The Queen's Agents: Using Collaborating Object-Based Dialogue Agents in the Queen's Communicator

Ian O'Neill, Philip Hanna, Xingkun Liu
School of Computer Science
Queen's University
Belfast BT7 1NN, N. Ireland
{i.oneill, p.hanna, xingkun.liu}@qub.ac.uk

Michael McTear
School of Computing and Mathematics
University of Ulster
Jordanstown, BT37 0QB, N. Ireland
mf.mctear@ulster.ac.uk

Abstract

A dialogue manager provides the decision making at the heart of a spoken dialogue system. In an object-oriented approach to dialogue management, generic behaviour, such as confirming new or modified information that has been supplied by the user, is inherited by more specialised classes. These specialised classes either encapsulate behaviour typical of a particular business domain (service agents) or make available dialogue abilities that may be required in many business domains (support agents). In this paper we consider the interplay between the agents' generic and specialised behaviour and consider the manner in which service and support agents collaborate within and across their respective groups.

1 Object-orientation and cross-domain, mixed initiative dialogue

Object-orientation provides an intuitive separation of, on the one hand, inheritable generic functionality and, on the other hand, domain-specific, specialized functionality that is supported by the generic elements of the system. Applied to the area of natural language dialogue, this has enabled us to create a generic, automated dialogue confirmation strategy – based on confirmation statuses and discourse pegs (see Section 3.3) – which supports domain-specific strategies to gather and provide information relating to particular transactions – for example booking a hotel or finding out about cinema times. Heuristics, or expert rules, specific to each transaction domain, prompt the user for significant missing information or assist the user by providing choices from a database (e.g. names of available hotels).

Thus, while our generic confirmation strategy ensures that information newly supplied by the user is confirmed, and information changed is reconfirmed, and so on, the nature of that information may differ significantly from domain to domain. Likewise the system may respond to confirmed information in quite different ways depending on the domain – as it either completes a domain-specific transaction or attempts to elicit important missing information from the user.

In the Queen's Communicator dialogue system, expertise for different transaction domains is encapsulated within corresponding expert classes or 'agents'. We have used this to our advantage by enabling the system to transfer between domains either at the user's or the system's initiative – in a mixed initiative dialogue either the user or the system may introduce new topics. Agents are able to announce their abilities to the system at large, or indeed to the user. Thus, when key words or phrases uttered by the user indicate that the topic of conversation has turned, for example, from accommodation booking to payment, the system's DomainSpotter (see Section 4) can ask the agents if any of them deal with payments. The most suitable agent is then given the task of managing the specialised subdialogue.

2 Spoken dialogue management

A spoken dialogue system typically comprises a number of components: an automatic speech recogniser, a semantic parser, a dialogue manager (DM), a database 'back-end', a natural language generator, and a text-to-speech engine. The focus of our present research is the development of an object-based DM that can support mixed initiative dialogues that involve a number of business domains.

Our DM operates within the DARPA[1] Communicator architecture, which is based on the Galaxy hub – a software router developed by the Spoken Language Systems group at MIT (www.sls.csail.mit.edu/sls/technologies/galaxy.shtml) and subsequently released as an open source package in collaboration with the MITRE Corporation (fofoca.mitre.org). In the 'Queen's Communicator' dialogue system, our newly developed DM interacts with a number of off-the-shelf components. For semantic parsing we use Phoenix (W. Ward, 1994), available from the

[1] Defense Advanced Research Projects Agency

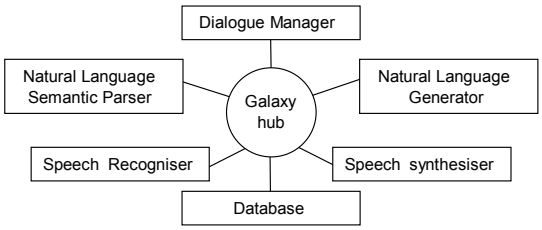

Figure 1. DARPA Communicator architecture.

University of Colorado's 'CU Communicator' download (communicator.colorado.edu). For recognition we use the Microsoft English ASR Version 5 Engine as supplied with Windows XP. Synthesised speech is provided by Festival (www.cstr.ed.ac.uk/projects/festival/), also taken from the CU download. Figure 1 shows a typical Communicator configuration.

The DM itself embodies decision-making similar to that of a human interlocutor as it queries, responds to and informs the user. Moreover, in mixed initiative dialogues that deal with more than one domain (e.g. enquiries about accommodation and events, and supporting exchanges about payment and addresses), the system has the additional task of identifying the (ongoing) topic of the dialogue and applying appropriate dialogue management expertise.

3 Object-based dialogue agents

3.1 A multi-agent approach to dialogue management

In order to enable mixed initiative interactions across domains, we model the system's behaviour as a collaboration between the cohort of implemented agents. Other developers have also adopted an agent-based approach to dialogue, though sometimes dialogue agents each perform very simple tasks rather than engage in extensive discourse: in (Turunen and Hakulinen, 2001) for example, simple generic error-handling agents, based on Java and XML, ask the user to repeat misunderstood input. In our case an agent is a specialist in a particular transactional area – e.g. booking accommodation or eliciting an address. An agent uses its own domain-specific 'expert-rules' to elicit information (e.g. information for making a hotel booking) that is then stored in a specialised dialogue frame. Each agent thus encapsulates a skillset for a substantial dialogue or subdialogue.

Like the Communicator team at Carnegie Mellon University, we view the dialogue product (the knowledge to be elicited) as a tree-like structure (Rudnicky and Xu, 1999) – though for us the nodes are complete dialogue frames rather than individual data items. In the Queen's Communicator the discourse structure evolves dynamically as agents are selected by a DomainSpotter, in the light of the user's utterances or as a consequence of the agents' own rules. It is this process, rather than an overarching dialogue plan or agenda, that drives the discourse forward, sometimes across domain boundaries. We do, however, maintain an ExpertFocusStack, which contains, in sequence, the name of the agent that is currently handling the dialogue and the names of agents that have last handled the dialogue and have unfinished business: this allows the system to quickly identify the current handler and to pass control back, once the current handling agent is finished.

3.2 Inherited and domain-specific behaviour

Our dialogue manager is implemented as a suite of Java classes (see Figure 2). The object-based approach (Booch, 1994) (O'Neill and McTear, 2000) has afforded us certain advantages. The domain specialists or 'Experts' within our system – AccommodationExpert, TheatreExpert, CinemaExpert, CreditCardExpert, etc. – all inherit generic dialogue handling skills from a DiscourseManager, whose role is to ensure that new information provided by the user is at least implicitly confirmed, and information that is changed or negated is subjected to more detailed, explicit confirmation (O'Neill and McTear, 2002) (O'Neill et al. 2003). The domain experts encapsulate specialised behaviour, which can be readily extended by additional classes. There are two families of domain experts:

- 'service agents' that provide front-line services to the user – like AccommodationExpert, whose behaviour emulates that of a human booking clerk, and
- 'support agents' like CreditCardExpert that are able to elicit information required to complete one of the front-line service transactions.

We refer to the corresponding discourse segments as 'service' and 'support' dialogues respectively. By assigning the agents (and the corresponding dialogues) to one of two families we give ourselves the option of restricting user-led transitions between main and ancillary transactions. However, the overall objective of our implementation is to maintain a high degree of flexibility in the manner in which the system reacts to unsolicited user utterances.

3.3 Using frames of information

The agents, whether they provide service or support, collect and manipulate frames of

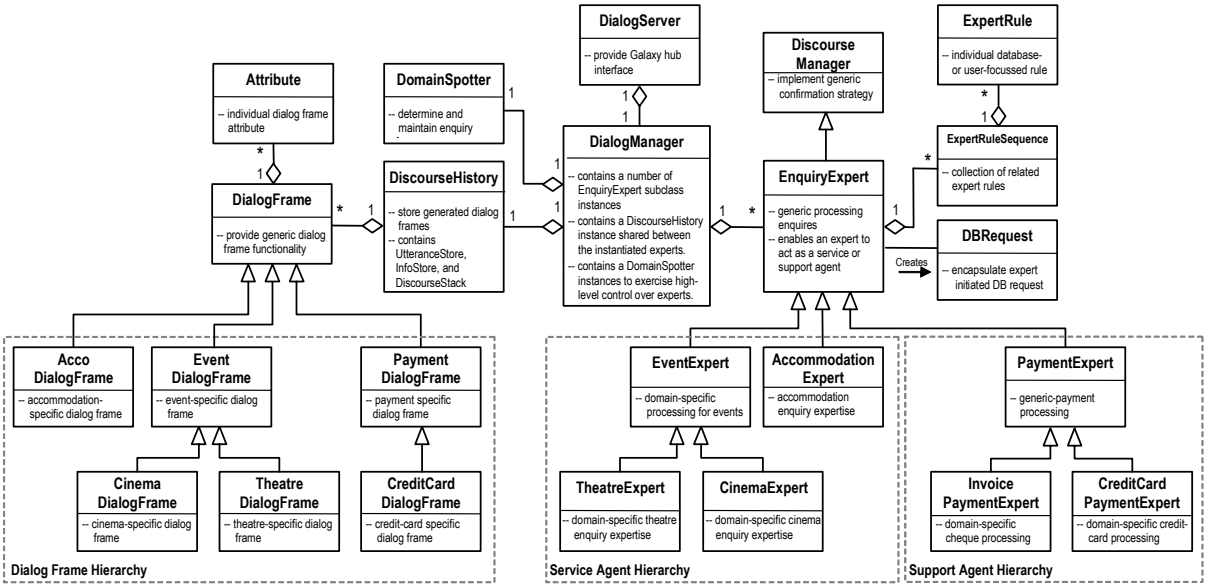

Figure 2: Class diagram of the dialogue manager.

information related to their own sphere of competence. The frames consist of Attribute objects, each of which stores:

- the type and elicited value of a single piece of information (datum);
- the confirmation status of the datum (e.g. new_for_system);
- the level to which the datum has been confirmed (through repetition, or by the user's affirmative response to a system prompt – the level is represented by a simple numeric 'peg');
- and the system intention regarding the datum (e.g. implicitly confirm new information; explicitly confirm information that has been negated; ask the user to specify information that is still required) (Heisterkamp and McGlashan, 1996).

The Attribute objects thus give a multi-facetted view of each piece of information that it is being considered by the system. The evolving domain-specific (and thus generally agent-specific) frames of Attributes are maintained on a DiscourseStack within the DiscourseHistory object. The agents use this stack to implement the inherited generic confirmation strategy. The frames of information are typically populated in the course of several discourse turns, as new or additional information is acquired from successive user-system interactions. Once it is handling a particular discourse segment, an agent uses its inherited confirmation strategy to compare the latest values in its current dialogue frame with the corresponding values and system intentions in the previous iteration of that frame. Thus the agent is able to determine which values have been confirmed (e.g. the user has not challenged an implicit confirmation request by the system) and which have been modified or negated.

3.4 Applying expert rules

In addition to its inherited confirmation strategies, each of the domain Experts, whether a service agent or a support agent, has its own expert rules, contained in one or more expert rule sequences. Typically the expert rule sequences will be of one of two kinds:

- 'user-focussed rules', which determine the agent's reaction to particular combinations of information supplied by the user – must the system now ask a follow-up question, must it perform a database look-up, or can it conclude a transaction ? – and
- 'database-focussed rules', which represent the agent's dialogue furthering strategy when database queries based on user-supplied combinations of information fail: because of its access to database content, the system may be able to modify a user-supplied constraint and so formulate a database query that will succeed (e.g. the system might suggest a four-star hotel if it cannot meet the user's request for a five-star hotel in a particular locality.)

These rules, encapsulated within the appropriate agent (e.g. AccommodationExpert), are applied to information that the agent has 'phrase-spotted' and placed in the appropriate dialogue frame (e.g. an AccommodationDialogueFrame). Sequences of rules, encapsulated within service and support agents and tested to see which rule can fire in the current discourse state, collectively embody the kinds of domain-specific behaviour that characterise a human expert.

4 Finding the right agent

4.1 Apppointing an initial handling agent

To begin the dialogue, in order to identify the most appropriate 'handling agent', the DomainSpotter supplies each *service* agent with the output of the semantic parse that represents the user's utterance. As it attempts to find an initial handling agent, the DomainSpotter considers only service agents (like AccommodationExpert or CinemaExpert) and not support agents (like CreditCardExpert). The service agents represent the primary transaction types (booking a hotel room, enquiring about a movie, etc.) that the system handles: the system is not, for example, intended to allow the user to process their credit account, though it may elicit credit card details *in support of* a service (a hotel booking for instance). Such restrictions help the system ground its primary functions with the user. Each service agent scores the parse of the initial user utterance against the semantic categories that it can process (each agent has a range of integer values – degrees of relevance – that it will assign to different domain-specific parse tags) and returns the score to the DomainSpotter. The service agent that scores highest is the one that the DialogManager asks to apply its domain-specific heuristics to the more detailed processing of the enquiry. For example, an AccommodationExpert might score highest and so become handling agent if the user has been asking about hotels in Belfast. Specialised agents give a higher score for specialised parser tags than generic agents. For example, a user request "I'd like to go to see Finding Nemo." might parse as: *event_enquiry:[Event_type].[Movies].FINDING NEMO*. Although the EventExpert could award a score for *event_enquiry*, the CinemaExpert, as a child of EventExpert, would award a score not only for *event_enquiry*, but for *Movies* as well, and so would be the winner.

4.2 Finding out what the system can do

If the DomainSpotter is unable to identify a winning agent, it will ask the user to choose between the domains in closest contention. Indeed, if the user's enquiry is so vague as to give no domain-related information ("I'd like to make an enquiry."), the DomainSpotter will ask the user to choose from one of its highest level service agents: "Please choose between *event booking* or *accommodation booking*." – the words in italics are actually provided by the service agents. The DomainSpotter is in effect relaying to the user information that the system components know about themselves: it is part of the system's design philosophy that higher level components are largely ignorant of the precise capabilities of lower level components. Similarly, if a service agent needs to avail of a support agent in a particular area, it tells the DomainSpotter to find it an expert that handles the particular specialism (*payments*, for instance): it does not name a specific expert object. So that its area of expertise can be identified, each agent has, as one of its attributes, a vector of the specialisms it deals with. The intention is that additional lower level expertise can be added to the system in such a way that higher level behaviour (i.e. requesting the expertise) remains unchanged. Where more than one expert (e.g. CreditCardExpert and InvoiceExpert) can deal with the requested specialism (e.g. *payments*), the DomainSpotter asks the user to choose.

4.3 Transferring control between service and support

In order to maintain the enquiry focus we use an ExpertFocusStack in the DiscouseHistory. Once an agent is selected to handle the current discourse segment, it is pushed on to the top of the stack. The agent then uses its expert rules to elicit all the information needed to complete its discourse segment: an AccommodationExpert, for example, will be looking for all information needed to complete an accommodation booking. Depending on the rules it encapsulates, a *service* agent may require help from a *support* agent. For example, if an AccommodationExpert has confirmed sufficient information to proceed with a reservation, it will request help from an agent whose specialism is *payment*, and the DomainSpotter will look for one

Let us pursue this example further. The PaymentExpert is identified as an appropriate payment handler, and is placed above AccommodationExpert on the ExpertFocusStack. However, let us suppose that eliciting payment details first involves eliciting address details, and so the PaymentExpert in its turn asks the DomainSpotter to find it an agent specialising in address processing – in this case the AddressExpert. The AddressExpert now goes to the top of the ExpertFocusStack, above the PaymentExpert. Just like any other agent the AddressExpert has its own rules that allow it to accept typical combinations of information supplied (prompted or unprompted) by the user and to ask appropriate follow-up questions for whatever information is still missing. Once a support agent has all the information it needs, one of its rules will fire to 'pass control back', along with a 'finished' message, to whatever agent was below it on the ExpertFocusStack. The 'finished' agent is removed from the stack. Thus

AddressExpert will pass control back to PaymentExpert in this example, whose rules, if the user does not introduce a new topic, will continue to fire until all necessary payment information has been elicited and the payment subdialogue can be concluded – at which point control is passed back to the AccommodationExpert.

4.4 Dialogue frames and user-led focus shifts

However, a mixed initiative dialogue manager needs to be able to cope with user-initiated shifts of discourse focus. For example, a user may supply address information unprompted while the system's intention is first to elicit the information shown on the user's credit card. At present we permit transfer of dialogue control between service agents: a user may, for example, want to discuss an event booking more or less in parallel with making accommodation arrangements. In order to ground the dialogue by eliciting information in a definite context, we impose some restrictions on user-initiated shifts of focus between support dialogues, and between support and service dialogues. Dialogue frames are instrumental in implementing these policies.

Dialogue frames help identify the support dialogues associated with each service dialogue: the specification of each frame type (e.g. an AccommodationDialogueFrame) indicates the type of each of its Attributes, some of which may themselves be links to other frames (e.g. a PaymentDialogueFrame). Dialogue frames that are associated with service dialogues can be expanded into a tree-like structure by recursively traversing the various support frames that are linked to the service dialog frame. For those frames which have already been in the discourse focus (i.e. frames representing dialogue tasks that have already been the subject of user-system interaction), this is a straightforward task. Additionally the frames of possible future handling agents can be predicted and included within the tree through the use of the DomainSpotter. For example, at the outset of an accommodation enquiry, the related service dialogue frame will not generally contain an explicitly linked payment frame. However, the DomainSpotter is able to determine which agents can provide payment support, and so the system generates a number of potential discourse paths relating to payment. Key words in the user's utterances determine which path is in fact used and which payment-related frames are linked to the accommodation frame.

As the dialogue evolves, the DomainSpotter tests which agents are best placed to handle the user's last utterance: the tree of dialogue frames indicates to the DomainSpotter which support agents have been or may be involved in the current service enquiry, and should therefore be considered; the DomainSpotter will poll service agents as a matter of course. If the user's utterance is scored most highly by a support agent (relevant to the current service) whose topic has *already* been in the discourse focus, the user can return to this topic (the shift may indicate the user's intention to add to or modify information that was previously supplied). As a safeguard, the system places on the ExpertFocusStack any support agents whose rules fired on the previous path to the revisited agent, and these support agents will be allowed to test their rules again (new address information, for instance, may affect a credit card option – e.g. if the revised address is in UK, the CreditCardExpert may mention UK cardholder offers, etc.). The system uses the linked dialogue frames of topics that have already been in the discourse focus to determine the order in which such support experts should be placed on to the ExpertFocusStack

Other requests for shifts of focus from and between support agents are generally deferred ("Thanks, I'll take the address details in a moment…"), until the rules of the current support expert allow transfer. The system does not ignore the contents of the utterance that led to the deferral: the DiscourseHistory contains an UtteranceStore, a stack of the parses of the user's utterances. When it takes control of the dialogue, because one of the handling expert's rules has allowed it to, an agent first looks to the UtteranceStore to see if there is any unprocessed information that it can handle. If there is, it takes the unprocessed parsed information and begins its processing as usual with its inherited confirmation strategy and its domain-specific expert rules ("So you said the street address was 18 Malone Road. Can I have the city please?").

5 The Queen's Communicator in action

The following portion of dialogue is typical of the exchanges currently supported by the system. User utterances are shown as U1, U2, etc. System utterances (S1a,…, S2, etc.) are taken from our new natural language generation (NLG) module, which we are continuing to develop. The NLG module takes as its input the semantic constructs output from the DM. These constructs are shown thus: `MESSAGE: process…`, etc. A brief commentary, indicating which main features of the DM are being called into play, is given at key points in the dialogue and is printed in *italics*.

U1: I'd like to book a three-star hotel in Belfast from the fifteenth of August to the twentieth of August.

Key words like three-star and hotel fall within the accommodation domain, and are scored by the AccommodationExpert . Accordingly the Domain-Spotter appoints the AccommodationExpert 'handling agent'. The AccommodationExpert populates its AccommodationDialogFrame with whatever relevant information has been supplied by the user.

S1a: OK, let's start a new accommodation booking. OK, that's a three-star hotel in Belfast from 2004-08-15 until 2004-08-20. Please wait...

```
MESSAGE: process accommodation
         booking
MESSAGE: GIVEN
         AccoType [ HOTEL  ]
         AccoClass [ three-star  ]
         Location [ BELFAST  ]
         DateFrom [ 2004-08-15 ]
         DateTo [ 2004-08-20 ]
MESSAGE: database enquiry initiated
```

The AccommodationExpert uses the generic confirmation strategies that it has inherited from the DiscourseManager to implicitly confirm new values, and it uses its own domain-specific rules to initiate a database lookup that will retrieve names of hotels that match the user's criteria.

S1b: You can choose between the following: the Jurys or Lisdara. Which would you prefer?

```
PROMPT: SPECIFY
AccoName [ jurys  ]
         [ lisdara  ]
```

U2: Could you make that a four-star?

Although the system in its utterance suggests that the user should select one of the hotel names retrieved from the database, the user may provide information different to that which was requested, since this is a mixed initiative system. In this case the user changes the star-rating, instead of selecting a hotel name.

S2: Ok, got it. So that's a four-star?

```
MESSAGE: user changed value
         AccoClass [ four-star  ]
QUERY: confirm changes
```

Again, using its inherited confirmation strategy, the AccommodationExpert compares the value for the star-rating in the previous frame with the star-rating in the current frame and performs a 'repair-confirm' on the changed value.

U3: Yes, that's right.

S3a: Please wait...

```
MESSAGE: database enquiry initiated
```

S3b: You can choose between the following: the Europa, Malone Lodge, Stormont or Holiday Inn. Which would you prefer?

```
PROMPT: SPECIFY
        AccoName [  europa  ]
                 [  malone-lodge  ]
                 [  stormont  ]
                 [  holiday-inn  ]
```

U4: I'll go for a double room at the Europa.

S4: So that's the four-star Europa hotel in Belfast with a double room, from 2004-08-15 until 2004-08-20. Is that correct?

```
MESSAGE: explicit confirm values
         AccoName [ europa  ]
         AccoType [ HOTEL  ]
         AccoClass [ four-star  ]
         Location [ BELFAST  ]
         AccoRoomType [ double  ]
         DateFrom [ 2004-08-15 ]
         DateTo [ 2004-08-20 ]
QUERY: CONFIRM values
```

In the current system implementation the handling agent explicitly confirms all 'key' values needed to conclude a major segment of the discourse – once these have been supplied by the user.

U5: Make that a single room.

In this case, however, the user again changes his/her mind. The immediate next steps in the dialogue (not shown here) would be to reconfirm the 'key' values, including the newly changed value; then ask if the user wishes to check availability and reserve; and if so elicit payment details with the aid of the PaymentExpert and AddressExpert components...

6 Related work

Although some currently available dialogue systems use object components in accordance with the latest software engineering orthodoxy – (Allen et al., 2000) – little published research addresses the question of how established techniques of object-oriented software engineering (Booch, 1994) (Booch et al., 1998) can contribute to the dialogue management task.

Some research groups confirm the suitability of Java for the development of interactive, agent-based systems – for example COLLAGEN (Rich et al. 2001). Indeed, the COLLAGEN architecture, like that of the Queen's Communicator, manages discourse using a 'focus stack', a classical idea in the theory of discourse structure (Grosz and Sidner, 1986).

For dialogues that are not primarily transaction-based or frame-based, and where the system must establish the user's broader objectives before

offering advice or presenting options, a discourse management strategy based on problem-solving (PS) objects (objectives, recipes, actions and resources) is appropriate (Blaylock et al., 2003). We are currently investigating means of using PS objects to orient a dialogue, before using expertise like that currently encapsulated in our domain agents to complete those frame-filling tasks that are needed to support the user's objectives.

7 Conclusions

We have decomposed the cross-domain dialogue management task intuitively into a number of sub-dialogues, each conducted by an implemented domain specialist with its own expert rules and associated frame of information to collect. By using inheritance we easily establish a common approach to dialogue management, independent of domain: all experts inherit the same confirmation strategy. Through inheritance we ensure that domain experts have common characteristics: they all have sequences of 'expert rules' that they can apply to user-supplied information to determine what the system should do next. Domain spotting enables us to identify appropriate dialogue handling expertise for each of the user's utterances. Since our DomainSpotter actively looks for relevant expertise amongst the cohort of service and support agents, new expertise can readily be added without disturbing the system's fundamental dialogue management strategies. Additionally, division of the available experts into (front-line) service agents and (ancillary) support experts helps us maintain discourse context by deferring user-led shifts of focus that interrupt coherent data elicitation.

Future developments are likely to include: addition of new dialogue domains (e.g. travel); and incorporation of multiple dialogue strategies (using frames for mixed initiative transactions, PS objects for collaborative problem solving, and finite state transition networks for system-led interaction). Multimodal input will also be considered, including input relating to the user's emotional state, as a factor for dynamically determining an appropriate dialogue strategy for a particular discourse segment.

8 Acknowledgements

This research is supported by the EPSRC under grant number GR/R91632/01.

References

J. Allen, D. Byron, M. Dzikovska, G. Ferguson, L. Galescu and A. Stent. 2000. An Architecture for a Generic Dialogue Shell. *Natural Language Engineering* 6 (3–4), pp. 1-16, Cambridge University Press.

N. Blaylock, J. Allen and G. Ferguson. 2003. Managing communicative intentions with collaborative problem solving. *Current and New Directions in Discourse and Dialogue* (eds. J. van Kuppevelt and R. Smith), pp. 63 – 84, Kluwer, Dordrecht.

G. Booch. 1994. *Object-Oriented Analysis and Design with Applications* (2nd Edition). Benjamin/Cummings, Redwood City, CA.

G. Booch, J. Rumbaugh and I. Jacobson. 1998. *The Unified Modeling Language User Guide.* Addison Wesley Longman, Reading, MA.

B. Grosz and C. Sidner. 1986. Attention, Intentions, and the Structure of Discourse. *Computational Linguistics*, 12:3, pp. 175 – 204, Cambridge, MA.

P. Heisterkamp and S. McGlashan. 1996. Units of Dialogue Management: An Example. *Proceedings of ICSLP96*, pp. 200–203, Philadelphia.

I. O'Neill and M. McTear. 2000. Object-Oriented Modelling of Spoken Language Dialogue Systems. *Natural Language Engineering* 6 (3–4), pp. 341–362, Cambridge University Press.

I. O'Neill and M. McTear. 2002. A Pragmatic Confirmation Mechanism for an Object-Based Spoken Dialogue Manager. *Proceedings of ICSLP-2002*, Vol. 3, pp. 2045–2048. Denver, CO.

I. O'Neill, P. Hanna, X. Liu and M. McTear. 2003. The Queen's Communicator: an Object-Oriented Dialogue Manager. *Proceedings of Eurospeech 2003*, pp. 593–596, Geneva.

C. Rich, C. Sidner and N. Lesh. 2001. COLLAGEN: Applying Collaborative Discourse Theory to Human-Computer Interaction. *Artificial Intelligence Magazine*, Vol 22, Issue 4, pp. 15-25, Menlo Park, CA.

A. Rudnicky and W. Xu. 1999. An agenda-based dialog management architecture for spoken language systems. *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop*, p. I–337.

M. Turunen and J. Hakulinen. 2001. Agent-Based Adaptive Interaction and Dialogue Management Architecture for Speech Applications. *Text Speech and Dialogue–Proceedings of the Fourth International Conference TSD*, pp. 357–364.

W. Ward. 1994. Extracting information in spontaneous speech. *Proceedings of ICSLP 94*, pp. 83–86, Yokohama.