

CONALD: The Configurable Plan-Based Dialogue System



Bernd Ludwig, Peter Reiss, Günther Görz
Institute of Computer Science 8
University of Erlangen-Nuremberg
Haberstrasse 2, D-91058 Erlangen, Germany

1 What is CONALD?

CONALD is a spoken language dialogue system¹ which enables the interaction between users and a technical system in dynamic environments. The primary goal of its design is to achieve quick configurability for various applications. It combines deep syntactic and semantic analysis, discourse processing, and language generation and features a complex semantics-pragmatics interface in the sense of (Brietzmann and Görz, 1982). Semantics is defined in terms of an extended version of DRT. Discourse and application pragmatics are considered independent; user utterances affect application pragmatics when their speech acts are executed. They can be specified in a script language interpreted by the dialogue manager.

In dynamic environments, changes can be consequences of user requests as well as of external events in the application. The design of our system allows configuration for a wide range of applications (like household applications, the automotive environment, medical purposes, etc.). For demonstration purposes, we designed a complete environment that simulates logistic tasks. It is a multi-agent system (MAS) for controlling a model train installation and a robot that puts boxes with goods to a production line and fetches the final products again. In a control window trains, railtracks, and track switches are simulated (see Fig. 1). The user controls the scenario either with spoken input or via a keyboard interface. The user may give simple commands like “Set speed of train 2 to 11!” or “Switch point 6!”, but even highly complex commands like “Bring the

final goods to station A!” are understood. Feedback about the current state of the application is given in natural language output (Sect. 3.5), the length and frequency of system utterances depends on the user preferences and the situation — we do not try to overload the user with unnecessary information.

The parser transforms the user’s utterance into a semantic representation, from which the dialogue system derives a goal how to change the environment (Sect. 3.1). A plan to reach this goal is computed and executed by a group of agents (sect. 3.2 and 3.3). Under the assumption of a closed world, the overall system features hierarchical planning, plan execution, and plan observation by several agents with different responsibilities and capabilities. In this way, system knowledge is distributed and organized hierarchically corresponding to the tasks each agent has to execute.

2 Application Control

Agents are organized in a hierarchy of layers. On top there is the Central Control Agent (CCA) which plays the role of an interface between the application and the dialogue system. The main task of the CCA is to satisfy the user’s goals by computing and executing high-level plans. In our demonstration application the CCA maintains a model about all Route Agents (RAs) which is represented as an undirected connectivity graph. Each node is a RA and the edges symbolize connections between two RAs.

While the CCA is a generic interface between the dialogue system and an application modelled in terms of plan operators, on the competence levels below the CCA there are application specific agents responsible for the execution of actions that

¹Acknowledgment: Our work is supported by the Bavarian Research Association FORSIP

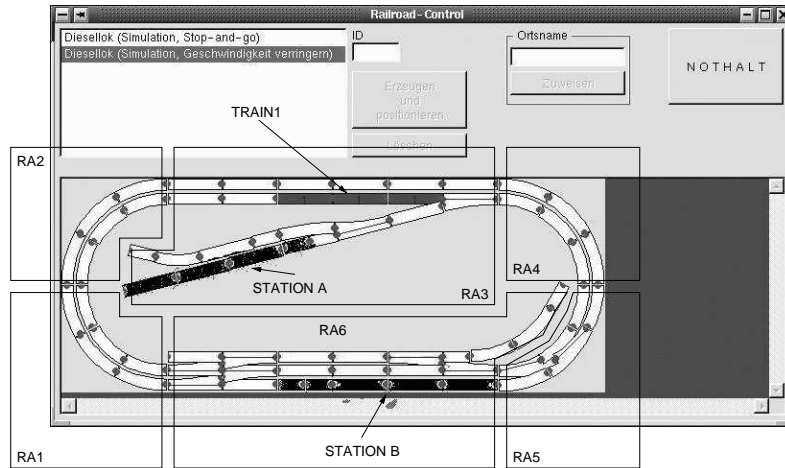


Figure 1: Display of the Simulation Agent.

are primitive from the *CCA*'s point of view. The *RAs* control the trains and switches in their section of the railroad: they compute the route that the train takes when passing the area they control and they change all involved switches.

Train Agents (*TR*) represent the trains in the installation. Each *TR* executes and observes the plan computed by its supervising *RA*. If the state of the model train installation differs from the expected one the *TR* informs the responsible *RA* for exception handling.

The Simulation Agent (*SA*) simulates the operations and controls the interface to the model trains. Finally, the Robot Agent (*RO*) controls the robot which in turn operates the production line.

3 The Dialogue System

Negotiation between user and *CCA* is handled by a dialogue manager. For user utterances, input from a speech recognizer is processed by a parser resulting in a representation for the meaning of the utterance.

3.1 Natural Language Analysis

The parser works in two phases: first, it segments the output from the speech recognizer into chunks (Abney, 1991) according to a chunk unification grammar. The chunks have to be translated into constraints for a (partial) description of a system state. For that purpose, an approach that is motivated by dependency theory is applied: valencies for the syntactic head of each chunk are analyzed if they can

c	box1	stationA] object transportation task object at destination type of destination involved object
final-goods(c)			
transport-with(c, box1)			
at(box1, stationA)			
station(stationA)			
box(box1)			

Figure 2: Domain specific meaning of a user utterance.

serve as the dependent for some other chunk (its regent). Dependent and regent have to meet three classes of criteria in parallel: syntactic constraints, semantic constraints (is the semantic part of the valency satisfied?), and pragmatic constraints (can a constraint in the application domain be derived from the triple regent, thematic role, and dependent?). For the example utterance “*Bring the box with the final goods to station A!*” the parser computes the following analysis:

Chunk	Semantics (informal)
Bring	transportation task
the box with the final goods	object for task
to station A	destination for task

From that analysis, the DRS in Fig. 2 is derived as a partial description of an application state.

In parallel to the analysis of the content, a speech act is assigned to the utterance in order to determine its discourse pragmatic function. In the example, from the syntactic observation that the utterance is

an imperative, a request to the application is derived. For planning a reaction, the dialogue system needs to find out first whether there is a plan to satisfy the request.

3.2 Plan Generation

We have configured the FF Planner (Hoffmann and Nebel, 2001) with plan operators for the model train domain. The initial state for the planning process is acquired by querying the application agents about their current state. If a plan for the goal computed by the parser can be found, it is forwarded to the CCA (Fig. 3 shows an example plan for goods delivery).

3.3 Hierarchical Execution

In order to satisfy the user's intentions, the CCA starts the execution of the computed plan. It delegates the actions to the responsible agents down in the hierarchy: For each action to be executed, the RA constructs a plan for the CCA action and thereby refines the CCA plan step by dividing it into several subactions that it can execute for the section it controls. In a recursive manner, the RA again delegates the execution of the actions it has planned to agents on a lower level in the hierarchy. In this case, the TR are addressed and in turn they inform the SA to update the state of the simulation.

3.4 When Is a User Request Satisfied?

To keep track of the current state of a dialogue, an extended version of DRT is implemented to provide an explicit formal representation of the currently pending requests. The hierarchical execution of plans requires involved agents to provide feedback about the execution of an action to the agent

```

0: switch-on cml
   put-to-prodline box1 cml stack1
   compute-route stationB siding engine1
1: go engine1 stationB siding
   fill-box cml box1
2: connect waggon1 engine1 siding
   compute-route siding stationC engine1
3: go engine1 siding stationC
4: put-on-waggon engine1 waggon1 box1
   CML stationC
   compute-route stationC stationA
   engine1
5: go engine1 stationC stationA

```

Figure 3: A plan for a complex user command.

```

proc ACCEPT(task contrib);
drs uttcont = content(contrib);
float rel=0.0, vol=0.0, speed, sda_dist;
begin
  forall referent t in uttcont do begin
    vol := vol+consumption(uttcont,t);
    rel := rel+priority(uttcont,t)
  end;
  sda_dist := time() - last_sda_time;
  last_sda_time := curr_time;
  speed := vol/sda_dist;
  old_speed := speed
  if speed < 0.25 then utter contrib;
  else if rel > 4.0 then utter contrib;
end;

```

Figure 4: Dialogue operation ACCEPT.

that initiated the request. The feedback relevant to the success of the user request is propagated to the dialogue system so that it can update the processing state of each user request. From that information it can conclude if a request is satisfied or still being processed.

3.5 Situation-Adaptive Interaction

However, reacting to utterances requires more than providing correct answers: We see our model train scenario as a good environment to test strategies for natural language system output, having in mind “real world” scenarios like assistance systems in automobiles. If a critical situation is detected, the driver has to be informed immediately. In non-critical situations there is enough time to give the driver less important information such as “*Your friv- ing style will cause a lot of fuel consumption*”. Nevertheless, it should be up to the user to set the preferences on what type and amount of information he would like to be provided with.

We configured our dialogue manager to take into account the fact that the cognitive capacity of the user is limited, in particular if he has to pay attention to many different tasks, as well as the relevance of the information. As can be seen in the implementation of the operator ACCEPT in Fig. 4, we first compute the total costs of the system utterance by summing up the volumes for each referent. In the same way we compute the relevance of the contribution. The costs of the utterance depend on various factors (e.g. the kind of presentation — letting the system speak a whole sentence takes much longer

than ringing a bell as a warning signal). Hence, the function `consumption` may be very complex. To avoid overload, the time that passed since the last system utterance is also taken into account. After all, if the importance of the message exceeds a defined threshold (or if the cost is low enough), the contribution is uttered. In our current implementation, the values for cost and relevance are retrieved via table-lookup in a file (Fig. 5 shows an example configuration). The user can tailor the system behavior by setting appropriate values in the column `priority`. The column `consumption` and matches the current output modality — spoken language.

4 Related Work and Conclusions

The presented dialogue system differs from others in three key issues: first, planning and processing of discourse and application are kept separate. This allows for a plug-and-play architecture: The functionality of the application can be modified without any consequences for the discourse model and vice versa. In (Allen et al., 2001) an architecture is described that integrates planning and discourse more closely and therefore seems to be less flexible. In systems as (Lamel et al., 1998), a dialogue strategy is hardwired in the dialogue manager’s kernel. In our system, however, a dialogue designer may modify an existing strategy or even add new strategies by adapting a control program (in the proprietary language of our dialogue system, Fig. 4 shows an example) that is part of the configuration. In this way, non linguistic facts of interaction (e.g. user emotions) may influence the system’s dialogue behaviour. These aspects are discussed in isolation in the literature; our system provides a platform for integrating them under real-time conditions. Second, the system implements a combination of deep processing for parsing and generation. Third, dialogue coherence is determined by analyzing the content of contributions to a dialogue with the help of a par-

tial logic (Abdallah, 1995): for each new utterance, the dialogue manager evaluates how it contributes to the completion of one of the currently active plans. Hence, the dialogue model extends the information state approach e.g. of (Larsson, 2002). Coherence is not limited by the expressiveness of finite automata; so, the dialogue model is more expressive than that of FSM-based approaches. In contrast to other configurable systems such as GALAXY (Seneff et al., 1998), in CONALD coherence is defined as a contribution to a plan.

References

Nait Abdallah. 1995. *The Logic of Partial Information*. Springer.

Steven Abney. 1991. Parsing by chunks. In R. Berwick, S. Abney, and C. Tenny, editors, *Principle-based Parsing*. Kluwer, Dordrecht.

James F. Allen, George Ferguson, and Amanda Stent. 2001. An architecture for more realistic conversational systems. In *Proceedings of Intelligent User Interfaces 2001 (IUI-01)*, pages 1–8, Santa Fe, NM.

Astrid Brietzmann and Günther Görz. 1982. Pragmatics in speech understanding – revisited. In J. Horecky, editor, *Proceedings of the Ninth International Conference on Computational Linguistics*, pages 49–54, Amsterdam. North-Holland.

Jörg Hoffmann and Bernhard Nebel. 2001. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302.

L. Lamel, S. Rosset, J. Gauvin, S. Bennacef, and G. Prouts. 1998. The limsi arise system. In *Proceedings of IEEE 4th Workshop on Interactive Voice Technology for Telecommunications Applications*, pages 209–214, Torino, Italy.

Staffan Larsson. 2002. *Issue-based Dialogue Management*. Ph.D. thesis, Department of Linguistics, Göteborg University, Göteborg, Sweden.

Stephanie Seneff, Ed Hurley, Raymond Lau, Christine Pao, Philipp Schmid, and Victor Zue. 1998. Galaxy-ii: A reference architecture for conversational system development. In *Proceedings of ICSLP 98*, Sydney, Australia.

CONCEPT	PRIORITY	CONSUMPTION
trainID	2.5	0.5
switchID	2.5	0.5
stringID	0.5	0.5
stringUSS	0.45	1.34

Figure 5: Discourse relevance of concepts.