

General Agent-based Architecture for Collaborative Dialogue Management

Tomáš Nestorovič

Department of Computer Science and Engineering
University of West Bohemia in Pilsen
Pilsen, Czech Republic
nestorov@kiv.zcu.cz

Abstract—In this paper, we focus on our agent-based approach to task-oriented dialogue management. We present our deliberation process based on optimizing the length of dialogue. As an uncommon feature, the manager accommodates a two-layered structure for representing the dialogue context, i.e., user's intentions detection and beliefs management. At the end of the paper, we suggest future extensions to the architecture.

Agent-based dialogue management; system adaptability; dialogue systems; artificial intelligence

I. INTRODUCTION

Dialogue management focuses on finding the machine's best response, given a user interaction history. During the past decades, many approaches emerged. What they have in common is the aim to manage and elicit knowledge from the user within a dialogue, however, their theoretical backgrounds differ. Ranging from simple finite state machines through intelligent agents, and Markov decision networks, there is a wide collection of methods on how to implement a dialogue manager. However, we decided to follow an agent-based approach to manage a spoken dialogue. In our case it accepts domain data description and intention satisfaction plans (instructing how to reach a solution to a given domain task). The agent follows a scheme of the BDI architecture (Beliefs, Desires, Intentions) [1].

The rest of the paper is divided as follows. First, we outline manager's overall structure (Section 2). Next, we move on to our approaches to modules storing and coping with Beliefs, Desires, and Intentions (Section 3). The paper is concluded with scheduled future work and a brief summarization (Sections 4 and 5).

II. DIALOGUE MANAGER

The manager plays a role of a collaborative conversational agent. It consists of five modules (Fig. 1). The *Context* module maintains information about a current dialogue (having some Beliefs¹ and defining some Desires). The *History* module serves as a source of historical data enabling user's utterances to be disambiguated (“the previous train”). The *Strategy Selection* module recognizes familiar situations within the dialogue and determines a corresponding initiative mode to be selected for the agent's next response production. The *Core* module controls all the

previous modules – on the basis of given Desires and current Beliefs it sets new Intentions. The manager produces Concept-to-Speech (CTS) utterance descriptions and feeds them into the *Prompt Planer* module (its aim is to transform CTS descriptions into naturally sounding utterances). However, this module is currently not regularly designed and we substitute its function by merely passing the input to output.

III. GENERAL DIALOGUE MANAGEMENT ARCHITECTURE

This section discusses each of the modules in detail, providing a comprehensive description of the architecture.

A. Context Module

The Context module follows a two layered approach (Fig. 2), in which the upper layer serves for user's intentions detection, while the lower layer maintains dialogue objects and relations among them (i.e., “data” obtained during the past dialogue). An input semantics preprocessing results into a block structure representation (Fig. 4) that passes through both of the layers, leaving a specific *imprint* behind. In the figure, boxes are objects we will refer to as *concepts* throughout this paper, and arrows are *relations* among them. In both layers the imprints take the form of a *facts* $\text{FACT}(\text{concept}, \text{instance}_1, \text{instance}_2, \text{cs}, \text{saliency}, \text{belief})$,¹

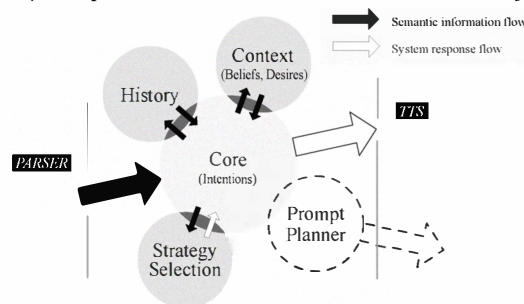


Figure 1. Manager modules topology and interaction

The facts play a role of providing a belief about concepts (the *concept* parameter in the fact definition) and *relations* (the *instance*_{1,2} parameters). The initial value for the *belief* parameter ($\in \langle 0;1 \rangle \equiv \langle \text{abs. disbelief; abs. confidence} \rangle$) is the confidence score (*cs*, $\text{cs} \in \langle 0.5;1.0 \rangle$) obtained from the ASR module. The *saliency* is an integer informing about how actual is this fact.

Let us start with the *lower layer* storing Beliefs about a dialogue. Considering a time-table domain data model (Fig. 3), the system may, for example, “belief” that a train is the

¹ We will refer to the conceptual components of the BDI architecture with corresponding terms with first letter capitalized.

desired transportation means and Prague is the particular city of arrival. The lower layer stores all data-like information (solid white or hatched concepts in Fig. 4). Knowledge the lower layer represents can be arranged into a nested structure as Fig. 5 shows.

The *upper layer* serves for user's intentions recognition. In a task-oriented dialogue management, intentions are expressed in some form of actions to perform. Instead of determining if a semantics particular concept relates to an action, we detect cases when it definitely does not. These are all concepts except those that contain data only. More formally, we can describe such concepts by introducing *cardinality of information* they carry. The following listing discusses all possibilities the cardinality may take on.

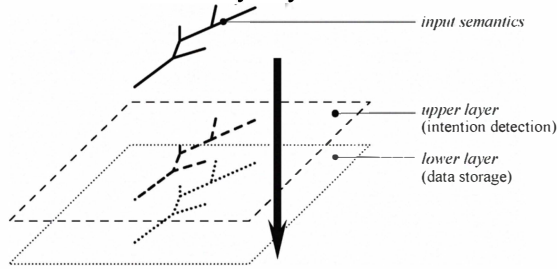


Figure 2. Two-layered Context module approach.

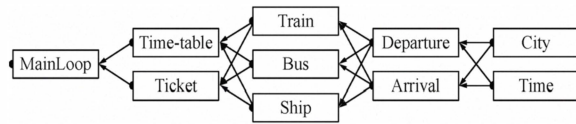


Figure 3. Time-table domain data model.

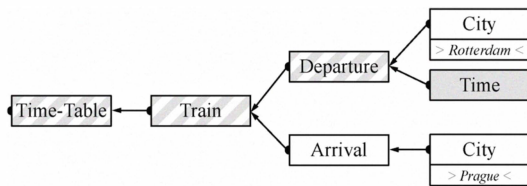


Figure 4. Block structure of preprocessed semantics of a complete request "When does a next train leave from Rotterdam to Prague?"; graphics represents concepts value cardinality: shadowed = infinite, hatched = non-zero, solid white = zero.

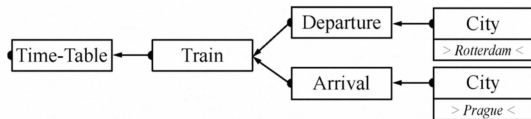


Figure 5. Lower layer contents only "data" mentioned in the dialogue.

TABLE I. Upper Layer SEMANTICS Imprinting Resolution.

		User's Sentence		
		Declarative	Imperative	Interrogative
Cardinality	Zero	–	–	–
	Non-Zero	–	Imprinted	Imprinted
	Infinite	–	Imprinted	Imprinted

- Let a leaf concept contain an *atomic information* (single time point, e.g. "2p.m."); atomic information has *zero cardinality*, i.e., zero uncertainty, and therefore, cannot carry any intention as there is nothing to discuss about it.
- Let a leaf concept contain a *non-atomic information* (time interval, e.g. "2p.m.-3p.m."); the information has *non-zero cardinality* as it has a certain level of uncertainty and as such may be a subject of a query.
- Let a leaf concept contain *no information* (undefined time value); we define such concept to have an infinite cardinality.

We recurrently can determine the cardinality of parent concepts in the block structure by considering the rule:

- Let the concept *C* contain at least one sub-concept with non-zero or infinite cardinality. Then *C* has a non-zero cardinality.

With knowing just the cardinalities, we are still unable to determine if a semantics segment should be imprinted into the upper layer. If we are on detecting actions to perform, we need to involve dialogue acts, more specifically, detect imperative or interrogative sentences. Our final approach to making imprints into the upper layer consists of combination of both – cardinalities and dialogue acts as Table I shows.

Once the semantics has been imprinted, the most recent intention is recognized. We use simple template matching approach, where each intention has its own pattern (Fig. 6). If it matches, the sum of concept saliences is computed. With more than one match, the pattern with highest total salience is considered as the actual user's intention.

To manage intentions, we have partially adopted Grosz and Sidner's work on intentions in a dialogue [3] – a *stack* of intentions managed by *dominance* among intentions. We currently omit *satisfaction-precedence* (as we rely on having exhaustive plan description) and user's *interruptions* (i.e., *temporal* changes of the dialogue course). The stack is managed by a single rule dictating that the user may introduce a new intention *I* without losing any of the current

$$(\forall I_s \in \text{Stack}; I_s \text{ DOM } I) \rightarrow \text{push}(I) \quad (2)$$

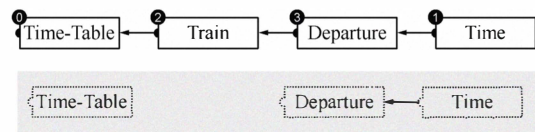


Figure 6. The upper layer content with saliences (numbers) and the DepartureTimeQuestion intention detection pattern (shadowed).

intentions already on the stack if each stacked intention *I_s* dominates *I*. If the rule does not apply, introducing *I* is considered as a permanent change of the dialogue course, causing intentions that do not dominate *I* to be popped out of the stack immediately.

B. Core Module

Once the intention stack is updated, the agent starts to process a *plan* on how to satisfy the top-positioned intention.

The plan resembles a tree [4] where nodes are holders of agent's activity (i.e., dictate utterances to say or back-end interaction to do). An example of a plan for a DepartureTimeQuery intention may be seen in Fig. 7. As first, the user is asked to say a transportation means to find time-table information for (the result is bound to the M variable for further processing). Next, the train parameters are constrained by posing some of the disambiguation questions. Finally, the database is queried and results presented to the user. After presenting them, the agent considers the user's intention to be *satisfied*. However, we postpone the popping of agent's corresponding desire out of the stack with respect to the user's next utterance – if s/he reopens the intention (by changing the underlying data in the lower layer), the desire remains on the stack, otherwise it is popped out [4].

The plan in Fig. 7 is a rather simple one, however, it is sufficient enough to demonstrate the first level of agent's adaptability. This kind of adaptability simply swaps plan tree branches according to the lower layer data salience. We are motivated by adopting the results of user's initiative – if s/he prefers to discuss certain part of a task prior to discussing the rest, the agent adopts the decision. As an example, consider user's elliptical utterance “by train to Rotterdam” is misrecognized by not understanding the transportation means. The city of arrival (Rotterdam) now has the highest salience in the lower layer. From the agent's point of view, the user wants to first discuss the city of arrival and then move to the rest. As a result it adjusts the plan tree structure by positioning the corresponding branch at the beginning of the plan (Fig. 8). However, this time the mandatory variable M is unbound due to the misrecognition. As a result, the agent starts to search the tree structure to find how to reach a value. After having found a solution, it puts the corresponding branch at the beginning of a plan again (Fig. 9).

TABLE II. EVENT TYPES (ORDERED DESCENDING BY IMPORTANCE); Applicability (IN PARENTHESES): L = LOWER LAYER, U = UPPER LAYER, P = PLAN, S = STACK

Event Type	Event Description
Desire satisfaction (P)	An event of the most importance processed as soon as the manager is able to satisfy a desire on top of the stack.
Generalization (U, L)	A given concept needs to be a part of a more general concept (e.g., City can be either of Departure or Arrival).
Disambiguation (P)	A concept queries a database and the number of results returned exceeds the number of results allowed.
Validation (U, L, S)	The ASR module recognized the given concept with a low confidence score and it needs to be further validated by the user.
Missing information (L)	A concept is missing an information (i.e., Time concept exists but carries no value).
Concept specification (P)	More detailed information is needed (the counter-event to the Concept-Generalization event).

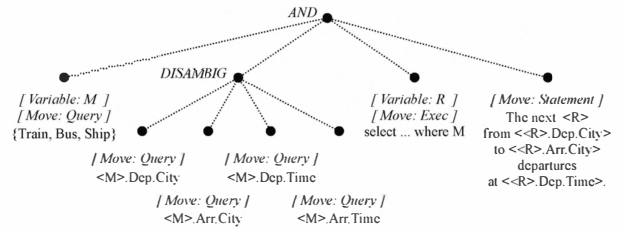


Figure 7. Initial plan for satisfying the DepartureTimeQuery intention.

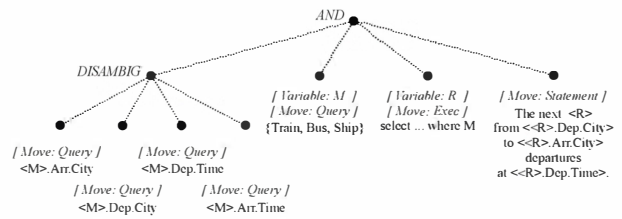


Figure 8. The DepartureTimeQuery plan with swapped branches.

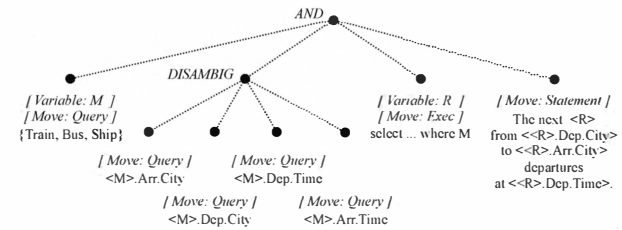


Figure 9. Feasible DepartureTimeQuery plan.

The characteristic feature of an agent-based dialogue management is the ability to optimize the dialogue flow in some way. So far, we mentioned several supporting means the dialogue agent consists of – intentions stack, data lower layer, action upper layer, and plans. We were searching how to bridge all of these to enable optimization of the agent's behaviour, and as a response, we created an events foundation. *Events* of different types define and represent elemental *operations* the agent is able to do (similar approach can be found in [5], however, it is applied on dialogue data only). As the events always relate to a specific *entity* (concept, relation, plan node, or intention), we preliminary can define them as

$$\text{EVENT}(\text{entity}, \text{operation}) . \quad (3)$$

For example, we might want to validate emerged intention on top of the stack by a *validation event*, or generalize a concept in the lower layer by a *generalization event* (e.g., the Departure concept is a “generalization” of the Time concept). Table II gives an overview of all events we distinguish (in order of importance).

Each event is considered as one option the manager is offered to take at a specific point in time. If the existence purpose of the event has been met (e.g. a concept the

validation event relates to has been validated by the user), we say that the event has been *satisfied*. As there may be more pending events in the context, the manager is given more choices regarding which one to start with at each of its turns. To find out the best order in that the events should be satisfied is the subject of the deliberation mechanism (see below).

We split events into *phases*. For majority of the events, the agent must 1) utter to the user, 2) wait for an answer, and finally 3) check the event satisfaction. All events follow this cascade model. Additionally, we track the state of recovery for each of the events. An event is said to be *recovered* if it reached the satisfaction phase but user's interaction has turned it back to the first phase (by making making changes in the context). Therefore, the final definition of an event is

$$\text{EVENT}(\text{entity}, \text{operation}, \text{phase}, \text{recovered}) . \quad (4)$$

As we noted above, events are means for the agent to make decisions, i.e., *plan* its behaviour. For example, given a Time concept with the Validation and Generalization events, the agent may either 1) first attempt to satisfy the Validation (“Did you say <time>?”) followed by the Generalization (“Does the time <time> refer to departure or arrival?”), or 2) it may attempt to satisfy straight the Generalization, as the Validation is involved (implicit validation).

In our implementation, we currently consider only one agent's behaviour optimization criterion (although there may be more): *the length of the dialogue in terms of dialogue turns*. From this point of view, the second mentioned choice (implicit validation) would be less *penalized*. Our future work also considers implementation of the optimization that regards the length in terms of elapsed real time. However, a missing clue still is how to combine both criteria into a single decision pattern.

The current context (upper and lower layers, intention stack and active plan) with all its events is considered as one of possible worlds from which we can move to another one by satisfying one or more of pending events. We search a space of possible worlds until it has been found the one in which the desire on top of the stack is satisfied. Fig. 10 shows the underlying algorithm.

To reach the optimization of agent's behaviour we use the *events penalization scheme* (Table III) to construct a plan. Recall that we consider a plan to be a sequence of events that is optimal with respect to the “minimal dialogue length” criterion above. Once the plan has been calculated, finding out what the system should say next is to follow the plan up to the nearest point where an interaction with the user is necessary. This is usually the point where the agent is missing some information, and hence, utters a *query*. Additionally, the agent may also utter a final answer that satisfies its desire, and hence, makes a *statement*. Agent's queries and statements are two of dialogue moves (Table IV) fed into the *dialogue moves stack*. The stack gathers moves performed by either of the participants. The purpose of the stack is to keep track of unfinished dialogue games [6] currently being played.

TABLE III. Event Penalization Criteria

Penalty Criterion and Explanation
<i>Event does not support a desire on top of the stack.</i> The more dominant desire it supports, the higher penalty, i.e., we want to support narrowed topics in the dialogue.
<i>Event cannot be reacted (is unavailable) due to a missing piece of information.</i> The system cannot query a time-table database if a transportation means is unknown; this results in an infinite penalty, i.e., stopping exploration in this direction.
<i>Concept salience.</i> The longer a given concept did not appear in the dialogue, the higher penalty – we want to stick to the current course of the dialogue.
<i>The number of pending events covered by the event.</i> For example, at least two events are covered in an implicit confirmation utterance: at least one validation event + some of information elicitation events; the lower the number, the higher the penalty.
<i>Event is not recovered.</i> Recovered events signal that the user has made corrections in the past dialogue – serving the corrections is given higher priority, i.e. not recovered events penalized.
<i>Event phase.</i> Events in their last processing phase are preferred – once user's answer is gained, we want to check if it satisfies system's question; events which do not meet this condition are penalized.
<i>Event type.</i> Events of most importance are preferred – see the previous section for ordered list of event types.

TABLE IV. Spoken Dialogue Moves

Dialogue Move	Example
Query	“When does a next train leave?” (user) “What time would you like to departure?” (system)
YN-Query	“Do you want to buy a ticket?” (system)
Validation	“Did you say...?” (system)
Statement	“Tomorrow morning” (user) “The next train departures at 7a.m.” (system)
Grounding	“I want” “Yes” “No” (user) “Ok” (system)

- 1 Duplicate the current world.
- 2 Repeat until you have to query the user, or you satisfy a desire on top of the stack.
 - 2.1 Choose one of unsatisfied events.
 - 2.2 Emulate a response to satisfy the selected event (e.g., validate a given concept). Here, we consider the domain of the event. E.g., for the Missing-information event we omit any emulation as the algorithm does not work with it at all, whereas for the Concept-specification and Generalization events we consider user's all immediate possible responses (we assume the number of them is always low in this case).
- 3 Recurrently repeat from step 1.

Figure 10. Searching algorithm for agent's deliberation.

TABLE V. Dialogue Strategies Criteria

System-initiative Strategy	Mixed-initiative Strategy	User-initiative Strategy
----------------------------	---------------------------	--------------------------

<ul style="list-style-type: none"> • dialogue quality estimation • correction of information elicited using higher initiative strategy • information to get has a large range • low recognition score • high recognition accuracy demand (unimplemented) 	<ul style="list-style-type: none"> • dialogue quality estimation • higher initiative strategy elicited information correction • acceptable recognition score 	<ul style="list-style-type: none"> • dialogue quality estimation • correction of information elicited using higher initiative strategy • high recognition score • user's intention is unknown
---	---	---

C. Strategy Selection Module

The purpose of the dialogue moves stack is to keep track of the very recent spoken interaction history. For example, despite we currently do not have the Prompt Planner module (Fig. 1) implemented, we yet are able to avoid agent's repetitive utterances by comparing agent's current move with moves already done. If there is a match, the agent may keep silent during its turn (handing the initiative back to the user).

The example situation depicted above is a very special one, more specifically, it requires high user speech recognition scores and smooth dialogue progress. However, as both factors may vary during the dialogue, this interaction style is not guaranteed to work each time, and the agent is forced to adapt what it says to the current situation observed. As a solution, we introduce four dialogue strategies: *system-initiative*, *mixed-initiative*, and *user-initiative* [7, 9]. All strategies are named with respect to the level of initiative the agent exhibits in each of them. Table V conceives a listing of dialogue features detected to determine the strategy the agent should use to generate its response (inspired by [7]). The decision process is similar to the Jaspis architecture [8]: with having three strategies to decide between, it is selected the one whose features reach the highest score.

D. History Module

The structure of the History module (Fig. 1) has not been changed since our last work [2], therefore, its description will be omitted in this paper.

IV. FUTURE WORK

In this paper, we have omitted to describe our approach to the History module, user's corrections, and structure of our semantics, however, all of these topics are covered in [2].

Our deliberation approach motivates us to an interesting direction of research – adaptability of intelligence based on an instant system performance. We perceive this as a good direction for the manager to be applicable in queuing service domains. We would like to simulate these experiments, instead of make real human tests. As a performance measurement we want to apply the PARADISE framework [10]. Last but not least, apart of the time-table domain, we also would like to apply the above presented dialogue manager in a personal assistance domain, considering e-mails and appointments management.

V. CONCLUSION

This paper presented our general dialogue management architecture, designed as a deliberative agent. We have presented a broad scale of algorithms that provide manager's particular capabilities some of which derive from well known approaches. One of the features the manager accommodates is the two-layered approach to detect user's intentions and maintain beliefs.

This paper presented foremost theoretical background of our research. The necessary changes to the architecture mentioned above are the subject of our current work.

ACKNOWLEDGMENT

This work was supported by grant no. 2C06009 Cot-Sewing.

REFERENCES

- [1] A. S. Rao, and M. P. Georgeff, "BDI agents: From theory to practice," Proc. 1st International Conference on Multi-Agent Systems (ICMAS), San Francisco, pp. 312-319, 1995.
- [2] T. Nestorovič, "A Frame-Based Dialogue Management Approach," Proc. 2nd International Conference on the Applications of Digital Information and Web Technologies (ICADIWT), London, pp. 338-343, 2009.
- [3] B. Grosz, and C. L. Sidner, "Attention, Intention and the Structure of Discourse," Computational Linguistics, vol XII(3), 1986, pp. 175-204.
- [4] C. Rich, C. L. Sidner, and N. Lesh, "COLLAGEN: Applying Collaborative Discourse Theory to Human-computer Interaction," AI Magazine, vol XXII, 2001, pp. 15-25.
- [5] S. McGlashan, "Towards Multimodal Dialogue Management," Proc. 11th Twente Workshop on Language Technology, Twente, pp. 1-10, 1996.
- [6] J. C. Kowtko, S. D. Isard, and G. M. Doherty, "Conversational Games Within Dialogue," Research Paper HCRC/RP-31, Human Communication Research Centre, University of Edinburgh, 1993.
- [7] S.-W. Chu, I. O'Neill, P. Hanna, M. McTear, "An Approach to Multi-strategy Dialogue Management", Proc. INTERSPEECH, pp. 865-868, 2005.
- [8] M. Turunen, J. Hakulinen, "Agent-based Adaptive Interaction and Dialogue Management Architecture for Speech Applications," Proc. of the 4th International Conference on Text, Speech and Dialogue, Springer, pp. 357-364, 2001.
- [9] T. Nestorovič, "Dialogue Systems," technical report Nr. DCSE/TR-2009-05, University of West Bohemia (UWB), Pilsen, 2009.
- [10] M. A. Walker, D. J. Litman, C. A. Kamm, A. Abella, "Evaluating Spoken Dialogue Agents with PARADISE: Two Case Studies," Computer Speech and Language, vol. XIV(4), 1998, pp. 317-348.