly accessible in the joined path.
3. Build and evaluate all possible trees upon joined paths. Currently in our manager, 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 and process (interpret) it as a LISP program structure. The basic interpretation may be affected by processing *system semantics* (for correction, for instance).

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 [6]. However, in comparison to it, our stack topics are frames themselves, not abstract descriptors. There is another difference: an absence of interruption detection, i.e., absence of a capability to detect a discussion topic change – to make a change, 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.

3.4 Corrections Made from the User Side

The ability to accept corrections of a current context must be an essential part of every manager due to ASR errors arising during an interaction (the ASR module serves as

the weakest part of every dialogue system [7]). However, sentences similar to “*I don't want Y, but X instead*” 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 is best suited for system's prompts informing a user about recognized frames immediately. For example, instead of “*Which time do you want to leave?*” a production of “*Which time do you want to leave by train from Prague?*” offers a possibility to make instant changes of transportation means or departure city.

The manager is able to infer an invalidation of related parts of the context on a basis of one particular change. We call this mechanism 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-elic 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, the mechanism helps to keep the 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 two types of operations: information *reading* 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.

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 Fig. 5. 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). Frames' journal contents are depicted in Fig. 6. 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 R₁ and R₄ is valid and the journal of the City of departure will be rolled-back up to R₄. 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.)

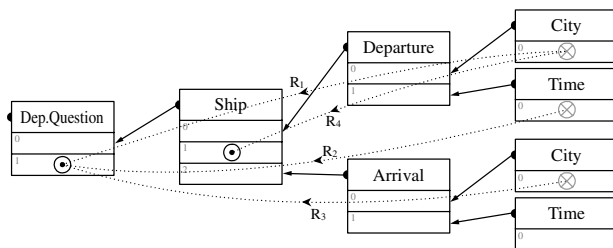


Fig. 5. Readings within given context fragment are dotted

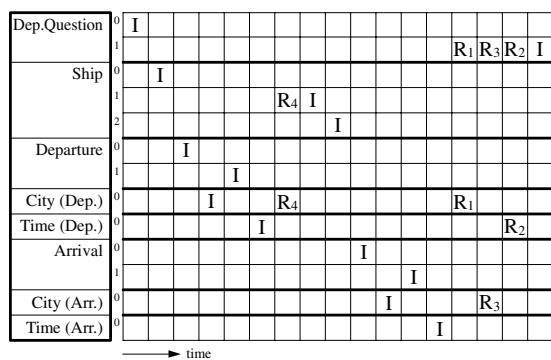


Fig. 6. Contents of frame journals in time; R represents entry with a reading, I with an interpretation

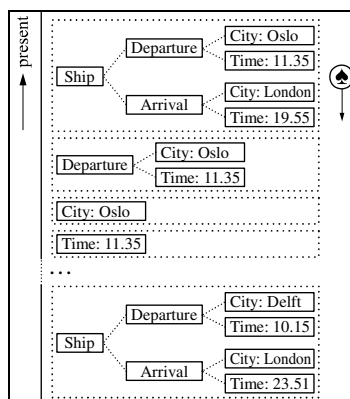


Fig. 7. History module content

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, 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, the information 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 Fig. 2. The structure of the History module consists of a series of previously used entities, similarly as proposed in [4]. 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 further disambiguation).

The history is built automatically after semantics has been integrated. If a context fragment meets the three conditions listed 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 – Fig. 7 demonstrates.

The inverse 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 Fig. 7 the above one), and once found, the “*previous <entity>*” must be dereferenced (in Fig. 7 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 Fig. 7).

4 Future Work

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

We also omitted to mention a disambiguation process – currently, it is in progress. We found an inspiration in the McGlashan's approach [5] consisting of lists of related topics (frames) which need to be discussed prior to a database query is performed.

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 fragments of context which should be considered as believable after a user's positive answer is obtained.

5 Conclusion

We are on a development of a domain-independent dialogue manager. It utilizes frames to represent context knowledge. 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, [4] served us as a basis for historical entities processing we augmented with stack of pointers – another stack besides the one (partially) adopted from [6]; we are inspired of the disambiguation process in [5], however, we would like to extend it to work “reversibly” 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 [3] was applied to 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).

References

1. Cenek, P.: Hybrid dialogue management in frame-based dialogue system exploiting VoiceXML. Ph.D. thesis proposal, Masaryk University, Brno (2004)
2. McTear, M.F.: Modeling spoken dialogues with state transition diagrams: experiences with the CSLU toolkit. In: Proc. of ICSLP, paper 0545 (1998)
3. Nestorovič, T.: Navigation System: An Experiment. In: Proc. of NAG-DAGA, paper 470, Rotterdam (2009)
4. Zahradil, J., Müller, L., Jurčiček, F.: Model světa hlasového dialogového systému. In: Proc. of Znalosti, Ostrava, pp. 404–409 (2003)
5. McGlashan, S.: Towards Multimodal Dialogue Management. In: Proc. of Twente Workshop on Language Technology, vol. 11, pp. 1–10
6. Grosz, B.J., Sidner, C.L.: Attention, intention and the structure of discourse. *Computational Linguistics* 12(3), 175–204
7. 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)