A 3-tier Planning Architecture for Managing Tutorial Dialogue*

Claus Zinn, Johanna D. Moore, and Mark G. Core

Division of Informatics, University of Edinburgh
2 Buccleuch Place, Edinburgh EH8 9LW, UK
[zinn|jmoore|markc]@cogsci.ed.ac.uk

Abstract. Managing tutorial dialogue is an intrinsically complex task that is only partially covered by current models of dialogue processing. After an analysis of such models identifying their strengths and weaknesses, we propose a flexible, modular, and thus re-usable computational framework, centered around a 3-tier dialogue planning architecture.

1 Motivation

Managing dialogue is an intrinsically complex task. Lewin *et al.* [1] note that dialogue management includes *turn-taking management*: who can speak next, when, and for how long; *topic management*: what can be spoken about next; *utterance understanding*: understanding the content of an utterance in the context of previous dialogue; *intention understanding*: understanding the point or aim behind an utterance in the context of previous dialogue; *context maintenance*: maintaining a dialogue context; *intention generation*: generating a system objective given a current dialogue context; and *utterance generation*: generating a suitable form to express an intention in the current dialogue context.

Now, how does dialogue management instantiate to the genre of tutorial dialogue? The work of Chi *et al.* [2] and our own analysis of a corpus of human tutorial dialogue indicate that a tutorial dialogue manager must (1) understand student utterances well enough to respond appropriately; (2) not ignore student confusion; (3) encourage the student to recognise and correct their own errors; (4) abandon questions that are no longer relevant; (5) handle multiple student actions in one turn; and (6) deal with student-initiated topic changes.

Whereas (1) stresses the need for a reasonably well performing input understanding engine, the latter tasks require the tutorial agent to monitor the execution of its dialogue strategies. In the case of failure, the agent needs to adapt its plan to the new situation: inserting a plan for a sub-dialogue to handle student confusion or a misconception (2+3), deleting parts of a dialogue plan because their effects are now irrelevant or already achieved (4+5), or reorganising sub-plans to handle topic changes (6). Consequently, there is no need for tutorial dialogue managers to generate elaborate discourse plans much in advance. Given the dynamics of tutorial dialogue — the large number of potential student

* The research presented in this paper is supported by Grant #N00014-91-J-1694 from the Office of Naval Research, Cognitive and Neural Sciences Division.

actions at any point and the limited ability of the tutor to predict them — it is a more viable approach to enter a tutorial conversation with a sketchy high-level dialogue plan. As the dialogue progresses, the dialogue manager then refines the high-level plan into low-level dialogue activities by considering the incrementally constructed dialogue context. The dialogue manager therefore interleaves high-level tutorial planning with on-the-fly situation-adaptive plan refinement and execution.

2 The State-of-the-art in Tutorial Dialogue Management

2.1 Models of Dialogue Processing

We review three industrial-strength models of dialogue processing, namely, *finite state machines* (FSMs), *form-filling*, and *dialogue components*, as well as an interesting cross-breed of FSMs and planning.

The FSM approach to dialogue management is characterised by defining a finite state automaton that contains all plausible dialogues. Necessarily, the dialogues are system-driven, turn-taking as well as system feedback are hardwired, and there is only a limited and well-defined amount of information available in each state of the network. A typical industrial dialogue system in the area of banking has 1500 states (personal communication, Arturo Trujillo, Vocalis plc).

Form-filling is a less rigid approach to dialogue management. Instead of anticipating and encoding all plausible dialogues, the dialogue engineer specifies the information the dialogue system must obtain from the user as a set of *forms* composed of *slots*. The structural complexity of possible dialogues is limited only by the form design and the intelligence of the form interpretation and filling algorithm. This algorithm may be able to fill more than one slot at a time, maintain several active forms simultaneously, and switch among them. In contrast to a FSM-based dialogue system, the user of a form-filling dialogue system can therefore supply more information than the system requested, or start a task before the system has offered to perform it. Form-filling systems are thus capable of performing *question accommodation* or *task accommodation*, respectively.

Dialogue components provide a modular add-on to dialogue engineering. In this paradigm, a dialogue, such as a bill payment dialogue, is divided into sub-dialogues, say, one to obtain a customer name and one to obtain his/her credit card number. Such sub-dialogues are specified in an object-oriented language, which allows the capture of common features. Atomic dialogues or *components* (*e.g.,* date) can be combined into complex dialogues called tasks (*e.g.,* bill payment=date+amount+bill-type), and tasks can be combined into *applications* (*e.g.,* banking=balance+bill payment+money transfer).

Although such dialogue components bring modularity to the form-filling approach, the problems of choosing among multiple refinements of a task, identifying which task the user is currently trying to perform, and detecting user-initiated switches among or abandonment of tasks, remain difficult and open. Also, as in the other two approaches, no global dialogue history is maintained.

This makes it hard if not impossible to properly handle meta-dialogues and other linguistic phenomena (*e.g.,* anaphora, ellipses) as well as to generate natural and effective feedback.

A major advantage of the three approaches is the robustness of their language understanding capabilities. Each approach has strong expectations about user input based on the state of the system. In the FSM approach, the nodes of the network can be attached to special grammars and language models; in the form-filling approach, one can associate actions with slot-filling events, for example, controlling the activation and combination of scoped grammars; in the dialogue components approach, grammars and language models may be associated with both atomic and complex components.

In each of the three approaches, all plausible dialogues (FSMs) or their content (form filling, dialogue components) have to be specified in advance. None of the approaches involve a deliberative component that can generate dialogue plans to achieve underlying goals, albeit dialogue components can be seen as instantiated plans. A deliberative component also proves useful in cases where the execution of the first planned dialogue strategy fails and re-planning is needed.

Dialogue management in the AUTOROUTE system combines deliberative planning with FSMs in a two tier approach [3]. The bottom tier consists of *dialogue games*, which are *generic* FSMs. Dialogue games encode typical adjacency pairs (*e.g.,* a question is followed by an answer which may be followed by a confirmation and an acknowledgement) that do not specify the content of the dialogue moves they contain. In the top-tier, a deliberative agent treats the I/O behaviour of a dialogue game as a primitive action. Given a goal, it generates a plan that has instantiated dialogue games as its primitive steps (*e.g.,* to pay a bill, obtain the date, amount, and bill-type; to obtain a date, play a question/answer game). Thus far, the AUTOROUTE approach has only been applied to the genre of information-seeking dialogues. Its planning architecture has not been fully exploited, and its re-planning capabilities allow only trivial cases of question and task accommodation. Moreover, turn taking is hardwired into the FSMs, which makes it hard to cope with situations where the user grabs the turn.

2.2 Previous Work in Tutorial Dialogue Systems

Existing tutorial dialogue systems perform dialogue management in an *ad hoc* manner. They adopt none of the aforementioned models of dialogue processing in their pure form; mainly, because none of the models explain how to generate effective tutorial feedback. We review the dialogue management in the tutoring systems AUTOTUTOR (domain: computer literacy) [4], ATLAS-ANDES (Newtonian mechanics) [5], and CIRCSIM/APE (circulatory system) [6] as well as in the EDGE explanation system (electrical devices) [7].

AUTOTUTOR's dialogue management can be regarded as an adaption of the form-filling approach to tutorial dialogue; to solve the feedback generation problem, it adds feedback moves to slots. AUTOTUTOR's dialogue management relies on a *curriculum script*, a sequence of *topic formats*, each of which contains a *main focal question*, and an *ideal complete answer*. The ideal complete answer consists of several sub-answers, called *aspects*. Each aspect includes: a list of anticipated bad answers corresponding to misconceptions and bugs that need correction; lists of prompts and hints that can be used to get the learner to contribute more information; and elaboration and summary moves that can be used to provide the learner with additional or summarising information. All of the moves are hard coded in English.

Using latent semantic analysis, AUTOTUTOR evaluates the student's answer to the main focal question against all the aspects of its ideal complete answer, and the anticipated bad answers. AUTOTUTOR gives immediate feedback based on the student's answer, and then executes dialogue moves that get the learner to contribute information until all answer aspects are sufficiently covered. The category and content of tutor dialogue moves are computed by a set of 20 fuzzy production rules and an algorithm that selects the next answer aspect to focus on.

While AUTOTUTOR's dialogue management performs well in the descriptive domain of computer literacy, it is unclear how well this approach will work in problem-solving domains such as algebra or circuit trouble-shooting. In these domains student answers will often require the tutor to engage the student in a multi-turn scaffolding or remediation sub-dialogue. Curriculum scripts are not nested and do not allow the representation of multi-turn dialogues.

The dialogue manager of **ATLAS-ANDES** teaches Newtonian mechanics and thus must address AUTOTUTOR's aforementioned limitations. It uses a combination of *knowledge construction dialogues (KCDs)*, which are recursive FSMs, and a generative planner [8]. The grammar of a KCD bears many similarities to an AUTOTUTOR curriculum script. A student answer to a tutor question can be divided into correct and incorrect sub-answers with associated tutorial remediations. Unlike AUTOTUTOR, this feedback may extend over multiple turns through the use of recursive KCDs.

While AUTOTUTOR requires a pre-defined and hand-crafted curriculum script, the ATLAS-ANDES approach allows on-the-fly generation of nested KCDs, using the APE discourse planner. The ATLAS-ANDES architecture is therefore similar to the 2-tier AUTOROUTE architecture: its KCDs are dialogue games that are more complex but domain-specific. The simple generic question-answer pair is replaced by a recursive automaton that can deal with a specific question and many anticipated possible correct and incorrect sub-answers. A compiler then maps KCDs into plan operators, which are used by APE to combine KCDs into larger recursive FSMs. Moreover, the developers of ATLAS-ANDES propose a solution to the rigidity that is typically associated with FSM-based systems. The reactive component of APE can skip around in the recursive KCD, for example, it can pop sub-networks that ATLAS-ANDES believes contain intentions that were already dealt with in prior dialogue.

In contrast, dialogue management in **EDGE** and **CIRCSIM/APE** is purely plan-based. EDGE provides two types of (STRIPS-like) operators: *discourse* and *content* operators. Discourse operators model Sinclair & Coulthard's four levels of discourse, namely, *transaction*, *exchange*, *move*, and *act* [9]. Content operators specify how to explain something. For example, "to describe a device: explain its function, its structure, and its behaviour". EDGE's content operators are quite general; their bodies contain abstract domain references that interface with a knowledge representation module. EDGE incrementally builds and executes plans. Before each tutor turn, the deliberative planner expands the current unfinished step with either a complex sub-plan or an elementary plan step. Elementary plan steps are then executed using simple template driven generation. Thus, planning is delayed as much as possible so that the most current student model can be consulted.

CIRCSIM/APE also incrementally constructs and executes plans, and uses simple template driven generation for realising elementary plan steps. However, a major drawback of CIRCSIM/APE (for others, see [10]) is that it embeds control in operators, unlike traditional planners, where control is separated from action descriptions. This makes writing operators difficult and puts an additional burden on the planner.

Conclusion. Although previous and ongoing work in tutorial dialogue systems has striven to support unconstrained natural language input and multi-turn tutorial strategies, there remain limitations that must be overcome: teaching strategies, encoded as curriculum scripts, KCDs, or plan operators, are domain-specific; the purely plan-based systems embed control in plan operators or, necessarily, conflate planning with student modelling and maintenance of the dialogue context; and all current tutorial dialogue systems except EDGE mix high-level tutorial planning with low-level communication management. These limitations can make systems difficult to maintain, extend, or reuse.

There is great benefit to be gained from integrating dialogue theories and dialogue system technology that have been developed in the computational linguistics and spoken dialogue systems communities with the wealth of knowledge about student learning and tutoring strategies that has been built up in the ITS community. It is therefore worth considering dialogue systems not designed for tutoring [3, 11–15]. These systems do not allow for conversational moves extending over multiple turns and the resulting need to abandon, suspend, or modify these moves. However, these systems aim for dialogue strategies that are independent of dialogue context management and communication management concerns. These strategies contain no domain knowledge; they query domain reasoners to fill in necessary details. Furthermore, in systems explicitly performing dialogue planning, control is never embedded in plan operators. Our goal is to combine these beneficial features (modularity and re-usability) with the flexibility and educational value of tutorial systems with reactive planners.

3 A 3-tier Architecture for Managing Tutorial Dialogue

We present a generic and modular architecture for the management of tutorial dialogue. It allows the effective combination of pedagogical strategies, domain

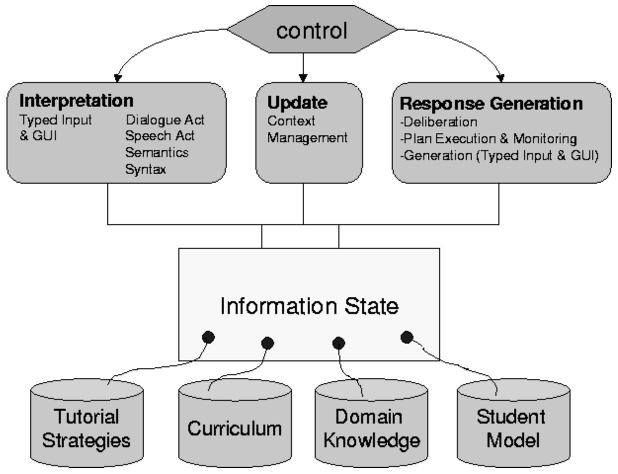

Fig. 1. A Modular Dialogue System Architecture.

knowledge, and dialogue management strategies, clearly separating the knowledge sources involved. Our overall architecture is depicted in Fig. 1. There are three major modules, *interpretation*, *update*, and *response generation*. Each of the modules can access the *information state (IS)*, which captures the overall dialogue context and interfaces with external knowledge sources (*e.g.,* student model, domain reasoner, curriculum). In particular, the IS will need to contain a dialogue history that records all prior dialogue moves, and the *common ground*, the set of propositions that both dialogue partners have agreed upon in prior discourse. The IS also contains a list of salient objects to facilitate the treatment of linguistic phenomena such as anaphora. Moreover, the IS maintains any pending discourse obligations and provides access to the tutorial dialogue plan. This allows the system to deduce the issues that it needs to, or intends to address in future dialogue continuations.

In the remainder of this section, we describe only briefly the interpretation and update modules. We then focus on the response generation module that is based upon a 3-tier planning architecture, which we regard as enabling technology for managing tutorial dialogue.[1]

The interpretation module allows the student to interact with the system via text and graphical means. It then identifies the meaning and intention behind a student utterance. This includes its syntactic analysis, the construction of its propositional content, the recognition of its speech act, and the recognition of the intent behind the utterance (*i.e.,* the dialoge act). The latter is complemented by an evaluation of the student's answer or action for correctness).

The update module maintains the context. It is clearly separated from the response generation module. *Update rules* encode conversational expertise and define how to update the current context given a new dialogue act. Two example update rules are given below:

[1] An example that explains how we can simulate human-human tutorial dialogues in the 3-tier architecture is available from the authors upon request.

```
do DiagQuery:
   precond:     latest move is of type DIAG_QUERY
   effects:     add latest move to CDU
                add obligation for hearer to address move to CDU
doAssert:
   precond:     latest move is of type assert
                asserted content addresses a previous DIAG_QUERY or INFO_REQ, say Q1
   effects:     add latest move to CDU
                remove speaker's obligation to address Q1
                add hearer's obligation to address assertion
                add that speaker is committed to propositional content of assertion
                add that if hearer accepts assertion, then add assertion to common ground
```

The *update rule engine* fires the rule *doDiagQuery* if the latest dialogue move is a diagnostic query. As a consequence, such a move would be entered as part of the current discourse unit (CDU), and an obligation for the hearer to address this move would be created. The rule *doAssert* fires if the latest move is an assertion and if its asserted content addresses a previous diagnostic query or information request. If this is the case, then the latest move is added to the CDU, the speaker's obligation to address the question is deleted, the hearer is now obliged to address the assertion, and the propositional content of the assertion is a candidate for entering the common ground.

The response generation module computes appropriate tutorial moves and synthesises tutorial feedback as text or other modalities using a 3-tier planning architecture. Fig. 2 depicts its three levels: a *deliberative planner* that projects

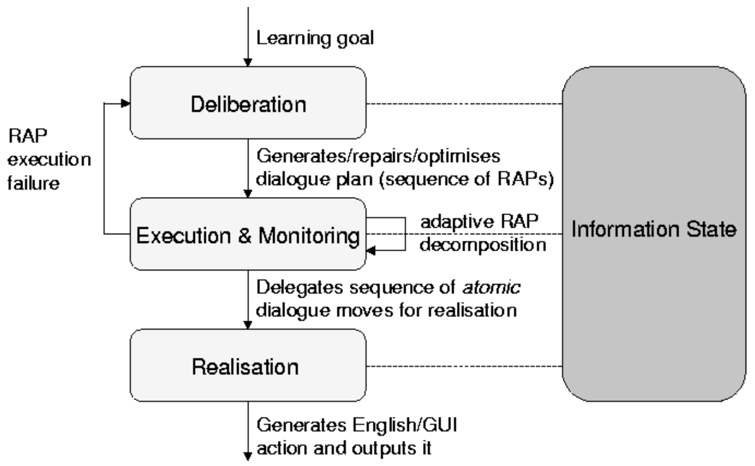
Learning goal
Deliberation
RAP execution failure
Generates/repairs/optimises dialogue plan (sequence of RAPs)
Execution & Monitoring
adaptive RAP decomposition
Information State
Delegates sequence of *atomic* dialogue moves for realisation
Realisation
Generates English/GUI action and outputs it

Fig. 2. A 3-tier Dialogue Planning Architecture

the future, anticipates and solves problems (top layer); a *plan execution and monitoring* system that performs adaptive on-the-fly refinement (middle layer); and an *action system* that performs primitive actions (bottom layer).

Top-tier: Deliberative Planning. The deliberation component synthesises plans from action descriptions at the highest possible level of abstraction. This abstraction not only minimises wasted effort, but also allows the middle layer to perform a maximum of situation-adaptive plan refinement. High-level planning results in a structured sequence of tasks that are passed to the *task agenda* of the middle layer; for each task, there is a reactive action package (see below) that achieves it, if successfully executed.

The deliberator is explicitly activated in two cases: when the tutor agent enters a tutorial dialogue, and during the dialogue when the middle layer fails to perform plan execution and therefore asks the top layer to perform a plan repair. However, the deliberative component has also a permanent background activity. It regularly inspects the agenda of the middle layer for two reasons: verifying whether pending discourse obligations (as recorded in the IS) are covered by the contents of the agenda, and searching to anticipate problems or to optimise the agenda's content. In both cases, the top layer can add or delete items from the task agenda as well as reorganise or aggregate them. The top and middle layer therefore need to synchronise their access to the agenda.

Middle-tier: Context-Driven Plan Refinement. Reactive Action Packages [16] are the basic blocks of a situation-driven plan refinement system. A Reactive Action Package (RAP) groups together and describes all ways to carry out a specific task in different situations. Fig. 3 displays (a simplified version of) the RAP *instruct_step*. It has three possible situation-specific ways of getting the student to perform a step in a procedure. Method M1 is applicable in any context. It

```
RAP:       instruct_step(?step)
precond:   nil
effects:   instructed_step(?step)

method:     M1
  context:  nil
  tasks:
      t1: sequence assert(next_step(?step))
                   direct(do(?step))
      t2: supply_feedback(did(?step))

method:     M2
  context:  didactic_mode(level(2))
            get_howto(?step, ?howto)
  tasks:
      t1: sequence assert(next_step(?step))
                   assert(howto(?step, ?howto))
                   check_student_understanding(ok)
      t2: direct(do(?step))
      t3: supply_feedback(did(?step))

method: M3
  contextsocratic_mode(level(2))
         get_effects(?step, ?effects)
  tasks:
      t1: sequence assert(next_desired_effects(?effects))
                   diag_query(which_action_achieves(?effects))
      t2: supply_feedback(answered(which_action_achieves(?effects)))
      t3: direct(do(?step))
      t4: supply_feedback(did(?step))
```

Fig. 3. A RAP for Instructing a Step of a Procedure.

spawns two sub-tasks, namely, the primitive task of generating a sequence of elementary dialogue moves, and the complex task of supplying feedback to the student's moves.

The RAP interpreter executes the contents of the agenda as follows. First, it selects the next task from the agenda. Then, it checks the selected task against the information state to see whether its effects have already been achieved. If this is the case, the task is deleted. Otherwise the interpreter identifies the RAP that can achieve the task. The methods of the identified RAP are checked, and the most appropriate of them is selected. If the chosen method is a primitive action, then it is delegated to the bottom layer for execution; if the method is a network of subtasks, then each subtask is put on the agenda, and a new interpretation cycle starts.

The execution of a RAP can fail for three reasons: its preconditions are not met, none of its methods are applicable, or the execution of one of its primitive methods fails. The RAP interpreter can cope with some failures, *e.g.,* it can try another applicable method. In the other cases, it has to call the top layer to cope with the failure.

Bottom-Tier: Action Execution. The bottom-tier is responsible for the execution of primitive dialogue actions. It gets a sequence of elementary speech acts and micro-plans the generation of multi-modal feedback (natural language utterances and GUI actions). The bottom-tier is supported by a sentence and media planner, both of which have access to the full dialogue context. In particular, these components consult the list of salient objects and the contents of the previous and current discourse unit to generate natural feedback that makes use of elliptical constructions and anaphoric expressions. Action execution fails if the micro-planner fails.

Turn-Taking Management. The tutoring agent releases the turn after it asks a question or requests that the student perform an action. In all other cases, the RAP interpreter “cycles” until a question or action request is generated. If the student takes the initiative and grabs the turn, then this dialogue act will be recorded in the IS, generating an obligation for the tutor to address the last student utterance. The top-layer then deliberates over the new situation and may change the contents of the agenda accordingly. Similarly, if the student fails to react within a time limit, then the interpretation module generates an appropriate dialogue act, which the update engine processes, generating an obligation for the tutor to address the student’s silence. The deliberative planner can then decide to either give the student more time, or to take the turn to supply help.

4 BEE Tutorial Learning Environment — BEETLE

We have built BEETLE, a prototype implementation of our computational framework for managing tutorial dialogue that serves to both validate and propagate our ideas. It is primarily based on two technologies, the TRINDIKIT dialogue system shell [17] and the Open Agent Architecture (OAA) [18]. In line with our goal to provide a flexible, modular, and thus reusable architecture, domain reasoning is performed by a single agent. The BEER agent encodes BEETLE’s knowledge about basic electricity and electronics. It has a rich LOOM representation of BEE concepts, can perform basic inferences in this domain, and has explicit representations of domain plans. BEETLE’s deliberative planner, LONGBOW [19], creates instantiated dialogue plans from its general dialogue strategies by accessing the relevant domain knowledge represented in BEER.

At the time of writing, our 3-tier planning architecture is only partially implemented. The next major step is to replace LONGBOW and our hand-built plan execution and refinement layer with a state-of-the-art planning environment. We are currently investigating the use of the Open Planning Architecture (O-Plan) [20] and its successor I-Plan.

References

1. Lewin, I., Rupp, C.J., Hieronymus, J., Milward, D., Larsson, S., Berman, A.: Siridus system architecure and interface report. Tech. report, Siridis D6.1 (2000)
2. Chi, M.T.H., Siler, S.A., Jeong, H., Yamauchi, T., Hausmann, R.G.: Learning from human tutoring. Cognitive Science **25** (2001) 471–533
3. Lewin, I.: Autoroute dialogue demonstrator. Technical Report CRC-073, SRI Cambridge (1998)
4. Graesser, A.C., Wiemer-Hastings, K., Wiemer-Hastings, P., Kreuz, R.: AutoTutor: A simulation of a human tutor. Cognitive Systems Research **1** (1999) 35–51
5. Schulze, K.G., Shelby, R.N., Treacy, D., Wintersgill, M.C., VanLehn, K., Gertner, A.: Andes: A coached learning environment for classical newtonian physics. The Journal of Electronic Publishing **6** (2000)
6. Khuwaja, R.A., Evens, M.W., Michael, J.A., Rovick, A.A.: Architecture of CIRCSIM-tutor (v.3). In: Proceedings of the 7th Annual IEEE Computer-Based Medical Systems Symposium, IEEE Computer Society Press (1994) 158–163
7. Cawsey, A.: Explanatory dialogues. Interacting with Computers **1** (1989) 69–92
8. Jordan, P.W., Rosé, C., VanLehn, K.: Tools for authoring tutorial dialogue knowledge. In Moore, J.D., Redfield, C.L., Johnson, W.L., eds.: 10th International Conference on Artificial Intelligence in Education, IOS Press (2001) 222–233
9. Sinclair, J.M., Coulthard, R.M.: Towards an Analysis of Discourse: The English used by teachers and pupils. Oxford University Press (1975)
10. Freedman, R.: An approach to increasing programming efficiency in plan-based dialogue systems. In Moore, J.D., Redfield, C.L., Johnson, W.L., eds.: 10th International Conference on Artificial Intelligence in Education, IOS Press (2001)
11. Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L., Stent, A.: An architecture for a generic dialogue shell. Natural Language Engineering **6** (2000)
12. Pieraccini, R., Levin, E., Eckert, W.: AMICA: The AT&T mixed initiative conversational architecture. In: Proceedings of the 5^{th} European Conference on Speech Communication and Technology (Eurospeech-97). (1997)
13. Larsson, S., Ljungloef, P., Cooper, R., Engdahl, E., Ericsson, S.: GoDiS - an accommodating dialogue system. In: Proceedings of ANLP/NAACL-2000 Workshop on Conversational Systems. (2000)
14. Rudnicky, A., Xu, W.: An agenda-based dialog management architecture for spoken language systems. In: Intl. Workshop on Automatic Speech Recognition and Understanding. (1999)
15. Chu-Carroll, J.: Form-based reasoning for mixed-initiative dialogue management in information-query systems. In: Proceedings of the 7^{th} European Conference on Speech Communication and Technology (Eurospeech-99). (1999) 1519–1522
16. Firby, R.J.: Adaptive Execution in Complex Dynamic Domains. PhD thesis, Yale University (1989) Technical Report YALEU/CSD/RR#672.
17. Larsson, S., Traum, D.: Information state and dialogue management in the TRINDI dialogue move engine toolkit. Natural Language Engineering **6** (2000) 323–340
18. Martin, D.L., Cheyer, A.J., Moran, D.B.: The open agent architecture: A framework for building distributed software systems. Applied Artificial Intelligence: An International Journal **13** (1999) 91–128
19. Young, R.M., Pollack, M.E., Moore, J.D.: Decomposition and causality in partial order planning. In: Proceedings of the Second International Conference on Artificial Intelligence and Planning Systems, Morgan Kaufman (1994) 188–193
20. Currie, K., Tate, A.: O-Plan: the open planning architecture. Artificial Intelligence **52** (1991) 49–86