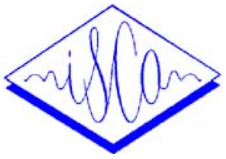
ISCA Archive
http://www.isca-speech.org/archive



INTERSPEECH 2004 - ICSLP
8th International Conference on Spoken Language Processing
ICC Jeju, Jeju Island, Korea
October 4-8, 2004

Cross Domain Dialogue Modelling: An Object-Based Approach

Ian O'Neill[1], Philip Hanna[1], Xingkun Liu[1] and Michael McTear[2]

[1]School of Computer Science, Queen's University, Belfast, N. Ireland
{i.oneill,p.hanna,xingkun.liu}@qub.ac.uk
[2]School of Computing and Mathematics, University of Ulster, Jordanstown, N. Ireland
mf.mctear@ulster.ac.uk

Abstract

Advanced spoken dialogue systems incorporate functionalities such as mixed-initiative and cross-domain dialogues. In this paper an object-based approach to cross domain dialogue modelling is described in which service agents representing primary transaction types and support agents representing tasks such as eliciting payment details are selected as appropriate by a domain spotter. The domain spotter also deals with user-led focus shifts and selects the agent that is best placed to respond to the user's utterances.

1. Introduction

Spoken dialogue systems have been typically used to help telephone callers complete a transaction in a well-defined business domain using system directed dialogues in which the system controls the dialogue flow by asking a series of questions that constrain the user's responses to a limited set of alternatives. More advanced systems permit mixed-initiative interactions that may deal with more than one domain, for example, enquiries about accommodation and events as well as supporting exchanges about payment and addresses. In these systems the dialogue manager must be able to identify the topic of the dialogue and then apply appropriate dialogue management expertise to complete the transaction. This paper describes an object-based, cross-domain, mixed initiative spoken dialogue manager (the Queen's Communicator) in which dialogue that crosses between several business domains can be modelled as an inheriting and collaborating suite of objects.

In order to enable mixed initiative interactions across domains, we model the system's behaviour as a collaboration between a cohort of 'agents'. An agent is a specialist in a particular transactional area – e.g. booking accommodation or eliciting an address – and uses its own domain-specific expert rules to elicit information. Each agent encapsulates a skillset for a substantial dialogue or subdialogue. The Jaspis architecture is also characterized by an agent-based approach, although in Jaspis agents are specialised to perform smaller dialogue tasks such as outputting a message in a certain way or using a particular form of confirmation rather than some alternative form [1].

2. System Architecture

The Queen's Communicator builds on the DARPA Communicator architecture, and specifically on the components supplied with the CU Communicator [2]. We have removed the dialogue management components from this system and replaced them with Java components of our own. As this system has been described in detail elsewhere [3], we will focus only on those elements that are relevant to the present paper.

Within the dialogue management component there is a DialogManager that contains a number of business domain experts, such as AccommodationExpert. The DialogManager also contains a DomainSpotter that helps select domain expertise and a DiscourseHistory that maintains a record of user-system interaction across domains in the form of a stack of DialogFrames, each of which in turn comprises a set of Attribute objects relevant to the particular business domain. The DiscourseManager, whose functionality is inherited by each of the domain experts, is responsible for the system's overall conversational style, determining the system's response to new, modified or negated information from the user, and also determining when domain-specific rules of thumb, encapsulated in the domain experts themselves, should be allowed to fire.

3. The DomainSpotter: Finding the right agent

3.1. Appointing an initial handling agent

At the beginning of the dialogue the DomainSpotter supplies each domain expert with the output of the semantic parse of the user's last utterance using a modified version of the Phoenix parser [4]. Phoenix analyses the user's input string and produces a sequence of semantic frames consisting of a named set of slots that represent pieces of information gathered within the dialogue. The values for these slots are determined by context-free semantic grammars that specify the strings of words that can fill each slot and that map on to the concept structures of the task. For example, given the user input:

I would like a four star hotel in Belfast from the 15th December to the 20th December (1)

the Phoenix parser will output the following semantic frames:

acco_enquiry_expert:[Acco_class].four-star
acco_enquiry_expert:[Acco_type].HOTEL
enquiry_expert:[Location].BELFAST
enquiry_expert:[When].[Date_from].[Date].[Day_Number].15 [Month_Name].[Month_Name_M].dec
[Date_End].[Date_to].[Date].[Day_Number].20 [Month_Name].[Month_Name_M].dec [Date_End] (2)

Each expert scores that parse against the semantic categories that it can process and returns the score to the DomainSpotter. The domain expert that scores highest will be the one that the DialogManager will ask to apply its domain-specific heuristics to the more detailed processing of the enquiry. As it

10.21437/Interspeech.2004-119

attempts to find an initial handling agent, the DomainSpotter considers only service agents, such as AccommodationExpert or CinemaExpert, and not support agents, such as CreditCardExpert or AddressExpert. The service agents represent the primary transaction types (booking a hotel room, enquiring about a movie, etc.) that the system handles. Each service agent scores the parse of the initial user utterance against the semantic categories that it can process and returns the score to the DomainSpotter. To do this, each agent has a range of integer values – degrees of relevance – that it assigns to different domain-specific parse tags. 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. 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 THE LION KING 3 (3)

might parse as:

event_enquiry:[Event_type].[Movies].THE LION KING 3 (4)

Although the generic 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.

3.2. Identifying the system's expertise

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 between one of its top-level service agents: e.g. "Please choose between *event booking* or *accommodation booking*." – where 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.

3.3. Moving between service and support dialogues

In order to maintain the enquiry focus we use an ExpertFocusStack in the DomainSpotter. 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 requests help from an agent whose specialism is payment, and the DomainSpotter looks 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. 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.

3.4. Dealing with user-led focus shifts

A mixed initiative dialogue manager needs to be able to cope with user-initiated shifts of discourse focus. For example, a user may supply credit card information unprompted, while the system's intention is first to elicit address information. At present we permit transfer of dialogue control between service agents but not between support agents or between support and service dialogues. For example, a user may want to discuss an event booking more or less in parallel with making accommodation arrangements. However, in order to ground the dialogue by eliciting information in a definite context, we impose some restrictions on such user-initiated shifts of focus. This supports the view that it may be desirable to hold the dialogue focus on a support dialogue, such as gathering payment details for a confirmed accommodation booking, rather than interrupt the support dialogue to start a new service enquiry, about cinema bookings, for instance. Dialogue frames are instrumental in implementing such 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 dialogue frame. For those frames that 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

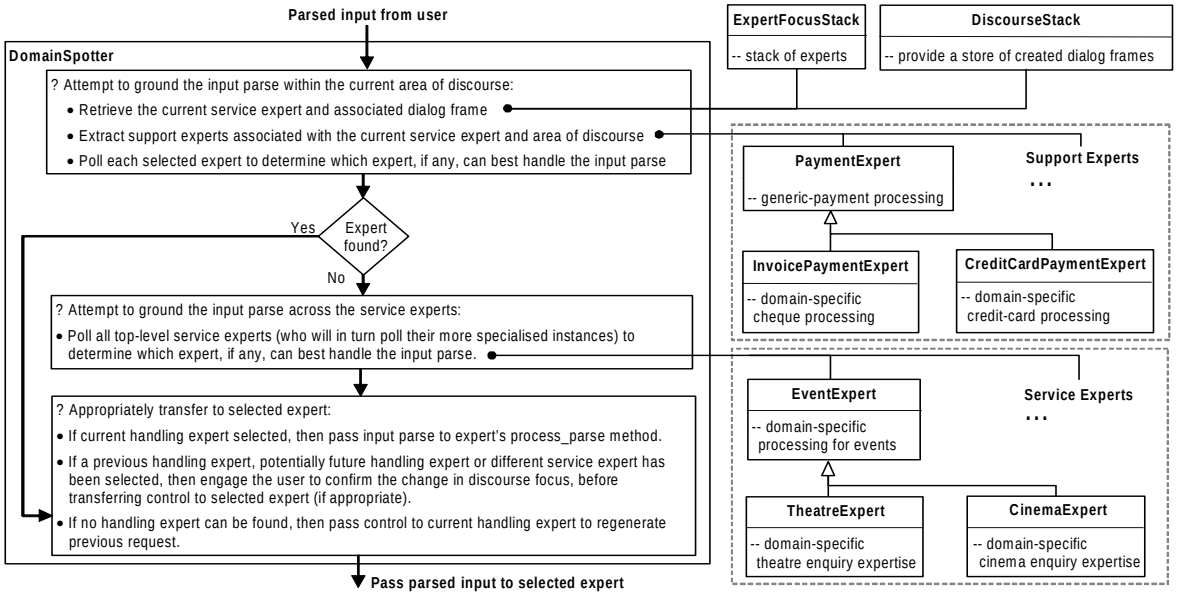


Figure 1: Domain spotting to further an on-going transaction

use of the DomainSpotter. For example, at the outset of an accommodation enquiry, the related service dialog 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 explicitly linked to the accommodation frame.

As the dialogue evolves, the DomainSpotter tests which agents are best placed to handle the user's last utterance. Figure 1 gives an overview of the DomainSpotter's decision making. The last utterance may of course relate to the domain dealt with by the current handling agent (e.g. the system may be dealing with accommodation details, and the user may, helpfully, supply information that also pertains to accommodation). However, when the user's utterance contains information that falls outside the domain of the current handling agent ('out-of-domain' information), the tree of dialogue frames associated with the current service dialogue frame indicates to the DomainSpotter which support agents have been or may be involved in the current service enquiry, and should therefore be considered as handlers for the last utterance (e.g. the user may have provided a telephone number when the AccommodationExpert is still asking about the type of accommodation required, but a telephone number may still be relevant to the broader accommodation transaction – it may be needed when the user gives personal information regarding payment).

Let us consider the permitted and prohibited shifts of discourse focus between the agents within the system. If the user's last utterance is scored most highly by a support agent which is relevant to the current service and 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 will reorder the ExpertFocusStack in these circumstances, so that any support agents whose rules fired on the previous path to the revisited agent 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.)

Other requests for shifts of focus from and to support agents are generally deferred until the rules of the current handling expert request transfer ("Thanks, I'll take the telephone details in a moment…"). 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 (e.g. because one of the handling expert's rules has requested its services), 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 ("You mentioned a telephone number. Let me just confirm the details: area code…").

If the DomainSpotter fails to locate a potential handling agent for an 'out-of-domain' utterance in the context of the current service transaction, it will poll the other service agents (does the user want to change from an accommodation enquiry to ask about cinema bookings?). Before transferring to (or indeed, before refusing a transfer to) a new handling agent – whether a service agent or a support agent – the DomainSpotter will always confirm the user's intentions ("Do you want to enquire about cinema bookings?"): the developers' intention is to avoid serious misinterpretations of users' free-form utterances, given that the system has the ability to work in a number of different though potentially related transaction domains. When deferring a request for a new service enquiry the DomainSpotter places the relevant

expert on the bottom of the ExpertFocusStack, so that it will come into the discourse focus later.

The following dialogue, based on an interaction with the system, shows how the system defers a user-initiated focus shift from a support agent to a new service agent until it has completed the transaction involving the current service agent. The developers' comments are given in italics.

S1: OK, I've got all your address details. Let's go on now with payment details. How would you like to pay?
U1: Could I just check what's on at the UGC Cinema?
The user takes the initiative by ignoring the request from the handling agent – in this case the PaymentExpert – for details of the type of payment and instead asks about cinema bookings. On the basis of key words in the user's utterance ('UGC', 'Cinema'), which result in the parse segment '[Event_venue].[Cinema].ugc', the DomainSpotter recognises that the user is attempting to change topic from a support agent to a new service agent, and at this stage defers the request. However, the DomainSpotter places the CinemaExpert at the bottom of the ExpertFocusStack for use later, and thanks to the system's UtteranceStore, the CinemaExpert, when it eventually becomes handling agent, can access information that the user has already supplied.
S2: Do you want to enquire about cinema bookings?
U2: Yes.
S3: Got it. I'll just complete the payment details first. We'll look at cinema bookings in a moment. Let's go on now with payment details. How would you like to pay?...
Once the payment details for the accommodation have been elicited and confirmed, and the accommodation booked, the system is ready to move on from the AccommodationExpert to the next agent on the ExpertFocusStack, in this case the CinemaExpert – which, like all agents, first of all checks the UtteranceStore for any relevant, previously parsed input.
S10: So that's the accommodation booking complete. Let's go on now with the cinema booking. You mentioned the UGC cinema. What date would you like to go?

4. Related Work

To our knowledge there has been no other work on cross-domain dialogue modelling that uses an object-based approach similar to the one presented here. Our focus has been less on the mechanics of the domain spotting per se and more on how the DomainSpotter assigns transactions to the relevant agents. In this respect we have not been concerned with the classification problem required by topic spotting systems which aim to identify which of a set of predefined topics are present in a document [5]. We have found the Phoenix parser to be a robust parsing system capable of returning a set of semantic frames relevant to the domains in which our system operates. One consideration is that currently semantic grammars have to be hand-crafted for each new domain. Komatani and colleagues describe a platform which automatically generates a lexicon and a language model of keyphrases based on a task description and on the structure of the domain database [6]. By spotting key-phrases using both the generated grammar and a word 2-gram model trained with dialogue corpora of similar domains, they are able to realize flexible speech understanding on a variety of utterances. This approach could help address the need for hand-crafted grammars as well as the potential problems that could arise when the system scales up to include a much larger set of domains. In another interesting approach the issue of dialogue act recognition based on the words in the user's input is accomplished using self-organising maps which allow the system to classify the user's utterances into 16 dialogue act types [7]. Such an approach could also prove beneficial when scaling up to a larger set of domains.

5. Future Developments

Future development plans include the addition of new service and support experts to cover a broader range of mixed initiative dialogues (e.g. travel enquiries). Accordingly our parser grammars will be extended to allow identification of more key words and phrases. For spoken input we intend to incorporate an off-the-shelf recognition engine into the Communicator architecture; and for output a new, phrase-based natural language generation (NLG) component will convert the DM's output concepts into well-formed natural language utterances suitable for each domain

6. Acknowledgement

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

7. References

[1] Turunen, M. and Hakulinen, J., "Jaspis^2 – An Architecture for Supporting Distributed Spoken Dialogues", *Proceedings of EuroSpeech2003*: 1913-1916, Geneva, September 2003.
[2] http://communicator.colorado.edu
[3] O'Neill, I.M., Hanna, P., Liu, X. and McTear, M.F., "The Queen's Communicator: An Object-oriented Dialogue Manager", *Proceedings of EuroSpeech2003*: 593-596. Geneva, September 2003.
[4] Ward, W., "Understanding Spontaneous Speech: the Phoenix System", *Proceedings of ICASSP'91*: 365-367, 1991.
[5] Wiener, E., Pedersen, J.O. and Weigend, A.S., "A Neural Network Approach to Topic Spotting", *Proceedings of SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval*, 1995.
[6] Komatani, K., Tanaka, K., Kashima, H. and Kawahara, T., "Domain-Independent Spoken Dialogue Platform using Key-Phrase Spotting based on Combined Language Model", *Proceedings of EuroSpeech2001*: 1319-1312, Aalborg, September 2001.
[7] Jokinen, K., Hurtig, T., Hynnä, K., Kanto, K., Kaipainen, M. and Kerminen, A., "Self-organising dialogue management," *Proceedings of the Natural Language Pacific Rim Symposium (NLPRS) Workshop: Neural Networks and Natural Language Processing,* Tokyo, Japan, 2001.