Towards Conversational Recommender Systems: A Dialogue Grammar Approach

Derek Bridge

Department of Computer Science
University College Cork
Ireland
d.bridge@cs.ucc.ie

Abstract. We highlight the importance of dialogue management in systems that support mixed-initiative interaction. For concreteness, we focus on interaction with recommender systems. We propose an approach that uses both a dialogue grammar and a recommendation strategy. The dialogue grammar makes explicit the opportunities the user and system have for *seizing* and *ceding* the initiative. But it maintains only local dialogue coherence. The recommendation strategy is used to achieve global dialogue coherence.

1 Introduction

One of our current research goals is to develop a new generation of Product Recommender Systems, ones that are more *conversational*. One aspect of a conversational system is that it will engage in *mixed-initiative interaction*. In our opinion, the following is central to mixed-initiative interaction: "At any one time, one agent might have the initiative —controlling the interaction— while the other works to assist it, contributing to the interaction as required. At other times, the roles are reversed..." [2].

Current product recommender systems generally do not engage in mixed-initiative interaction. Consider systems in which customer requirements are elicited through a 'deep interview' [14]. The system asks questions and recommends products; the user answers the questions. The system controls the interaction; the roles (who is asking, who is answering) are fixed. In systems in which customer requirements are elicited through on-screen forms, arguably, the user controls the interaction; but, either way, the roles are still fixed.

But product recommendation is a two-way information *exchange*. The system knows which products are possible/impossible (e.g., from the database schema and any additional domain constraints) and which products are available/ unavailable (e.g., from the contents of the database), and it must convey parts of this 'space' to the user. Users know (perhaps only subconsciously) their preference relations, and they try to articulate parts of these to the system. The system's contributions to the dialogue help users to focus what they will reveal of their preference structure; and the user's contributions help the system to focus what it reveals of the product space.

The motivation for mixed-initiative recommender systems is to accommodate a wider range of *conversational moves*, some of which the system will initiate, others of which the user will initiate. In the Adaptive Place Advisor [9], for example, one of the system's conversational moves ("dialogue operators") is ASK-CONSTRAIN, to ask the user for the value of a product attribute; one of the user's moves is QUERY-VALUES, to ask the system for the possible values of a product attribute. These two moves exemplify fluidity of role: the dialogue participant doing the asking is different in each. An even wider range of moves (perhaps including proposals, assertions, definitions, warnings, explanations, reminders, etc.) would give the dialogue participants more freedom to make and seek descriptions of the product space and the user's preference structure.

Although this paper is exemplified with examples of products that are 'ready-made' or 'off-the-shelf' (in our case, rental accommodation), the need for a wider range of moves becomes even more acute for 'tailor-made' or 'bespoke' products. Examples are products that are assembled through a planning and/or scheduling process, such as journey itineraries, or products that are configurable, such as personal computers. For these, the dialogue participants engage in collaborative problem-solving (planning, design, configuration, etc.).

In this paper, we present one possible framework within which research into conversational product recommender systems can be conducted. The framework makes use of a dialogue grammar, drawing ideas from Conversational Analysis, described in Section 2. To demonstrate the feasibility of the approach, we have built a simple recommender system. It is far from being the kind of conversational system that we are aiming for in our research. Numerous simplifying assumptions render it unrealistic. But, including a description of it here does make our presentation more vivid and concrete. The system is described in Section 3, and the dialogue grammar that it uses is described in Section 4. Our framework also requires a recommendation strategy. In Section 5, we explain how the recommendation strategy complements the dialogue grammar and how it relates to work on conversation policies in general and abstract task models in particular.

2 Conversational Analysis

Conversational Analysis (CA) is an approach to the study of naturally-occurring conversation with a view to discovering recurring patterns. The next four paragraphs summarize parts of the treatment given in Chapter 6 of [16].

Conversation comprises sequences of *turns*: times during which a single participant speaks. A set of rules governs the allocation of turns. In human conversation, the rules of turn-taking are quite subtle. But, in this paper, if we assume a human-computer dialogue in which there is only keyboard and mouse input and screen output, simple rules govern turn-taking: the user's turn ends when s/he presses the Submit button (or equivalent); the system's turn ends when it displays a prompt (or re-enables widgets, etc.). (This precludes concurrent ac-

tivity; see [22] for a system that still uses keyboard and mouse input and screen output but does not have this limitation.)

Of more interest to us are *adjacency-pairs*, which account for much, but not all, conversation. Adjacency-pairs are sequences of two utterances that are:

1. adjacent (unless separated by an insertion sequence —see below);
2. produced by different speakers;
3. ordered as a *first part* (which we will refer to also as the *initiative*) and a *second part* (which we will refer to also as the *response*);
4. typed, so that a particular initiative requires a certain type or range of types of response.

Prototypical examples of adjacency-pairs are question-answer, greeting-greeting, offer-acceptance/decline, etc. Where there is a range of potential responses to an initiative (as with offer-acceptance/decline), a ranking operates over the options designating one response as preferred (in the sense of normal, more usual) and others as dis-preferred (abnormal, less usual). Dis-preferred responses tend to be linguistically more complex (e.g., they may be longer).

Having produced a first part of some pair, the current speaker must stop speaking and it is expected that the next speaker will produce one of the allowable second parts of the same pair. The second part will often follow immediately (hence, adjacency). However, there frequently occur *insertion sequences*. These are sequences of turns that intervene between the first and second parts of a pair; the second part is held in abeyance during the insertion sequence. An insertion sequence will be topically related to the pair it interrupts and may be used to sort out preliminaries for providing the second part of the original pair. Insertion sequences typically contain further adjacency-pairs, which may themselves be interrupted by further insertion sequences.

If a first part is followed neither by its second part nor a turn that begins an insertion sequence (by virtue of being topically related to the first part), then the pair has been aborted. In this case, the speaker of the first part is entitled to draw inferences, e.g., that the second speaker is sulking, is not interested, is being deliberately rude, didn't understand, etc.

CA has much more to say about the organization of conversation but we have covered enough for our purposes in this paper.

CA can be criticized for being essentially *descriptive* and employing informal categories (such as question, answer, greeting, etc.). What we will do in the rest of this paper, however, is draw ideas from CA to develop something that is *prescriptive*: we will develop a *dialogue grammar* that will *specify* the set of legal dialogues our users can have with our recommender system. (For a similar use of CA but for natural language interfaces to databases, see [5].) A grammar based on turn-taking and adjacency-pairs can naturally accommodate mixed-initiative interaction by allowing either participant to contribute the first parts of different adjacency-pairs. It makes explicit the opportunities a user and a system have for *seizing* and *ceding* the initiative in the dialogue.

Although we are drawing ideas from linguistics, we wish to emphasize that we are not proposing the use of natural language processing in recommender

systems. In the system that we illustrate in the next section, the system's outputs all make use of canned text; the user's inputs are confined to a simple command language. We believe that a similar grammar could also be used to control other kinds of interaction, e.g., form-filling or direct manipulation.

3 The Sermo Conversational Recommender System

We have built a simple Prolog recommender system called Sermo. Interaction with Sermo is governed by a dialogue grammar, described in the next section.

Sermo is presently configured to help the user choose rental accommodation from a database of 794 London properties that were available for rent in Summer 2001. (This dataset is available from `www.cs.ucc.ie/~dgb/research/obr.html`).

Sermo's recommendation strategy is simple to the point of being untenable for any real recommender system. The assumptions we made allowed us to implement the recommendation strategy very quickly. The purpose of the system is to act as a vivid and concrete demonstration of the feasibility of our approach to mixed-initiative interaction. It is an indication of the promise of this line of enquiry, rather than a report of finished research.

The *recommendation strategy* is:

- Sermo asks the user to supply desired values for each of a number of attributes: the location, number of bedrooms, price per week, whether the property is furnished or not, the type of property (house or flat) and the number of bathrooms. It uses exact matching against these values when retrieving properties from the database.
- Its question-selection strategy is unintelligent: taking the attributes in the order they were listed in the previous bullet point, Sermo asks about the first unconstrained attribute in the list.
  There are two elements of variation: the user can retract a previously-specified attribute value (see the next bullet point), which makes that attribute eligible for re-selection; and if the user aborts an adjacency-pair in which the value of an attribute was being sought, Sermo takes this to be a sign that the user would rather not supply a value for the attribute (at least at this point in the dialogue), so that attribute is moved to the end of the list; its value can be asked about again at a later juncture (if it reaches the head of the list of unconstrained attributes again before the dialogue concludes).
- The user can voluntarily retract a constraint at any point. Additionally, if a point is reached where no product exactly matches all of the user-specified values, the user is required to retract a constraint.
- The dialogue continues in this fashion until at least one but no more than three properties match the user's requirements, at which point Sermo displays the properties from which the user can choose.

Figure 1 shows an annotated dialogue with Sermo that exemplifies many of the features of the dialogue grammar. User input comprises a one-word command (e.g., `constrain`, `relax`, etc.) followed by one or more parameters (e.g.,

attributes, values, etc.). It is preceded by a prompt and shown in italics. There has been a little post-editing of the system output but only so that it fits the page.

We recognise, of course, that Sermo's recommendation strategy is too simple. Our goal was to explore the use of dialogue grammars, and so we used the simplest recommendation strategy that we could. The following lists weaknesses and suggests improvements:

- With exact matching, the likelihood that no product will be retrieved is high. Even a small change to Sermo to allow it to accept minimum values, maximum values, ranges of values or sets of values would be an improvement. But, obviously, our intention for later versions of the system is to draw ideas from Case-Based Reasoning. Exact matching will be replaced or supplemented by similarity-based retrieval [25] or by our own new approach, Order-Based Retrieval [3].
- In future versions of the system, we will need to implement a dynamic question-selection strategy, in which questions are chosen at run-time in the light of the dialogue so far and on the basis of their discriminatory power, their answering cost, and so on. There is a body of work that we can draw ideas from, e.g., [6], [13], [19], [24]. Of possible relevance too are approaches to case engineering for so-called Conversational Case-Based Reasoning systems [1].
- As can be seen in Figure 1, when Sermo asks the user to retract a constraint, it simply lists all attributes on which constraints are currently in place. It would be better if Sermo were to make a recommendation at this juncture rather than leaving the decision entirely in the user's hands.
  Other ways of dealing with an over-constrained (or under-constrained) query could be supported, e.g., if a future version of Sermo allows the user to supply a range of desirable values, the range can later be loosened (or tightened). Yet further actions might apply once exact matching is abandoned or supplemented with other forms of retrieval.
- A wider range of conversational moves is possible. For defining these, we can take inspiration from the conversational moves supported by existing recommender systems. The following three alone give a wide range of ideas: [9], [12] & [20]. These include making one attribute's requirements more important than another's, specifying trade-offs, specifying tweaks, and so on.
- Future versions of the system could use different criteria, even dynamic criteria, for deciding at what points in the dialogue to propose products.
- Sermo is not user-adaptive or personalised in any way. User models can be built to encode and use the preferences of stereotypical or individual users. Relevant work includes [9] and [17]. The incorporation into the question-selection strategy of the notion of the cost to the user of answering a question is also relevant here [13], as may be recent work in Taxonomical Conversational Case-Based Reasoning systems [11], which takes account of abstraction relations between questions.

`What value for location?`	System initiates question-answer pair.
> *constrain location hampstead.*	User responds & completes the pair.
`What value for bdrms?`	System initiates another pair.
> *constrain bdrms two. constrain price 450.*	User responds and then seizes the initiative by volunteering another requirement, thus beginning a new pair.

`OK.`
 	
`What value for furnished?`
	System acknowledges the volunteered information, thus completing the user-initiated pair.
	Then the system initiates another pair.

> *possVals furnished.*	The user does not provide the second part. But the user's new initiative is on the same topic as the ongoing pair. The dialogue has entered an insertion sequence.
`Values for furnished are: yes, no` `What value for furnished?`	The system answers the user's question. Then the system reminds the user of the interrupted adjacency-pair.
> *mostPopVal furnished.*	The user still fails to complete the interrupted pair. The insertion sequence continues with another user initiative.

`Most popular value for furnished is:`
`yes`
`What value for furnished?`
	Again, the system answers.
	
	And again it reminds the user of the interrupted pair.

> *constrain furnished no.*	At last, the insertion sequence ends: the user has provided the second part of the pair.
`What value for type?`	The system initiates yet another pair.
> *constrain bthrms three.*	The user aborts the system-initiated pair by ignoring it and initiating a pair on a different topic.

`OK.`
 	
`No products satisfy your requirements.`
`Which constraint do you want to relax:`
`bthrms, furnished, price, bdrms,`
`location`
	The system completes the user's pair with an acknowledgment.
	The system initiates another pair.

> *relax bthrms.*	The user completes the pair.
`What value for bthrms?`	Another system-initiated pair.
> *constrain bthrms one.*	And a user response.

`Which item do you want:`
`316: hampstead, ...details...`
`318: hampstead, ...details...`
`319: hampstead, ...details...`
	A final system-initiated pair.

> *choose 318.*	And the user's response.

Fig. 1. A Dialogue with Sermo

4 Sermo's Dialogue Grammar

The most obvious dialogue grammar is not necessarily the most useful:

$$\begin{aligned} \text{dialogue} &\rightarrow \text{adjacency-pair} \\ \text{dialogue} &\rightarrow \text{adjacency-pair dialogue} \\ \text{adjacency-pair} &\rightarrow \text{initiative response} \\ \text{adjacency-pair} &\rightarrow \text{initiative dialogue response} \end{aligned}$$

This grammar shows dialogue to comprise one or more adjacency-pairs; an adjacency-pair comprises an initiative and a response, optionally interrupted by an insertion sequence, which is itself a sequence of one or more adjacency-pairs. (This grammar ignores a number of fine details: responses must be paired with their corresponding type of initiative; insertion sequences must be topically-related to the pair they interrupt; etc.) (The grammar used in [5] for natural language interaction with databases is similar to this one.)

This grammar gives very satisfactory parse trees: the first and second parts of a pair will be siblings, for example. But, it is not a very practical grammar for our purposes. We need to use the grammar both to *parse* user input and to *generate* system output. But, the problem with this grammar is that the individual rules that define adjacency-pairs cover both a user initiative and a system response, or vice versa. After parsing a user initiative using the first constituent in the right-hand side of these rules, the system has to switch to generating a response using the final constituent (or vice versa).

We have developed a more practical grammar (albeit one that gives less satisfactory parse trees), based on writing one rule per conversational turn:

$$\begin{aligned} \text{dialogue} &\rightarrow \text{exchange} \\ \text{dialogue} &\rightarrow \text{exchange dialogue} \\ \text{exchange} &\rightarrow \text{turn turn} \\ \text{turn} &\rightarrow \text{initiative} \\ \text{turn} &\rightarrow \text{response} \\ \text{turn} &\rightarrow \text{response initiative} \end{aligned}$$

A dialogue comprises one or more exchanges, each made up of two turns. A turn may initiate a new adjacency-pair, it may respond to an on-going pair, or it may do both.

We must refine this grammar so that it specifies constraints (such as the requirement that responses match previous initiatives). We can do this by using a unification grammar formalism. To make it easy to build our prototype recommender system, we chose to write a definite clause grammar (DCG); this formalism is directly supported by the Prolog programming language. We show the rules that define turns (using Prolog grammar rule notation) in Figure 2. The adjacency-pairs that we encoded for the Sermo demonstration dialogue in Section 3 are given in Figure 3.

In the grammar, `turn` is a 3-ary predicate. The first argument is the 'speaker': `system` or `user`. The second and third arguments are stacks of ongoing adjacency-pairs. The second argument represents the contents of the stack at the start of

```
% Rule 1
turn(system, [], [(Type, Topic)]) --> initiative(system, Type, Topic).

% Rule 2
turn(user, [(Type, Topic) | Rest], Rest) --> response(user, Type, Topic).

% Rule 3
turn(system, [(Type, Topic)], [(Type1, Topic1)]) -->
   response(system, Type, Topic), initiative(system, Type1, Topic1).

% Rule 4
turn(system, [(Type, Topic), (Type1, Topic) | Rest], [(Type1, Topic) | Rest]) -->
   response(system, Type, Topic), initiative(system, Type1, Topic).

% Rule 5
turn(user, [(Type, Topic) | _], [(Type1, Topic1)]) -->
   response(user, Type, Topic), initiative(user, Type1, Topic1).

% Rule 6
turn(user, [(_, Topic) | _], [(Type1, Topic1)]) -->
    initiative(user, Type1, Topic1), {Topic \= Topic1}.

% Rule 7
turn(user, [(Type, Topic) | Rest], [(Type1, Topic), (Type, Topic) | Rest]) -->
   initiative(user, Type1, Topic).
```

Fig. 2. Sermo's Dialogue Grammar

the turn; the third argument represents the contents of the stack at the end of the turn. Two pieces of data are stored about each ongoing adjacency-pair on the stack: the type and topic. An example of a type from Figure 3 is `qaConstraint`, which represents the kind of adjacency-pairs in which the system questions the user for a value to constrain an unconstrained attribute (e.g., how many bedrooms would you like) and the user gives an answer. The topic in this example would be the attribute in question (in this case, `bdrms`, the number of bedrooms). This is a fairly crude representation of types and topics: it is adequate to the prototype system, but could be refined in the future.

Here are paraphrases of the rules:

Rule 1: There are no ongoing pairs. The system starts a new pair.

Rule 2: There is at least one ongoing pair. The user provides a response of the same type and on the same topic, thus completing the pair.

Rule 3: There is a single ongoing pair. The system provides a response of the same type and on the same topic. Then the system initiates a new pair of a possibly different type and on a possibly different topic.

Rule 4: There are at least two ongoing pairs on the same topic. So the dialogue must have entered an insertion sequence. The system provides a response to

Name	Initiator	Initiative	Responses
qaConstraint	System	System questions user for a value to constrain an attribute	User answers with a value to constrain the attribute
pcItem	System	System proposes a list of products to the user.	User chooses one or none.
pcRelax	System	System proposes a list of constrained attributes for user to relax the constraint on one of them.	User chooses one of the attributes.
taConstraint	User	User tells the system of a constraint for an attribute.	System acknowledges the constraint.
taRelax	User	User tells the system of an attribute whose constraint is to be relaxed.	System acknowledges.
qaPossVals	User	User questions the system for all possible values of an attribute.	System answers with a list of the values.
qaMostPopVal	User	User questions the system for the most popular value for an attribute.	System answers with the value.

Fig. 3. Sermo's Adjacency-Pairs

complete the most recent pair. The system reminds the user of the ongoing pair. The grammar achieves this by requiring that the system initiate a new pair of the same type and topic as the ongoing one but it does not push it onto the stack of ongoing pairs, which remains unchanged.

Rule 5: There is at least one ongoing pair. The user provides a response to complete the pair and initiates a new pair. This aborts any other ongoing pairs so the stack contains only the new pair.

Rule 6: There is at least one ongoing pair. The user aborts it and initiates something new. We know this is not an insertion sequence because the topic is different.

Rule 7: There is at least one ongoing pair. The user begins an insertion sequence by not responding to the ongoing pair but by initiating a new pair on the same topic. Both pairs are now on the stack.

The grammar is not as general as it could be. A number of decisions have been made in writing the grammar that constrain the contributions that the *system* can make to the dialogue:

– The system cannot abort pairs: rules 5 and 6 apply only to the user. We feel that it is inappropriate for the system to ignore user initiatives. The grammar can, of course, be changed if alternative (ruder) system behaviour is warranted!

- The system's turn always includes an initiative: see the final constituent of the right-hand sides of rules 1, 3 and 4. By contrast, the user is allowed to respond to a system initiative without then initiating a new pair (rule 2). We made this decision to ensure that the dialogue never stalls. Of course, the user is free to ignore the system's latest initiative by aborting the adjacency-pair. Again, the grammar can be easily changed to achieve different dialogue behaviour, if desired.
- The system cannot initiate insertion sequences: rule 7 applies only to the user. This is our least defensible decision, and would almost certainly be changed if we were to equip Sermo with a wider range of conversational moves (i.e. if we extended Figure 3 with more adjacency-pairs). For example, if the user volunteers a constraint on an attribute, there may be occasions when the system should initiate an insertion sequence to ask the user to confirm what s/he has volunteered (e.g., if the volunteered information is over-constraining or if it contradicts information that was supplied earlier in the dialogue).

5 Task Models

There is a vast literature on dialogue much of it, implicitly or explicitly, concerned with mixed-initiative interaction. Within AI, some of the literature is concerned with dialogue between software agents (e.g., [10] and other papers from the Workshop on Specifying and Implementing Conversation Policies at the Third International Conference on Autonomous Agents); most of it is concerned with human-computer dialogue, whether in natural language or otherwise (e.g., [18] and other papers at the AAAI Workshop on Mixed-Initiative Intelligence).

This literature accepts, at least tacitly, the theory that conversational moves are actions ("speech acts" [23]) that can be formalized as planning operators and instantiated by plan generation and plan recognition processes (e.g., [4]). This theoretical perspective, if carried over directly to the design and implementation of dialogue systems, implies sophisticated reasoning engines carrying out computationally expensive dialogue-time reasoning. As a reaction against this, in both the software agents and human-computer arenas, alternative, less general but cheaper approaches to dialogue management have been proposed, e.g., [5], [7], [8], [10], [22]. These alternative approaches all, to a lesser or greater degree, attempt to 'shortcut' the reasoning that leads from intentions to utterances, and vice versa.

We will look more closely at the work reported by Elio et al. in [7] and [8], taking it to be representative of these less computationally expensive approaches to dialogue management. Elio et al. mostly present their work in the context of communication between software agents, where it constrains the generation and interpretation of messages exchanged between agents who are using an Agent Communication Language (ACL) such as KQML. But it has applicability to other forms of dialogue, including human-computer dialogue, and it is the work upon which aspects of the Adaptive Place Advisor, mentioned earlier, are based [9], [15].

The idea in this work is to define a *conversation policy* [10] which constrains the system's conversational moves and sets up strong expectations about the user's conversational moves, thus reducing the inference needed to reason between utterances and intentions. This is easier in the software agents arena than the human-computer dialogue arena: agents can be built to follow policies, but humans may not know the policy or may not wish to follow it. The challenge is to define a policy that is constraining enough to reduce inference but flexible enough to accommodate wide-ranging user behaviour.

One part of this is the specification of what Elio et al. refer to as *protocols*, which specify the semantics of conversational sub-units. These protocols correspond to what we have been defining in this paper using a dialogue grammar. We have demonstrated that this gives a way of constraining both generation and interpretation of conversational moves.

However, if we are to achieve the flexibility mentioned above, these protocols must not over-constrain the human user. They must allow for violations of expectations, such as when users shift focus, make queries on the side, etc [8]. A long-standing worry is that it is impractical, if not impossible, to predict all the courses of the dialogue and encode these in a grammar (or, equivalently, a state-transition network). But, as this paper has shown, a dialogue grammar that is inspired by CA can accommodate many of these eventualities in a quite general and therefore containable way. It does this by allowing for: multiple (including dis-preferred) responses; insertion sequences; and the option of altogether aborting an adjacency-pair. We should not be surprised at the flexibility we are afforded: after all, our grammar captures the regularities of naturally-occurring human dialogues.

These protocols are useful and the way we have defined them using dialogue grammars based on CA gives good flexibility. But Elio et al. argue convincingly that such protocols on their own are insufficient: they maintain *local coherence* within short sequences of utterances but they do not ensure *global coherence*. "Focusing on protocols alone will not fully address the primary function of a conversation policy...: to *constrain the messages that are sent*. While protocol definitions do this locally, they do not do this globally. After one protocol completes, what constrains the *next* protocol? And in what sense does the concatenation of any sequence of protocols constitute a globally-coherent conversation?" [8] (their italics).

For the extra constraints and expectations required for global coherence, Elio et al. propose *abstract task models*. Example abstract tasks are scheduling, database search, diagnosis and, in our case, product recommendation. Elio et al. assume that agents coming to a conversation know what their (abstract) task is. An abstract task model constrains the set of intentions each agent can have by specifying the objects of discourse in the domain, the actions the dialogue participants can carry out on those objects (which limits each agent's goals), how to make progress on the task and how to know when the task is finished.

Sermo has an abstract task model, but we have been referring to it up until now as its "recommendation strategy". The objects of discourse include the prod-

ucts, their attributes and the attribute values. The tasks include constraining an attribute to a certain value, retracting a constraint and describing products and attributes. The task is finished when all attributes are constrained or at least one but not more than three products match the user's requirements. Progress towards this termination criterion is achieved through the way we have engineered the preconditions of the tasks: if too many products satisfy the user's requirements but there are unconstrained attributes, another attribute must be constrained; if no products satisfy the user's requirements, a constraint must be retracted.

Elio et al. say that the task model can be formulated as a state space: there is some initial state (e.g., all attributes unconstrained), a goal state (e.g., between one and three products satisfy the constraints) and operators (such as placing and lifting a constraint on an attribute) whose preconditions specify the states in which they can be applied and whose postconditions specify the new state that results after their application. Sermo implements its task model (recommendation strategy) through semantic actions associated with the grammar rules (as per [21]), which is a little less declarative than Elio et al.'s proposal. Our Sermo system might achieve a cleaner separation too if we take a different view of the adjacency-pairs in Figure 3: at the moment pair types such as `qaConstrain` conflate an adjacency pair (question-answer, `qa`), with a task from the task model (constraining an attribute, `constrain`).

6 Conclusions

In this paper, we have highlighted the role of dialogue management in mixed-initiative systems. We have described work reported in the literature on conversation policies, which provides ways of 'short-cutting' inferences between utterances and intentions. Following work reported in [7], [8], we have proposed the use of both an abstract task model (in our case, a recommendation strategy) for global coherence and protocols of short sequences of utterances for local coherence. For the latter, we have shown the advantages of a dialogue grammar based on ideas drawn from Conversational Analysis.

For us, this is the start of a research programme directed towards building conversational product recommender systems. Sermo demonstrates the feasibility of these ideas. We have already mentioned that we must improve the recommendation strategy (both in terms of the way the retrieval is done and the way that the next action is chosen) and thereby increase the range of conversational moves supported.

The use of a conversation policy tries to strike a middle ground between systems that must carry out sophisticated inference on intentions on the one hand, and systems that support only inflexible (over-constrained) or incoherent (under-constrained) dialogue on the other hand. It remains to be seen whether these middle ground approaches can cope with applications that have a richer task structure than the one we have been considering here. Dialogues for the recommendation of 'ready-made' products have limited cause for insertion se-

quences and shifts of topic. If, however, the task were the recommendation of bespoke products, or other tasks that include configuration, design or planning, then the dialogue structure would be richer and might exceed the capabilities of simple conversation policies.

References

1. Aha, D.W. & L.A. Breslow: Refining Conversational Case Libraries, in D.B.Leake & E.Plaza (eds.), *Case-Based Reasoning Research and Development (Procs. of the Second International Conference on Case-Based Reasoning)*, Lecture Notes in Artificial Intelligence 1266, pp.267–278, Springer, 1997.
2. Allen, J.: Mixed-Initiative Interaction. *Procs. IEEE Intelligent Systems*, vol.6, pp.14–16, 1999.
3. Bridge, D. & A. Ferguson: Diverse Product Recommendations using an Expressive Language for Case Retrieval. *Procs. Sixth European Conference on Case-Based Reasoning*, 2002.
4. Cohen, P.R. C.R. Perrault, Elements of a Plan-Based Theory of Speech Acts, *Cognitive Science*, vol.3(3), pp.177–212, 1979.
5. Dahlbäck, N. & A. Jönsson: An Empirically Based Computationally Tractable Dialogue Model. *Procs. of the Fourteenth Annual Meeting of The Cognitive Science Society*, pp.785–790, 1992.
6. Doyle, M. & P. Cunningham: A Dynamic Approach to Reducing Dialog in On-Line Decision Guides, in E. Blanzieri & L. Portinale (eds.), *Advances in Case-Based Reasoning (Procs. of the Fifth European Workshop on Case-Based Reasoning)*, Lecture Notes in Artificial Intelligence 1898, pp.49–60, Springer, 2000.
7. Elio, R., A. Haddadi & A. Singh: Task Models, Intentions and Agent Conversation Policies, *Procs. of the Pacific Rim Conference on AI*, Lecture Notes in Artificial Intelligence 1886, pp.394–403, Springer, 2000.
8. Elio, R. A. Haddadi & A. Singh: Abstract Task Specifications for Conversation Policies. *Procs. of the Fourth International Conference on Autonomous Agents* 2000.
9. Göker, M. & C.A. Thompson: Personalized Conversational Case-Based Recommendation, in E. Blanzieri & L. Portinale (eds.), *Advances in Case-Based Reasoning (Procs. of the Fifth European Workshop on Case-Based Reasoning)*, Lecture Notes in Artificial Intelligence 1898, pp.99–111, Springer, 2000.
10. Greaves, M., H. Holmback & J. Bradshaw: What is a Conversation Policy?, *Procs. of the Workshop on Specifying and Implementing Conversation Policies*, Workshop Programme of the Third International Conference on Autonomous Agents, 1999.
11. Gupta, K.M.: Taxonomic Conversational Case-Based Reasoning, in D.W. Aha & I. Watson (eds.), *Case-Based Reasoning Research and Development (Procs. of the Fourth International Conference on Case-Based Reasoning)*, Lecture Notes in Artificial Intelligence 2080, pp.219–233, Springer, 2001.
12. Hammond, K.J., R. Burke & K. Schmitt: Case Based Approach to Knowledge Navigation, in D.B. Leake (ed.), *Case-Based Reasoning – Experiences, Lessons and Future Directions*, pp.125–136, MIT Press, 1996.
13. Kohlmaier, A., S. Schmitt & R. Bergmann: A Similarity-Based Approach to Attribute Selection in User-Adaptive Sales Dialogs, in D.W. Aha & I. Watson (eds.), *Case-Based Reasoning Research and Development (Procs. of the Fourth International Conference on Case-Based Reasoning)*, Lecture Notes in Artificial Intelligence 2080, pp.306–320, Springer, 2001.

14. Krantz, M.: The Web's Middleman, *Time*, vol.149(7), pp.67–68, 1997.
15. Langley, P., C. Thompson, R. Elio & A. Haddadi: An Adaptive Conversational Interface for Destination Advice, *Procs. of the Third International Workshop on Cooperative Information Agents*, pp.347–364, Springer, 1999.
16. Levinson, S.C.: *Pragmatics*. Cambridge University Press, 1983.
17. Linden, G., S. Hanks & N. Lesh: Interactive Assessment of User Preference Models: The Automated Travel Assistant, in A.Jameson, C.Paris & C.Tasso(eds.), *Procs. of the Sixth International Conference on User Modeling*, Springer, 1997.
18. McRoy, S.W. & S.S. Ali: A Practical, Declarative Theory of Dialog, *Procs. of the Workshop on Mixed-Initiative Intelligence*, Workshop Programme, AAAI, 1999.
19. McSherry, D.: Minimizing Dialog Length in Interactive Case-Based Reasoning, in B. Nebel (ed.), *Procs. of the Seventeenth International Joint Conference on Artificial Intelligence*, pp.993–998, Morgan Kaufmann, 2001.
20. McSherry, D.: Recommendation Engineering, *Procs. of the European Conference on Artificial Intelligence*, 2002.
21. Pereira, F.C.N. & S.M. Shieber: *Prolog and Natural-Language Analysis*, Center for the Study of Language and Information Lecture Notes 10, 1987.
22. Rich, C. & C. Sidner: COLLAGEN: A Collaboration Manager for Software Interface Agents, *User Modeling and User-Adapted Interaction*, vol.8, pp.315–350, 1998.
23. Searle, J.: *Speech Acts*, Cambridge University Press, 1969.
24. Shimazu, H.: ExpertClerk: Navigating Shoppers' Buying Process with the Combination of Asking and Proposing, in B. Nebel (ed.), *Procs. of the Seventeenth International Joint Conference on Artificial Intelligence*, pp.1443–1448, Morgan Kaufmann, 2001.
25. Wilke, W., M. Lenz & S. Wess: Intelligent Sales Support with CBR, in Lenz, M., B. Bartsch-Sporl, H.D. Burkhard & S. Wess (eds), *Case-Based Reasoning Technology: From Foundations to Applications*, Lecture Notes in Artificial Intelligence 1400, pp.91–113, Springer, 1998.