An Object-Oriented Model for the Design of Cross-Domain Dialogue Systems

Ian M. O'Neill and **Michael F. McTear**
School of Information and Software Engineering
University of Ulster
Shore Road
Newtownabbey
BT37 0QB, N.Ireland
mf.mctear@ulst.ac.uk

Abstract

Our approach to speech-based dialogue modelling aims to exploit, in the context of an object-oriented architecture, dialogue processing abilities that are common to many application domains. The coded objects that comprise the system contribute both recognition rules and processing rules (heuristics). A Domain Spotter supports the ability to move between domains and between individual skillsets. A Dialogue Model records individual concepts as they occur; notes the extent to which concepts have been confirmed; populates request templates; and fulfils a remembering and reminding role as the system attempts to gather coherent information from an imperfect speech recognition component. Our work will aim to confirm the extent to which the potential strengths of an object-oriented paradigm (system extensibility, component reuse, etc.) can be realised in a natural language dialogue system, and the extent to which a functionally rich suite of collaborating and inheriting objects can support purposeful human-computer conversations that are adaptable in structure, and wide ranging in subject matter and skillsets.

1 Introduction

The system we propose addresses two key issues that face developers of speech-based natural language dialogue systems. Firstly, how can developers exploit the commonality that exists between different application domains - to make the development task easier on the one hand, and on the other hand to make systems as computationally efficient and as functionally wide-ranging as possible? Secondly, given the current inaccuracies of speech recognition, how can developers implement domain independent strategies for limiting the damage caused by misrecognition, while at the same time maintaining an apparently natural conversational flow between system and user? An object-oriented development paradigm offers valuable insights into how these challenges might be addressed. In this respect the current approach builds on previous work involving an object-oriented approach to dialogue management (Sparks, Meiskey & Brunner, 1994), in which the main system components might be regarded as forming a developer's toolkit. We envisage system components that draw on the strength of an object-oriented architecture. Inheritance and association relationships will be used to ensure that generic functionality which can be shared by more specialised system components need be defined only once and can be introduced into the dialogue flow, in real time, as and when required.

Based on the notion of an evolving, multi-layered dialogue model (McGlashan, 1996), our system design includes a number of dialogue model classes (collectively the Dialogue Model) whose role it is to record each concept (a booking request, for example) as it is identified; to monitor and guide the process by which concept's attributes (destination, departure time, etc.) are confirmed or assumed; and to populate a request template that will ultimately be used in database accesses.

Central to our project is a notion of discrete, reusable system components, some of which are intended to work collaboratively in software mechanisms, some to provide generic functionality that can be tailored or augmented to suit particular applications. Identifying and exploiting areas of commonality and specialisation between different processing

domains promises rich rewards. We have been inspired to some extent by the premise that everyday, person-to-person dialogues (whether it is a booking clerk at a theatre responding to a customer's enquiries, or a teacher helping a pupil with a mathematics problem) are in some sense 'scripted'. Previous experience of a situation, or explicit tutoring in a particular task, means that real-life dialogues often consist of elements that have been rehearsed, and are therefore predictable. However, as in natural human speech, the system must recognise and accommodate spontaneous shifts from one script to another, and be able to cope with changes in the detailed content and structure of a script in different circumstances.

To make three broad distinctions, one may view these 'set pieces' as occurring at a meta-level, a domain level and a skill level - and these levels are reflected in the system architecture we are evolving. At a meta-level, for example, people tend to recognise cues for taking, relinquishing or continuing a dialogue turn; at a domain level, someone wanting to reserve a ticket for a show broadly knows the sorts of questions they can ask at the theatre booking office and the sorts of answer they are likely to receive; at a skill level, people generally know how to do conversions between dates on the one hand and days of the week or duration on the other. We have endeavoured to identify some of these set pieces at their different dialogue levels, with a view to creating classes that encapsulate the meta-dialogue behaviour that is common to the great majority of interactions (and which is represented in our generic Dialogue Intention class), the business domain expertise that in human terms distinguishes professionals in one field from those in another (our Business Expert classes), and the individual skills like handling dates and numbers, that are used in many different business domains (our Skill Expert classes). In general terms, adherence to best practice in object-oriented development offers the prospect of systems that can be readily extended and customised, in building block fashion. More significantly, though, it is our intention to use our suite of classes in implementations that support highly complex interactions with the user: a single dialogue may range over several business domains, each of which may use several distinct skill sets. The system has the intelligence to decide, in real time, which business expertise and which skillsets are required to pursue the user's enquiries, and calls upon the services of the appropriate coded objects.

To give a flavour of our system's architecture, we include outline descriptions of some of its most important classes: Dialogue Manager; Dialogue Intention; Find Enquiry Type; and Domain Expert. The preliminary class relationship model (see Figure 1) further sets these classes in context - the model uses a simplified version of the Booch notation (Booch, 1994).

2 Dialogue Manager

- The Dialogue Manager is responsible for the overall control of interaction between the system and the user, and between the main system subcomponents - which in broad terms include Coms facilities, Generate Speech facilities, the enquiry processing objects, and the system Database.
- The Dialogue Manager is responsible for selecting the current Dialogue Intention, of which there are several subclasses. By default the Dialogue Manager pursues a sequence of dialogue intentions that is typical of the majority of dialogue domains: the system greets the user; determines the nature of the user's enquiry; gathers the data necessary for the successful answering of the enquiry; handles any (database) transactions associated with the enquiry; checks if the user has any further enquiries; and concludes the dialogue.
- It uses system resources to identify and respond appropriately to user interruptions.

3 Dialogue Intention

- Dialogue Intention embodies generic functionality for the furtherance of a dialogue.
- Dialogue Flow. The Dialogue Intention class encapsulates a variety of approaches to phrasing, rephrasing and personalising system utterances, with the aim of handling (in as natural a manner as possible) communication errors and processing delays.
- Use of Expertise/Skills. Dialogue Intentions may themselves encapsulate heuristics that allow them to instantiate a Dialogue Model (and by extension the associated Dialogue Objects, Discourse States and Request Templates) for relatively high-level processing tasks (Greet, Find Enquiry Type, for example). However, most Dialogue Intentions make use of the Skill and Domain Expert classes, whose heuristics permit rather more specialised enquiries involving either generic but complex skillsets (working with colours or gathering address information, for example) or specialised application domains

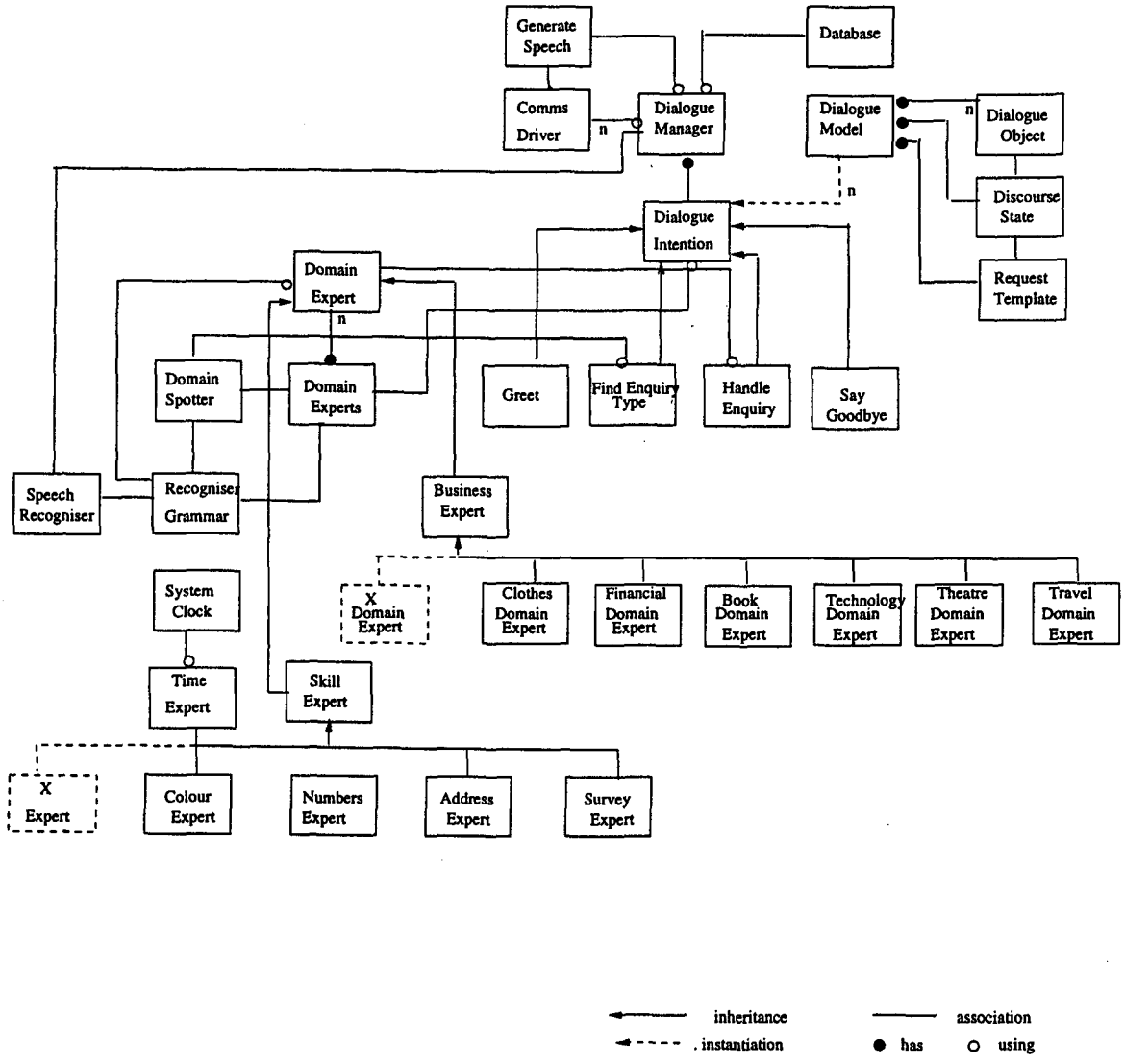

Figure 1: Class Relationship Model

(organising travel itineraries, or booking theatre tickets, for example). Again these skills and expertise subclasses provide the Dialogue Intention subclass with the necessary heuristics to instantiate a Dialogue Model.

4 Find Enquiry Type

- The Find Enquiry Type class (a subclass of Dialogue Intention) allows the Dialogue Manager, both to prompt the user into specifying the nature of his/her inquiry, and to interpret the nature of a user's utterance when it receives an indication that the user has spoken unprompted.

- The Find Enquiry Type class uses a Domain Spotter class to identify the Domain Expert that is best suited to handling the enquiry. An appropriate Domain Expert is confirmed through the elaboration of an appropriate Dialogue Model. The Dialogue Manager supplies the Handle Enquiry dialogue intention with details of the selected Domain Expert.

5 Domain Expert

- Each Domain Expert class, regardless of the specific domain its subclass addresses, typically provides the following functionality:

  1. Request template structure for the domain;
  2. Enquiry processing algorithms for the domain (typically IF...THEN...ELSE constructs), including recommended use of any Skills Expert, for specialised but non-domain-specific processing (e.g. handling colours, times, etc.)
  3. Word combinations (bigrams or trigrams) from the domain to extend the generic capabilities of the Recogniser Grammar.

- The Domain Expert is used to instantiate and evolve a related Dialogue Model.

6 Dialogue Model: Dialogue Object, Discourse State, Request Template

- The Dialogue Model class is a containment class encompassing Dialogue Objects (semantic interpretations of user utterances in the light of specialist knowledge brought to bear by the appropriate Domain Expert); the Discourse State (which records the current status - confirmed, assumed, etc. - of the parameters that apply to the Dialogue Objects) and the Request Template (which when fully populated is used by the Handle Transaction class - a database driver - to make a database access).

- The Dialogue Model evolves in a manner similar to that outlined by (McGlashan, 1996). Confirmation strategies are tailored to the particular operating environment and the specialised domain. They are recorded in the Dialogue Intention class, or in the relevant Domain Expert subclass.

7 Concluding remarks

A key aim of our work will be to ascertain if our suite of objects (which in combination encompass dialogue skills from the generic to the highly specialised) can be built into co-operative mechanisms in real time to simulate realistically the richness, robustness and adaptability of natural human dialogue. If this does indeed prove to be the case, our dialogue model will have attained its core communicative goal: more than this, its object-oriented architecture will facilitate the work of the software engineer by providing a set of discrete components that can be easily reused, modified or extended in new dialogue systems.

References

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

S. McGlashan. 1996. Towards Multimodal Dialogue Management. In S. Luperfoy, A. Nijholt, and G. Veldhuijzen van Zanten, editors, *Dialogue Management in Natural Language Systems, Proceedings of the Twente Workshop on Language Technology 11*, Enschede: Universiteit Twente.

R. Sparks, L. Meiskey, and H. Brunner. 1994. An Object-Oriented Approach to Dialogue Management in Spoken Language Systems. In *Human Factors in Computing Systems - CHI '94*, New York: ACM, 211-217.