
An Agent-based Platform for Dialogue Management

MARK BUCKLEY

Dept. of Computer Science, Saarland University

markb@ags.uni-sb.de

ABSTRACT. In this paper we propose a platform for developing dialogue management applications supporting the information state update approach which uses agent-based techniques and a hierarchical blackboard architecture to provide concurrency, runtime flexibility and the inclusion of heuristic control.

1 Introduction

Flexible natural language tutorial dialogue is an application area which puts heavy demands on a dialogue manager, for instance in orchestrating system execution, facilitating communication between system modules, and supporting multimodality. In this paper we propose a platform for developing dialogue management applications based on agent-based techniques borrowed from Ω -Ants (Benzmüller and Sorge 2000), a hierarchical blackboard architecture for guiding interactive proof planning. The proposed dialogue manager will provide concurrency, runtime flexibility, and allow the application of heuristics. It is motivated by our work on the DIALOG¹ (Benzmüller et al. 2003; Pinkal et al. 2001; Pinkal et al. 2004; Pinkal et al. 2004) project in the Sonderforschungsbereich (SFB 378) *Resource-adaptive Cognitive Processes* at Saarland University. The DIALOG project is a cooperative project whose aim is to investigate flexible natural language dialogue in mathematics, with the final goal of natural tutorial dialogue between a student and a mathematical assistance system.

This paper is divided into five sections. In section 2 we describe the DIALOG project and outline its system architecture. Section 3 is an overview of the Ω -Ants system, detailing its use of agent-based technology for guiding interactive proof planning. In section 4, which contains the core ideas of the paper, we present the dialogue manager itself, followed by some conclusions and related work in section 5.

¹<http://www.ags.uni-sb.de/~chris/dialog/>, <http://www.coli.uni-sb.de/sfb/>

2 The DIALOG Project

In this section we describe the research directions of the DIALOG project, and outline the system architecture.

In the initial phase of the DIALOG project a Wizard of Oz experiment (Fiedler and Gabsdil 2002) was conducted to collect a corpus (Wolska et al. 2004) of student/tutor dialogues which has become the basis for guiding research in this area. The phenomena found in the corpus can be divided into 3 levels. At the linguistic level there was evidence of ambiguity, varying degrees of formal content, and tight interleaving of natural and mathematical language. An example is the utterance (1), which shows the mixture of natural and mathematical language. It also shows underspecification, because the user does not make clear how this fact was derived.

(1) USER: B does not contain any $x \in A$.

At the tutorial level (Fiedler and Tsovaltzi 2003) we found mixed effectiveness of didactic tutoring, in which answers and explanations are constantly provided by the tutor, and socratic tutoring, where hints are used to achieve self-explanation. On the domain reasoning level we found evidence of much underspecification of mathematical concepts and statements. This is shown in (2), where it is not clear if the user is stating an axiom or performing a proof step applying that axiom.

(2) USER: According to DeMorgan-1, $K(A \cup B)$ is $K(A) \cap K(B)$.

We also found that a more human-oriented view of the notion of proof is needed in which assertion level reasoning (Huang 1994) plays an essential role. These phenomena show that a close interplay of natural language understanding, mathematical reasoning, tutorial reasoning, and dialogue modelling is required in order to accurately model mathematical tutorial dialogues. To facilitate an account of these phenomena we decided to use the information state update (ISU) approach to dialogue management, recognising that it is the dialogue manager that connects and orchestrates other system modules. This is presented in section 2.2.

2.1 DIALOG System Architecture

In this section we briefly introduce the various subsystems of the DIALOG tutor system, in order to motivate the required functionality of the dialogue manager. The initial coarse grained architecture of the system is shown in Figure 4.1.

The natural language (NL) analyser receives the student's utterance and determines its linguistic meaning and proof content. Input is syntactically parsed using the openCCG² parser, and its linguistic meaning is represented using *Hybrid Logic Dependence Semantics* (HLDS) (Baldrige and Kruijff 2002). It outputs a structure containing the linguistic meaning (LM) of the utterance represented in HLDS and the underspecified proof step

²<http://openccg.sourceforge.net/>

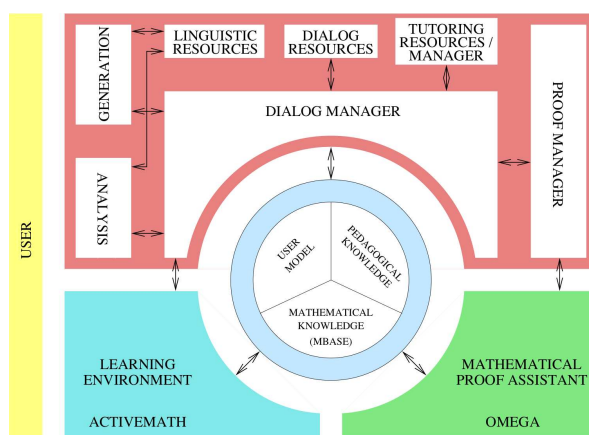


Figure 4.1: The architecture of the DIALOG system.

contained in the utterance, in an ad-hoc LISP-like representation (LU) (Autexier et al. 2003).

The proof manager is the mediator between the dialogue manager and the mathematical proof assistance system Ω MEGA-CORE (Siekmann and Benzmler 2004; Siekmann et al. 2003). The proof manager replays and stores the status of the partial proof which has been built by the student so far, and based on this partial proof, it analyses the soundness, relevance and granularity of a next proof step. The proof manager tries to resolve ambiguity and underspecification in the representation of the proof step. It receives the underspecified proof step which was extracted from the user's utterance by the input analyser, reconstructs the proof step that the student has made, and then outputs its fully specified representation.

The job of the tutoring manager is to use pedagogical knowledge to decide on how to elicit information or give hints to the user (Fiedler and Tsovaltzi 2003). These take the form of, for instance, a mathematical concept. It can also simply accept or reject the proof step in the case that it is correct or incorrect, respectively, depending on the tutorial mode.

The natural language generation system *P.rer* (Fiedler 2001), which presents complete mathematical proofs in natural language, has been adapted for the DIALOG project. The NL generator receives as input a dialogue move, which is a specification of the possibly many functions of an utterance in a dialogue, and generates the corresponding utterance.

2.2 Dialogue Management using the ISU approach

As shown in Figure 4.1, the dialogue manager has a central role to play in the DIALOG system. The two functions that the dialogue manager has to fulfil in the scenario of the DIALOG project are maintaining a dialogue model and facilitating communication between system modules.

In terms of maintaining a dialogue state, there are a number of approaches which can be considered for DIALOG. Finite-state methods, containing all possible dialogues, such as

the CSLU toolkit (McTear 1998), are suitable when the number of possible dialogues is relatively small. Another approach is form-filling, such as in the AUTOTUTOR system (Graesser et al. 1999), which is more adaptable than finite-state systems, and typically used in information elicitation scenarios. However, DIALOG requires a degree of flexibility which is not possible with either of these techniques alone.

We decided on employing the *information state update* (ISU) approach, as used in the SIRIDUS³ and TRINDI⁴ projects, and implemented in TrindiKit (Traum et al. 1999). This is heavily based on the theory of grounding (Traum 1999), in which the achievement of a common ground is done by grounding acts, each of which is identified with an utterance unit. In the ISU design the dialogue manager maintains a description of the state of the discourse and its participants, known as the information state (IS), which then forms the basis for the choice of action of the dialogue manager.

The dialogue manager also forms the link between the subsystems of a dialogue application. As shown in Figure 4.1, modules in our system are not able to communicate directly with each other, rather their communication is handled by the dialogue manager, and the dialogue manager is able to control module execution.

The components of such a manager are an information state (IS) and its concrete representation, a set of dialogue moves for the domain at hand, a set of update rules defined on the IS, and an update strategy for choosing update rules (Traum and Larsson 2003). The IS is a storage area for dialogue-level knowledge, and is realised as a set of abstractly typed slots whose values can be read and altered by the update rules. Update rules can be seen as transitions between information states. An update rule consists of preconditions, which are constraints on values in the IS, and effects, which are updates to the IS.

2.3 The DIALOG Demonstrator

As part of a demonstrator system (Buckley and Benz Müller 2003) for DIALOG we implemented a tailor-made dialogue manager. The dialogue manager was built using Rubin (Fliedner and Bobbert 2003), a platform for developing ISU based dialogue applications from CLT⁵. Rubin provides the functionality to maintain an information state, interface with external modules, and to update the IS based on input rules which are triggered by input from modules.

During the development of the demonstrator, we determined a number of desiderata for a dialogue manager in DIALOG:

Flexible flow of control System action in a dialogue manager is triggered by input from a module which causes an input rule to fire. This means that in each input rule, the dialogue manager has to pass control to some module which it knows will return some data. However we require a more flexible way to control system execution.

³<http://www.ling.gu.se/projekt/siridus/>

⁴<http://www.ling.gu.se/research/projects/trindi/>

⁵<http://www.clt-st.de>

Direct IS access Modules should be able to directly read information from the IS rather than waiting to be sent information by the dialogue manager. This would facilitate concurrent module execution.

Runtime flexibility If we could alter the dialogue manager at runtime, it would be possible to for instance to rearrange rule order or change constraints on rules.

Meta-level control In Rubin there is no meta-level which controls execution of input rules. A meta-level would allow us to include heuristics for controlling overall system execution, to compare possible IS updates before they are executed, and to decouple IS updates from information flow.

In section 4 we show how the proposed dialogue manager can support these functionalities.

3 Ω -Ants: An Agent-based Resource-adaptive System

In this section we introduce Ω -Ants and its agent-based design. Ω -Ants is a suggestion mechanism for the automated theorem prover Ω MEGA (Siekmann and Benzmüller 2004), which employs artificial intelligence techniques to the field of mathematical theorem proving and education. The Ω MEGA system is based on *knowledge-based proof planning* (Melis and Siekmann 1999), which makes extensive use of mathematical knowledge. For a particular domain of mathematics, a theory can be defined which contains the knowledge, encoded in so-called commands, necessary to find a proof of a given theorem in the domain. Ω MEGA uses AI planning techniques to build a proof using methods as planning operators. External reasoning systems which usually encapsulate knowledge in a special domain of mathematics, e.g. deduction systems or computer algebra systems, form another important source of mathematical knowledge.

Ω -Ants is a suggestion mechanism whose function is to support a human user in constructing a proof in Ω MEGA. Ω -Ants runs a number of background processes in addition to the Ω MEGA system, which are used to continually compute suggestions for a proof step that can be applied to the current proof object. The suggestions are then heuristically sorted and presented to the user in a graphical user interface, and the user chooses one of them to apply to the proof object.

Ω -Ants is an agent-based system. That is, each command in the current theory is represented by a society of software agents called argument agents, one for each argument pattern of the command. These agents are separate independent processes which search the proof data structure (\mathcal{PDS}) for command instantiations, which are then written as suggestions to a command blackboard, as shown in Figure 4.2. Finally the suggestions are written by command agents to the suggestion blackboard where they are heuristically sorted.

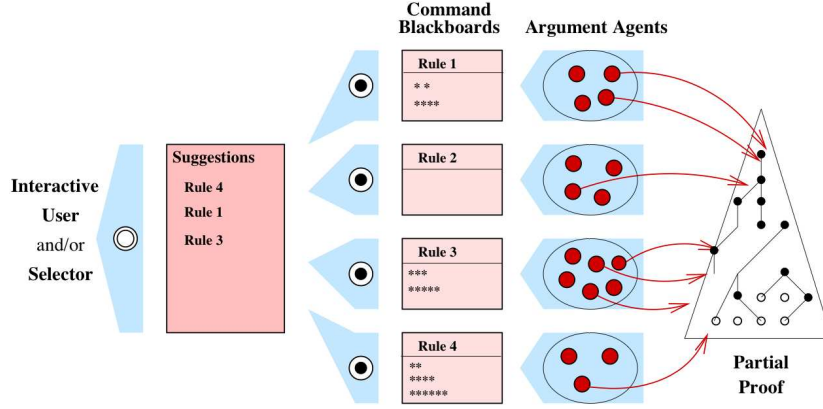


Figure 4.2: The Ω -Ants architecture.

3.1 Ω -Ants Agent Definition

A command in Ω MEGA can be seen as an inference rule consisting of premises and a conclusion which together make up the parameters of the command. An agent which represents this command is defined by subsets of parameters: the goal set, which are the parameters for which the agent computes an instantiation, and the dependency set, which are the parameters which must be present in the \mathcal{PDS} . When each of these sets are satisfied by some parameters from the command blackboard, the agent can compute a new instantiation and write this back to the command blackboard. An example is the Ω MEGA command for conjunction introduction in the forward direction, and the associated Ω -Ants agent:

$$\frac{LConj \ RConj}{Conj} \text{AndI} \Rightarrow \mathfrak{G}^{\{Conj\}}_{\{LConj, RConj\}}$$

This agent can execute when it finds an instantiation for the two conjuncts $LConj$ and $RConj$, and computes the formula $Conj$.

3.2 Benefits to Theorem Proving

Concurrency Ω -Ants agents run as independent processes, which means that many agents can concurrently compute suggestions, thus maximising the use of system resources. This also leads to an “anytime” character; the quality of suggestions improves continually over time. For instance, the number of instantiated arguments of the suggested command increases.

A major benefit of concurrency becomes clear when one considers external systems such as other theorem provers or computer algebra systems. These are used by Ω MEGA as slave systems, and there are corresponding commands in Ω MEGA which make calls to these systems. The Ω -Ants agents which represent these commands can make calls to the

external systems in order to compute suggestions, and thus many external systems can be used simultaneously.

Resource-adaptiveness In Ω -Ants the execution of each agent can be monitored and controlled separately by a resource agent which can allocate more or less system resources to an agent, depending on how effective the agent is in the suggestion process.

Runtime Flexibility Ω -Ants includes the functionality to define and add agents at runtime. For example, if a new external prover becomes available, then the user can simply define a new agent to interface with this system, insert this agent dynamically into the system, and receive suggestions from this prover right away.

4 Agent-based Dialogue Management

In this section we describe in more detail the software agents of Ω -Ants, and present the proposed design of the agent-based dialogue manager.

The solution we propose is to apply the agent-based technology used in the Ω -Ants system to build a platform for dialogue management applications based on the ISU approach, thus combining the functionality of an ISU based manager with the benefits of agent-based design outlined above. Each update rule will be represented by a software agent in the spirit of Ω -Ants. Update rules intermittently check if their preconditions hold, and if so, write their effects to an update blackboard. This in turn is monitored by an update agent, which carries out one or more of the proposed updates.

4.1 An Agent-based Dialogue Manager

We now show how a parallel notion of agents to the one used in Ω -Ants can be used as a solution to dialogue management. We first indicate some similarities between the Ω -Ants system and the proposed dialogue manager. Both have at their core a central data structure; for Ω -Ants the \mathcal{PDS} , for the dialogue manager the IS. In the same way that Ω -Ants uses agents to compute command instantiations, the dialogue manager uses agents to represent the update rules of the dialogue manager. These agents will search the IS and compute IS updates. An important similarity is the integration of external systems. Ω -Ants agents allow the natural and efficient integration of external reasoning systems into the proof planning process. The dialogue manager must also integrate a number of external modules, and the same techniques can be applied in dialogue management.

In order to actually define the behaviour of the dialogue manager, the developer of a dialogue application defines the structure of the IS and a set of update rules on it.

The *information state* is defined as a set of typed slots, each with an optional initial value and an optional test for legal values of the slot. Slots and their types are completely freely defined. The IS provides the functionality for slots to be read and updated.

An example of a well-defined IS slot is for the tutorial mode, which is a flag indicating whether we are currently in the didactic, socratic, or minimal feedback mode:

```
(tutorialmode :doc "the current tutorial mode"
```

```

:init "min"
:test #'(lambda (val list) (find val list :test #'equal))
:args (("min" "soc" "did"))

```

The test on this slot expands to a check whether the proposed value is in the list ("min" "soc" "did"), thus ensuring that the slot can only be given correct values. The slot is given the initial value of "min", since this is the most basic tutorial strategy and was used at the start of the demonstration.

Update rules compute updates of slots in the IS, and consist of preconditions, side conditions and effects. Preconditions are constraints on the values of slots in the IS which must hold in order for the rule to fire. A side condition introduces a variable which is visible in the effects and the remaining side conditions, and which becomes bound to the result of evaluating a given arbitrary expression. An effect is an assignment of a value to an IS slot, subject to the type checking in the IS.

An update rule is defined as a tuple (U, P, S, E) , referring to the name of the rule, and its preconditions, side conditions, and effects, respectively. P and E must contain at least one element each. The general form of an update rule is:

$$\frac{p_1 : test_1, \dots, p_n : test_n}{(e_1, expr_1), \dots, (e_m, expr_m)} U ((s_1, expr_1), \dots, (s_l, expr_l))$$

with preconditions $P := p_1, \dots, p_n$, effects $E := e_1, \dots, e_m$, (where all p_i and e_i are well-defined slots in the IS) and side conditions $S := s_1, \dots, s_l$.

An update rule can fire when its preconditions are satisfied by the current IS, that is, when for each precondition $p_i : test_i \in P$ the function $test_i$ evaluates to true for the value of the slot p_i . When the rule fires the set of sideconditions S is evaluated. That is, for each side condition $(s_i, expr_i) \in S$, the variable s_i is bound to the result of evaluating $expr_i$. After the sideconditions have been evaluated, an *IS update* is computed. This is a set E of effects (e_i, v_i) , where e_i is the IS slot to be updated and v_i is the new value which has been computed for the slot, and all e_i are unique. The IS update is then written to the update blackboard, and the rule (more specifically, the agent representing the rule) goes back to monitoring the IS and checking its preconditions.

We now give an example of an update rule, showing how it can be used to integrate an external module, namely a natural language analyser:

```

(ur~define-update-rule :name "NL Analyser"
  :preconds ((utterance :test #'stringp))
  :sideconds ((result (call-to-NL-module utterance))
              )
  :effects ((lm result))
  )

```

The precondition states that the rule can fire when the IS slot `utterance` contains a string, and the function `call-to-NL-module` is the interface to the external module.

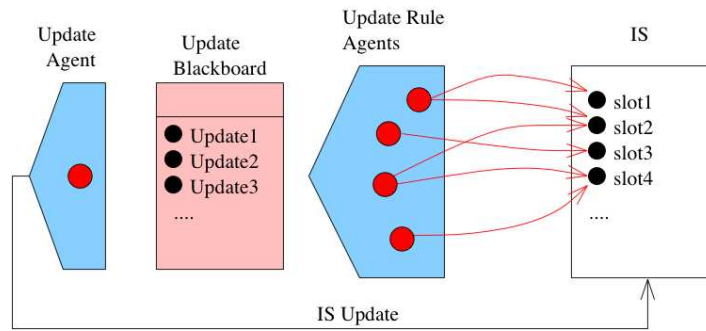


Figure 4.3: The architecture of the dialogue manager

The IS update that the rule computes contains an update to a single IS slot, $1m$, namely assigning it the value which was returned by the NL analyser.

The *update agent* is the equivalent of the command agent in Ω -Ants which surveys a particular command blackboard. The update agent surveys the update blackboard, comparing and sorting IS updates. After a certain length of time, it chooses an update based on heuristics, executes its effects (thus updating the IS) and resets the system. The update agent contains dialogue-level knowledge and heuristics to enable it to choose the “best” update, or union of disjunct updates. A system reset means that the update blackboard is cleared and all agents are reset. Any partially complete computations are aborted, since the information from the IS that they rely on may now be out of date. One of the benefits of the blackboard design is that the update agent, which is where heuristics are built in, does not have to reason about static rule definitions, but rather can reason based on fully instantiated concrete IS updates. This way the update agent knows the full outcome of any proposed update.

Figure 4.3 shows the architecture of the dialogue manager. On the right is the IS, the central data structure of the system. Each of the software agents which represent the update rules of the dialogue manager has in its preconditions a subset of the set of slots in the IS. When the agent sees that its preconditions hold, the rule can fire. The agent then computes the IS update that is defined by the rule, and writes this to the update blackboard. This happens in a concurrent fashion, so that many agents may be simultaneously computing results, or may have made calls to external systems. On the left in the diagram is the update agent, which surveys the update blackboard, and after a timeout or some stimulus makes the heuristically preferred IS update, and resets the system.

4.2 Benefits to the DIALOG Project

An agent-based dialogue manager accounts for the desiderata outlined in section 2.3 and brings a number of benefits to DIALOG:

Single specification of system behaviour Update rules provide an elegant, unified and

intuitive way to specify system behaviour, both with regard to external systems, internal updates and the flow of control.

Application of heuristics With the proposed dialogue manager, the DIALOG system will have a meta-level where dialogue-level heuristics can be implemented, which can have an effect on the whole system.

Inter-module communication Since agents have direct IS access, they can communicate by writing information to shared slots.

Account of barge-in Many dialogue systems use an architecture in which processing is based on an iteration of NL understanding \rightarrow dialogue-level reasoning \rightarrow NL generation, in which barge-in is difficult to account for. With an agent-based solution, the execution of the system would not have to be interrupted in order to receive a new utterance.

Adaptation to developing modules As the DIALOG system develops, the dialogue manager will be able to adapt to improvements and alterations to other system modules. Because module interfaces are defined in the update rules, adding or altering an interface simply involves rewriting its update rules.

5 Conclusions and Related Work

We argue that the unique research focus of the DIALOG project puts new demands on a dialogue manager, and propose a solution that uses the agent-based resource-adaptive techniques of the Ω -Ants system to build a dialogue manager that fulfils all the requirements of DIALOG, as well as bringing the benefits of agent-based technology to the project, such as concurrency, flexibility, and the easy integration of external systems. It also allows a crucial level of heuristic control over the complex interplay of modules.

Another solution to dialogue management based on software agents is the Dipper system (Bos et al. 2003), currently used in the TALK project⁶. The Dipper architecture is a collection of software agents implemented in the Open Agent Architecture (OAA) (Martin et al. 1999) for prototyping dialogue applications using the information state update approach. The OAA is used to interface with agents which represent external modules, and so asynchronous dialogue systems can be built. In contrast to our solution the execution control in Dipper has no meta-level in which to reason about choice of update rule.

It is not yet clear what the heuristics to guide the dialogue manager are, but the PARADISE algorithm (Walker et al. 1997) may help recognise suitable dialogue strategies.

6 Acknowledgements

This work was supported by the German Academic Exchange Service (DAAD), grant number A/03/15283, and by the Collaborative Research Centre 378 for Resource-adaptive Cognitive Processes.

⁶<http://www.talk-project.org/>

Bibliography

- Autexier, S., C. Benzmüller, A. Fiedler, H. Horacek, and B. Q. Vo (2003). Assertion-level proof representation with under-specification. *Electronic in Theoretical Computer Science* 93, 5–23.
- Baldrige, J. and G.-J. M. Kruijff (2002). Coupling CCG with Hybrid Logic Dependency Semantics. In *Proceedings of the 40th Annual Meeting of the Association of Computational Linguistics (ACL'02), June 7-12*, University of Pennsylvania, Philadelphia.
- Benzmüller, C., A. Fiedler, M. Gabsdil, H. Horacek, I. Kruijff-Korbayova, M. Pinkal, J. Siekmann, D. Tsovaltzi, B. Q. Vo, and M. Wolska (2003). Tutorial dialogs on mathematical proofs. In *Proceedings of IJCAI-03 Workshop on Knowledge Representation and Automated Reasoning for E-Learning Systems*, Acapulco, Mexico, pp. 12–22.
- Benzmüller, C. and V. Sorge (2000). OANTS – an open approach at combining interactive and automated theorem proving. In M. Kerber and M. Kohlhase (Eds.), *Symbolic Computation and Automated Reasoning*, pp. 81–97. A.K.Peters.
- Bos, J., E. Klein, O. Lemon, and T. Oka (2003). Dipper: Description and formalisation of an information-state update dialogue system architecture. In *Proceedings of the 4th SIGdial Workshop on Discourse and Dialogue*, Sapporo, Japan.
- Buckley, M. and C. Benzmüller (2003). A Dialogue Manager for the DIALOG Demonstrator. SEKI Report SR-04-06, Fachrichtung Informatik, Universität des Saarlandes, Saarbrücken, Germany.
- Fiedler, A. (2001). Dialog-driven adaptation of explanations of proofs. In B. Nebel (Ed.), *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, Seattle, WA, pp. 1295–1300. Morgan Kaufmann.
- Fiedler, A. and M. Gabsdil (2002). Supporting progressive refinement of Wizard-of-Oz experiments. In C. P. Rosé and V. Alevén (Eds.), *Proceedings of the ITS 2002 — Workshop on Empirical Methods for Tutorial Dialogue Systems*, San Sebastián, Spain, pp. 62–69.
- Fiedler, A. and D. Tsovaltzi (2003). Automating hinting in mathematical tutorial dialogue. In *Proceedings of the EACL-03 Workshop on Dialogue Systems: Interaction, Adaptation and Styles of Management*, Budapest, pp. 45–52.
- Fliedner, G. and D. Bobbert (2003). A framework for information-state based dialogue (demo abstract). In *Proceedings of the 7th workshop on the semantics and pragmatics*

of dialogue *DiaBruck*, Saarbrücken.

- Graesser, A. C., K. Wiemer-Hastings, P. Wiemer-Hastings, and R. Kreuz (1999). Auto-tutor: A simulation of a human tutor. *Cognitive Systems Research* 1, 35–51.
- Huang, X. (1994). Reconstructing proofs at the assertion level. In A. Bundy (Ed.), *Proceedings of the 12th Conference on Automated Deduction*, pp. 738–752. Springer.
- Martin, D. L., A. J. Cheyer, and D. B. Moran (1999). The Open Agent Architecture: A framework for building distributed software systems. *Applied Artificial Intelligence* 13(1–2), 91–128.
- McTear, M. (1998). Modelling spoken dialogues with state transition diagrams: Experiences with the CSLU toolkit. In *Proceedings of the 5th International Conference on Spoken Language Processing*, Sydney, Australia.
- Melis, E. and J. Siekmann (1999). Knowledge-based proof planning. *Artificial Intelligence Journal* 115(1), 65–105.
- Pinkal, M., J. Siekmann, and C. Benz Müller (2001). Projektantrag Teilprojekt MI3 — DIALOG: Tutorieller Dialog mit einem mathematischen Assistenzsystem. In *Fortsetzungsantrag SFB 378 — Ressourcenadaptive kognitive Prozesse*, Universität des Saarlandes, Saarbrücken, Germany.
- Pinkal, M., J. Siekmann, and C. Benz Müller (2004). Dialog: Tutorial dialog with an assistance system for mathematics. Project report in the Collaborative Research Centre SFB 378 on Resource-adaptive Cognitive Processes.
- Pinkal, M., J. Siekmann, C. Benz Müller, and I. Kruijff-Korbayova (2004). Dialog: Natural language-based interaction with a mathematics assistance system. Project proposal in the Collaborative Research Centre SFB 378 on Resource-adaptive Cognitive Processes.
- Siekmann, J. and C. Benz Müller (2004). Omega: Computer supported mathematics. In *Proceedings of the 27th German Conference on Artificial Intelligence (KI 2004)*, Ulm, Germany.
- Siekmann, J., C. Benz Müller, A. Fiedler, A. Meier, I. Normann, and M. Pollet (2003). *Proof Development in OMEGA: The Irrationality of Square Root of 2*, pp. 271–314. Kluwer Applied Logic series (28). Kluwer Academic Publishers. ISBN 1-4020-1656-5.
- Traum, D., J. Bos, R. Cooper, S. Larsson, I. Lewin, C. Matheson, and M. Poesio (1999). A model of dialogue moves and information state revision. Technical report TRINDI project deliverable D2.1, University of Gothenburg.
- Traum, D. and S. Larsson (2003). The information state approach to dialogue management. In J. van Kuppevelt and R. Smith (Eds.), *Current and new directions in discourse and dialogue*. Kluwer. <http://www.ict.usc.edu/~traum/Papers/traumlarsson.pdf>.
- Traum, D. R. (1999). Computational models of grounding in collaborative systems. In S. E. Brennan, A. Giboin, and D. Traum (Eds.), *Working Papers of the AAAI Fall*

- Symposium on Psychological Models of Communication in Collaborative Systems*, Menlo Park, California, pp. 124–131. American Association for Artificial Intelligence.
- Walker, M. A., D. J. Litman, C. A. Kamm, and A. Abella (1997). PARADISE: A framework for evaluating spoken dialogue agents. In P. R. Cohen and W. Wahlster (Eds.), *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, Somerset, New Jersey, pp. 271–280. Association for Computational Linguistics.
- Wolska, M., B. Q. Vo, D. Tsovaltzi, I. Kruijff-Korbayova, E. Karagjosova, H. Horacek, M. Gabsdil, A. Fiedler, and C. Benzmüller (2004). An annotated corpus of tutorial dialogs on mathematical theorem proving. In *Proceedings of International Conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Portugal. ELDA.