Developing Extensible and Reusable Spoken Dialogue Components: An Examination of the Queen’s Communicator

Philip Hanna¹, Ian O’Neill¹, Xingkun Liu¹, Michael McTear²

¹School of Computer Science
Queen’s University of Belfast, N.Ireland
{p.hanna, i.oneill}@qub.ac.uk
Xingkun.Liu@ed.ac.uk

²School of Computing and Mathematics
University of Ulster, N.Ireland
mf.mctear@ulster.ac.uk

Abstract

This paper presents an examination of the Queen’s Communicator, a cross-domain, mixed initiative spoken dialogue system, in terms of the system’s extensibility and reusability. The paper explores how an object-oriented design methodology can be used to define both high-level discourse objects and appropriate forms of relationship between discourse objects. The paper concludes that the application of an object-based approach to dialogue system design makes possible the use of a number of different mechanisms that facilitate the extension and re-use of discourse expertise.

1. Introduction

It is widely acknowledged that spoken language dialogue systems offer a natural and capable interface for many different forms of human-computer interaction; from selecting pizza toppings to battlefield training. However, given the high development costs associated with constructing a modestly complex spoken language dialogue system, there is often a strong desire to be able to adapt or reuse existing dialogue components when building a new dialogue system, thereby offering the potential of significantly reduced development times and associated costs.

Whilst the domain-independence hypothesis [1] is broadly accepted, i.e. that the complexity associated with language interpretation and dialogue management can be largely considered independently of the task being performed, the question of how a domain-independent system should be developed continues to pose a significant challenge.

Part of this challenge arises as dialogue tasks, and hence the architectures developed to solve such tasks, differ in terms of their complexity, ranging from those concerned with presenting discrete options to the user, across those intended to elicit a certain predetermined frame of information, to those intended to solve complex forms of problem through negotiation. Within this framework, a number of different domain-independent dialogue systems have been developed, e.g. [1-3], ranging from complex systems designed to explore sophisticated forms of problem solving to commercial systems that offer high levels of robustness for simple forms of discourse [4].

As the Queen’s Communicator draws upon the DARPA Communicator infrastructure and makes use of a number of ‘off-the-shelf’ components to support recognition, parsing and synthesis, the examination within this paper will solely be concerned with the discourse management component. The paper is organised as follows: firstly high-level discourse objects are examined, before then moving on to the formation of a discourse project and finally an examination of the Queen’s Communicator infrastructure and makes use of a number of discourse-related expertise.

1.1. Examination of the Queen’s Communicator

This paper reports the findings of an examination of the Queen’s Communicator with regard to the system’s design goals. The examination does not consider low-level implementation specifics; instead it focuses on the high-level architectural design in terms of adding and extending and modifying discourse-related expertise.

More precise details concerning the implementation and features of the Queen’s Communicator can be found in [6].

2. Encapsulating expertise within objects

Figure 1 outlines the various high-level discourse objects that are defined within the Queen’s Communicator.

![Figure 1: High-level discourse objects defined within the Queen’s Communicator](image_url)
2.1. ‘Expert’ objects

Conceptually, each ‘expert’ object can be considered as an instantiation of a particular form of agent, i.e. an autonomous entity which can serve a human interlocutor through spoken dialogue towards completion of some shared task. A number of different types of ‘expert’ object are defined, with each specific type providing a measure of expertise within a particular domain. The ‘expert’ objects (hereafter simply referred to as ‘agents’) are registered with the dialogue manager, thereby making available their expertise to the wider system.

Within the Queen’s Communicator, agents have been separated into two categories: service agents and support agents. Service agents are defined as providing front-line services to the user, e.g. typically complete transactions such as booking cinema tickets, etc. In contrast, support agents are intended to provide supporting services, e.g. eliciting a segment of information that is needed within a wider enquiry. In broad terms, support agents can be viewed as providing set-piece dialogue assemblies that are of use within a number of different enquiries/transactions, and as such, represent portable, reusable elements of discourse functionality.

Within a typical dialogue, the user may wish to complete a number of specific enquiries/transactions, e.g. booking cinema tickets and a table at a nearby restaurant, with each type of enquiry/transaction under the control of a relevant service agent which will draw upon supporting agents as necessary. In turn, a support agent may also contract another support agent in order to fulfil its objectives.

2.2. Managing interactions between agents

In order to reduce the degree of dependency between agents it was decided that agents should act autonomously and not require any specific, hard-coded, knowledge concerning other agents. In order to achieve this, all discourse requests are structured in terms of a description of the desired expertise, not in terms of registered agents. Also, each agent is responsible for ascertaining and indicating if it can adequately satisfy a request for a certain form of expertise.

Within this structure, it is the role of the DomainSpotter to appropriately manage agent interactions and requests concerning expertise, be they user or agent initiated. For example, the TheatreExpert may send a request for ‘payment commencement’ to the DomainSpotter, potentially augmented with relevant user-provided information, e.g. a preference for credit card payment. The DomainSpotter will then poll all registered agents using the received request. Each agent will consider the request and return a ‘score’ indicating the agent’s ability to successfully handle the request. The DomainSpotter will then select the most appropriate agent to handle the request and arrange for that agent to be incorporated within the dialogue.

By expressing agent selection requests in terms of an expression of desired expertise and providing an ‘agent management object’, i.e. the DomainSpotter, both system maintainability and extensibility are improved as the operation of the system is not dependent upon the agents that are registered, but is instead dependent upon the expertise that registered agents make available.

As can be seen from Figure 1, agents may also be grouped into hierarchies of increasing degrees of specialism. This form of relationship was introduced as it provides a means of defining agents that can deal with general forms of domain enquiry/transaction (e.g. the user wishes to pay for their cinema tickets) and which can evolve the enquiry/transaction into a more specialised form if and when necessary (e.g. the user wishes to pay using a credit-card). As such, this form of relationship differs from the notion of ‘contracting’ a separate agent to carry out a portion of work on behalf of a requesting agent, in that it represents an ‘evolution’ of an enquiry/transaction from a more general form into a more specific form, thereby providing a mechanism to adapt to task refinement within the dialogue.

3. Developing a dialogue product

Figure 2 shows the objects which are used to encapsulate discourse information.

![Diagram showing objects used to encapsulate discourse information](image)

As the Queen’s Communicator is built upon a slot-filling approach, ‘Frame’ objects form the core building blocks of the discourse product. Each frame groups together a number of related ‘Attributes’ objects (i.e. slots). In turn, each attribute is defined in terms of a number of different fields [6], including a discourse status, system intention, pegged confidence, attribute type value(s), etc.

As shown in Figure 2, frames are defined within an inheritance hierarchy, whereby more generic frames are extended by more specialised frames. For example, the Event frame contains attributes (slots), common to all event related enquires, which are inherited by all specialised forms of event frame. Encompassing frames within an inheritance hierarchy is beneficial as it provides a contextual framework within which more specialised frames can be readily defined. It is potentially limited, however, in that extending frames are conceptually viewed as ‘using’ all inherited Attributes - this may not always be the case.

In order to facilitate complex forms of enquiry/transaction, frames within the Queen’s Communicator can be linked together to form larger, composite, discourse objects.
For example, a theatre enquiry may have progressed onto a booking using a credit card and require the collection of the user’s address and telephone number; a setup illustrated in Figure 3. Frame linkage is achieved through the use of LinkedFrame attributes which provide a linking mechanism whereby a particular frame can indicate that additional frames of information may need to be collected.

![Diagram of theatre booking process](image)

**Figure 3:** Example dialogue product involving payment elicitations in support of a theatre booking

In order to adapt to new or revised agent expertise, the linkage between frames is not predetermined. Instead, each LinkedFrame attribute will contain a description of requested expertise. This makes possible dynamic linking using the same polling process as that used by agents who wish to contract some other agent. By linking dynamically, it also becomes possible to incorporate information imparted by the user alongside that contained within the LinkedFrame attribute to select the most appropriate linked frame.

Expressing the dialogue product of an enquiry/transaction as a hierarchy of linked frames is useful from a number of different perspectives. Firstly, it introduces a mechanism whereby frame sizes are unlikely to grow to an unwieldy size (in terms of the number of slots) by breaking up a complex, typically multi-slotted, enquiry/transaction into a number of smaller chunks. Secondly, splitting up the enquiry/transaction into a number of smaller chunks provides a structure that can facilitate both the delegation and delimitation of agent responsibility (agents will typically be tasked with the collection of one or more frames of information). Thirdly, it provides an open-ended structure that can dynamically grow and adapt depending upon the objectives set by the user.

However, making use of a hierarchy of linked frames is not without its challenges. In particular, the division of an enquiry/transaction into a number of linked frames should, in most instances, be imperceptible to the user. This entails that users, if permitted, should be able to revisit previously ‘completed’ frames or offer information that will be of use within ‘future’, i.e. currently unlinked, frames. Details of how this is accomplished within the Queen’s Communicator are reported elsewhere [6].

Additionally, assigning responsibility for the completion of a particular frame to an individual agent may also be seen as limited in that it separates the discourse into a number of discrete segments and thereby limits the opportunities for agents to cooperate with one another. Whilst this criticism is justified, it is likely to be the case that the types of application that require agents to engage in collaboration are unlikely to lend themselves to a predominantly slot-filling approach.

### 3.1. User and system attributes

Attributes within a frame can be classified in terms of type as being either user or system attributes. User attributes are intended to represent ‘slots’ that are populated by information supplied by the user. In contrast, system attributes are intended for internal use and cannot be populated by the user, thereby providing a means of representing information that has currency within a particular enquiry/transaction but is not directly provided by the user. For example, system attributes might contain mundane items of information such as Boolean flags for recording progress or database confirmation of values, etc. and could also include information inferred from the user’s previous utterances that can be used to control the order in which subsequent slots are filled.

In effect, all data associated with a particular enquiry/transaction is encapsulated within a corresponding frame object. Because of this, it becomes possible to cleanly separate enquiry/transaction data from the expertise that deals with the collection and manipulation of such data. This is advantageous in that it permits one agent to process a number of different instances of a particular enquiry/transaction type at the same time, e.g. as might arise when dealing with payment across a number of different transactions. However, it is acknowledged that this form of encapsulation is potentially limited as it does not provide a straightforward means of representing session-wide information that may pertain to a number of different enquiries/transactions, e.g. hotel booking dates may constrain possible theatre bookings.

### 4. Internal agent structure

Whilst the design of agents as autonomous entities that can individually engage the user in dialogue results in a simplified architectural model of the system, it entails that agents must independently deal with the issues surrounding spoken dialogue. For example, in addition to undertaking discourse furtherance actions towards completion of a shared goal, it is also the responsibility of each agent to ensure that the discourse is appropriately grounded and that user or system instigated repairs/negations are appropriately executed, etc.

As many of these elements of discourse behaviour will be common across all agents, e.g. the mechanism whereby information is confirmed or repairs acknowledged is likely to be common across different enquiries/transactions, it becomes possible to define a common core of discourse behaviour that can be augmented with domain specific expertise. This is achieved using the inheritance structure shown in Figure 4.

![Inheritance structure within agents](image)

**Figure 4:** Inheritance structure within agents
4.1. Encapsulation of generic discourse behaviour

The DiscourseManager has responsibility for the overall management of turn-based interactions between the user and agent. The management encompasses the determination of when it is appropriate to invoke high-level, domain-specific expertise to further the overall dialogue, alongside common forms of discourse behaviour such as grounding, repairing and arbitrating between different dialogue intentions. The DiscourseManager will only seek to invoke domain-specific expertise if it is satisfied that all previous system intentions have been acted upon, with appropriate grounding where necessary.

4.2. Introduction of expert rules

Whilst the DiscourseManager embodies functionality concerning the broad management of turn-based interaction between the agent and the user, it is the EnquiryExpert that introduces the mechanism through which domain specific expertise may be encapsulated and used to further the discourse towards some shared objective.

This is achieved through the introduction of an engine that supports the execution of expert rules. An individual expert rule is defined by a number of preconditions and a number of actions. Each precondition refers to some property of a given attribute, for example the attribute's value or peg, or the system intention or status associated with that attribute. In turn, the actions associated with a particular rule relate to discourse furtherance. Whilst this can include changes to the properties of an attribute, e.g. the system intention, it also encompasses the initiation of database queries, NLG utterances, discourse flow changes, inter-agent collaboration, etc. In broad terms, the preconditions of a rule describe a particular situation, e.g. the user has provided a cinema name and a particular time, whilst the actions describe what should be done to appropriately further the dialogue, e.g. a database enquiry should be initiated to establish which movies are showing at the stated time. Depending on the number and nature of the preconditions, both very general and very specific situations, alongside the actions that are appropriate in such situations, can be readily described.

4.3. Encapsulating domain expertise

Typically agents introduced at this level need only define a collection of expert rules embodying their expertise. In order to facilitate this process, collections of related expertise can be packaged into ExpertRuleSequence objects which are then added to one of three different groupings: discourse-furtherance rules, which are concerned with advancing the dialogue; database-furtherance rules, which are concerned with recovering from unsuccessful database enquiries, e.g. by relaxing certain search constraints; housekeeping-rules, which are not concerned with discourse furtherance, but rather internal actions, e.g. frame maintenance, flag changes, etc.

When determining which rule should fire, an agent will firstly test its own expert rule sequences before considering inherited expert rule sequences. This structure provides a means of applying specialised expertise, before backing off onto more general forms of expertise if necessary. The structure also supports maintainability as a more specialised agent need only introduce rules which define new expertise or refine inherited expertise.

From a design perspective, expert rules were used as they were felt to offer a designer friendly means of expressing expertise in a manner that was both intuitive and readily extensible. The decision also recognises that rule-based programming offers a means of capturing and expressing expertise associated with enquiry and transactional forms of discourse. However, it is acknowledged that a rule-based approach does not offer an adequate means of dealing with all forms of complex reasoning.

5. Conclusions

A central goal of the Queen’s Communicator project was to explore the extent to which an object-oriented methodology could be used to improve the maintainability, extensibility and reusability of a cross-domain, mixed initiative spoken dialogue system for enquiry and transactional forms of discourse. Based on the examination outlined within this paper, it is our opinion that the Queen’s Communicator provides a number of mechanisms which contribute towards this goal.

The encapsulation of domain-expertise, including set-piece dialogues, into families of agents provides a mechanism for the reuse and extension of expertise, although the service/support distinction may benefit from refinement in order to support distinct ‘libraries’ of expertise. In addition, the use of statements of desired expertise when selecting and contracting agents provides a layer of abstraction that aids maintainability. The notion of extensibility also applies to the dialogue product, which, whilst based on slot filling and therefore not suited to certain forms of application, offers a dynamic, domain-independent structure that can adapt and grow as the dialogue progresses. In terms of the internal structure of each agent, the encapsulation and inheritance of common discourse management functionality, coupled with the use of inheritable expert rule sequences, provides a mechanism that enables domain expertise to be readily expressed within a structure that facilitates the extension and refinement of encapsulated expertise.

6. References


