Use of the QuADS Architecture for Multimodal Output Generation


Published in:
MOG 2010
Use of the QuADS Architecture for Multimodal Output Generation

Ian O'Neill, Philip Hanna, Darryl Stewart, Xiwu Gu,
School of Electronics, Electrical Engineering and Computer Science, College of Computer Science and Technology,
Queen’s University Belfast, Huazhong University of Science
Belfast, BT7 1NN, Northern Ireland. and Technology,
{ioneill, p.hanna, dw.stewart}@qub.ac.uk Wuhan 430074, China.
guxw_wang@sina.com

Abstract

QuADS (Queen’s Advanced Dialogue System) is a suite of highly generic and customizable Java classes for the development of spoken and multimodal dialogue systems. Some of the classes in QuADS represent communicative acts, such as are found in information-providing or transaction-based dialogues: in classifying these acts, whether generated by the system or understood by the system from the user’s input, QuADS uses a dialogue act hierarchy based on the DIT++ taxonomy. Other classes are concerned with the underlying task of finding the information or service that the user requires, or, when the user’s specific information request cannot be satisfied, of presenting reasonable alternatives. In this paper, as well as giving an overview of the QuADS architecture, we examine the means by which a system developed with QuADS selects the modalities that it will use to present information to the user, taking into account the availability of a particular modality in a given system configuration, and considering also the user’s preference for particular modalities. Although we apply some obvious measures to avoid ‘information overload’, at present we are concerned not so much with the ‘optimal’ modality or combination of modalities for a particular task as with the mechanisms within a generic, domain-independent framework that make selection of modalities possible in accordance with system capabilities and user preferences.

Keywords: Multimodal Dialogue, Object-Oriented Development.

1 INTRODUCTION

ISIS is an EPSRC-sponsored Integrated Sensor Information System that analyses and stores video information gathered from public transport vehicles. Sponsored by the UK’s Engineering and Physical Sciences Research Council (EPSRC, Project No. EP/E028640/1), its purpose is to detect situations that are potentially threatening to people or property. ISIS-NL is the multimodal information retrieval component of ISIS and is based on natural language dialogue. It uses a suite of re-usable dialogue components that we have developed at Queen’s University Belfast: collectively these components are known as QuADS, Queen’s Advanced Dialogue System (Hanna, 2008). QuADS represents a new generation of object-based natural language dialogue technology at Queen’s, succeeding our Queen’s Communicator architecture (O’Neill et al., 2003). Developed in Java, the QuADS toolkit adheres to the same object-oriented precepts as its predecessor, inspired by the approaches to OO development set out since the 1990’s by Grady Booch and others (Booch, 2007).

ISIS-NL is intended to demonstrate how busy staff in a network operations centre can use spoken (or in some cases keyed) instructions to retrieve video footage relating to incidents of interest – for example, ‘Can you tell me if a man got on a number 45 bus at Blackheath between seven-thirty p.m. and nine-thirty p.m.?’. To respond to the user’s request, the system potentially has available a combination of text and speech, as well as the still images and video that represent the retrieved information itself. The manner in which it presents its information and the amount of information it presents depend, amongst other things, on the number of database ‘hits’ or ‘alternative suggestions’ that the system has to convey, as well as on the user’s preferences for particular modalities. The system is also limited to the output modalities that are actually available in a particular system configuration.
In our latest implementation of ISIS-NL, other than introducing commonsense limitations to the amount of information the system attempts to convey in a particular modality (for example, the number of alternatives it tries to tell the user about though speech alone), we have not attempted rigorously to optimise the system’s use of modalities. We have not, for instance, attempted to ensure that presentation of information is optimised to a particular individual’s ability to assimilate it. However, we recognise that for some years the interplay between the modalities as channels of communication, and the effect of different modalities on the individual user in particular situations, has been a lively area of research. How and when, for example, might one modality be used to reference material that is being used in another modality? Almost two decades ago Maybury pointed out the challenges that these issues would pose for developers of multimodal interfaces (Maybury, 1992). Since then he and others have gone on to explore how information that is available in different modalities might be selected and presented in accordance with the user’s preferences (Bernsen & Dybkjær, 1999; Light & Maybury, 2002; Oviatt et al. 2004). Ideally the computer-based system will replicate in its use of modalities the most natural behaviours of human conversation partners: it should be able to decide when to use speech alone, and when the communicative task best served by speech in combination with a visual indicator that is equivalent to a human conversation-partner pointing (Van der Sluis et al., 2008).

We foresee that, in many QuADS-based applications, the user himself or herself might simply decide (via a customisation GUI or even a spoken customisation sub-dialogue) which combination of outputs, which customisable configuration, is generally most effective for him or her in a typical working environment. However, our dialogue architecture is designed deliberately to provide a very high-level framework that can accommodate quite specialised behaviour at any stage of the dialogue-handling process. In Section 2.3.1 we discuss the manner in which ‘forums’
are used to facilitate communication between specialised processing objects, which we refer to as Agents, each of which can make decisions about, and advise a co-ordinating Manager about, its own specialised area of dialogue-handling: some Agents might specialise in processing the transaction underlying the dialogue, others might deal with database queries and help reformulate failed information requests, and so on. While the prototype dialogue system currently implemented in QuADS has static, and rather simple hand-crafted rules to control slot-based, transaction-handling behaviour, our architecture is flexible enough to accommodate more active and subtle dialogue strategies, including those that would allow a system to learn optimal multimodal strategies from interaction with the user (Rieser & Lemon, 2010). Dialogue-handling expertise, whatever its theoretical motivation or particular implementation, can be incorporated into the system in the form of Agents that advise appropriate dialogue- and task-handling Managers. In this respect the QuADS architecture has been ‘future-proofed’ to meet eventual research demands at Queen’s, and accommodate the aspiration to adaptive, naturalistic, multimodal dialogue that is being expressed by researchers more generally (Geertzen et al., 2004).

For demonstration purposes we have included with the latest version of ISIS-NL a number of static user- and system profiles. For each user-profile, the system, according to its capabilities, exhibits different behaviours as it attempts to satisfy the user’s inquiry: these variations affect the number of choices that the system offers the user, and the modality in which it announces these choices (e.g. “I couldn’t find an exact match for your request. Here is the first option I have to suggest. Can you tell me if you want...?” etc.). We will examine the policy for composing these system turns and choosing their modality in Section 2.3.4. Whenever a match that satisfies or substantially satisfies the user’s information request is possible, the system presents video segments for the user to examine. Figure 1 gives a user’s view of the system in action as it reaches this stage: the system displays a selection of ‘thumbnail’ still images that represent the best matches for the inquiry; each time the user selects a thumbnail, he or she then uses the large window to review the corresponding video footage (‘Play video one.’ ‘Pause video.’ ‘Go forward five seconds.’).

Figure 2. A representative selection of acts from the Dialogue Act hierarchy in QuADS.
2. THE QUADS ARCHITECTURE

2.1 THE BACKGROUND TO QUADS

The QuADS architecture draws on Dynamic Interpretation Theory, and in particular the DIT++ taxonomy of dialogue acts, proposed by Bunt (2000, 2007, 2009a). Dynamic Interpretation Theory is motivated by the insight that dialogue is rarely conducted for its own sake: rather, there is often an underlying goal (a need to accomplish something in the real world – like booking a ticket or finding a person). According to DIT, Dialogue acts are instrumental in achieving such goals: alongside dialogue acts whose role is to maintain the smooth interaction between the dialogue partners (giving feedback, managing turn-taking, dealing with matters of etiquette, etc.), there are also the core, task-related, information transfer acts that are used either to seek or to provide information, as well as the acts that are used to make offers and promises and to give instructions.

QuADS comprises an extensive collection of highly generic dialogue classes written in Java. These classes represent not just typical dialogue acts, of the type referred to above, but additionally provide a framework of dialogue management and task management classes within which developers may implement the mechanisms for maintaining a human-computer ‘conversation’ and progressing an underlying task. While the QuADS framework is not in itself prescriptive of particular dialogue- and task-management functionality, it provides outlines for some key components, whose interaction makes dialogue- and task-management possible. A representative selection of acts from the Dialogue Act hierarchy is shown in Figure 2.

2.2 THE MAIN COMPONENTS

The QuADS architecture encompasses the full cycle of processing in a spoken dialogue system, as shown in Figure 3. The key steps that are required to support naturalistic spoken exchanges between system and user are outlined in the sections that follow. Particular reference is made to the ISIS-NL implementation.

2.2.1 Speech recognition

The speech recognition component of the system converts the user-utterance to a text string. ISIS-NL uses the Automatic Speech Recogniser (ASR) supplied with Microsoft’s Speech Application Program Interface (SAPI). Since the recogniser (once trained) works in the manner of a general, large-vocabulary dictation engine, very free spoken input is possible and the problem of out-of-vocabulary or ill-formed words at the recognition stage is greatly reduced. The possibility of misrecognition does, however, remain, and for this reason the system’s Dialogue Manager (described in more detail below) has a range of confirmation and grounding tactics – for example, the spoken and textual content synopses that the system provides when it displays retrieved image/video content. The textual representation of the user’s input, whether the input was originally spoken or keyed, is passed to the natural language understanding component.
2.2.2 Natural language understanding
To understand the significance of the user’s input, ISIS-NL spots key words and phrases that are commonly used in the application domain, and that in context can be assumed to have typical pragmatic and semantic weight. (In a well-defined application area, where the possibilities of what the user will say are restricted and the intent behind them is easily understood, the natural language understanding task (NLU) of the overall dialogue system can be greatly simplified.) In QuADS-based applications, Dialogue Act objects represent both the user’s speech act and its semantic content. They are objects of a particular type (an Inform act, a Disambiguate act, a Specify act and so on), and depending on the kind of act they represent, may contain Elements (small chunks of information, often subclassed as problem-solving Resources) that express their full meaning – for instance, an Inform act, generated as a result of a user’s utterance, may be used to tell the system that the user is looking for a Bus Route (a contained Element of sub-class Resource within the Inform object) and that the number of the Bus Route is 57. We shall return to this example in Section 2.3.2. Currently in QuADS the input string is read by a series of word- and phrase-spotters, each of which can recognise concepts relevant to the domain and match these with the most likely dialogue act in the application context.

2.2.3 Dialogue and task management
Dialogue management determines how the system responds to information from the user, and sometimes to information emerging from its own sub-components. As it advances the dialogue towards task completion, the system introduces new dialogue acts and new sub-goals. In ISIS-NL, dialogue management involves filling task-related ‘slots’ of information and monitoring the confirmation status of that information (new, confirmed, modified, negated, etc.); in this sense the process has much in common with the frame-based approaches to dialogue management described by McTear (2002), and exemplified by Heisterkamp and McGlashan (1996). In ISIS-NL, the dialogue management task is broadened to include the choice of modality that the system uses when it communicates with the user: for this turn in the dialogue, will output be speech-based or text-based and will the modalities be used individually or in combination?

In ISIS-NL, responsibility for completing the underlying information retrieval task is shared between a closely collaborating ‘team’ of objects:-

- **The Dialogue Manager** itself, which is responsible for the system’s task-independent dialogue progression. (When should the system confirm user-provided information? How should it confirm it? What happens when there is information to show or describe to the user, or a question to ask? ...And so on.) – In order to decide which dialogue management action it will perform next and in what manner, the Dialogue Manager has, respectively, a number of advancement policies (what to do next) and realisation policies (how to do it). In particular it uses its realisation policies to determine the modalities that will be used to realise a particular dialogue act, given a particular system configuration and user preferences.

- **The Task Manager**, which, as a class of problem solver, has the job of working out whether it has enough information from the user to attempt to answer his or her inquiry, or whether more information should be requested, and what that information should be. – Sitting alongside the broad sweep of the dialogue cycle shown in Figure 1, the Task Manager is invoked when its input is required by the Dialogue Manager, the latter having reached a point in its processing when it is able to turn to the ‘real-world’ task, as opposed to managing the corrections and confirmations of the ‘communicative’ task.

- **The Information Management Agent**, which interfaces with the system’s database in order to retrieve the information that the Task Manager has requested, or, when a specific request fails, to examine the database more closely to see what information might be retrieved if certain inquiry constraints were relaxed. – The Information Management Agent is a domain specialist, encapsulating a combination of real-world expertise (How do I relax the constraints of this inquiry if I am not getting any hits?) and technical know-how (How do I formulate the request for this type of database?).
2.2.4 Natural language generation and speech synthesis
Traditionally, any ‘concepts’ or dialogue acts that are generated into well-formed natural language strings (the role of the NLG component) are also spoken by the speech synthesizer. However, in the multimodal QuADS environment, a generated string may be spoken, or it may simply remain as text and be displayed as such, or a combination of both text and speech may be required. QuADS first ‘internally’ generates the text representing a potential dialogue act, and then assesses how it should be realised, according to system capabilities and user preferences. Indeed, it may reduce the amount of information that is generated, in order to accommodate the output capabilities of the system. In Sections 2.3.3 and 2.3.4 we will further consider this process of realising output.

2.3 USING QUADS TO MAINTAIN A BASIC MULTIMODAL DIALOGUE

2.3.1 The QuADS architecture in context
In QuADS the dialogue cycle of Figure 3 is maintained by a group of Manager components operating within an Assembly, where each manager is called on in sequence – the Assembly Sequence – to provide its contribution to the overall dialogue-handling task. In addition, each Manager is associated with a particular Forum, where it has at its disposal one or more specialised Agents (implemented as software objects) to help it with its dialogue-related task. We have previously discussed our interpretation of ‘agents’ as inheriting, collaborating dialogue-handling experts, implemented as software objects, instances of classes from an object-oriented hierarchy (O’Neill et al., 2004). The forum-based architecture has much in common with the blackboard model, which has been successfully used elsewhere (in terms of dialogue systems perhaps most notably in SmartKom (Wahlster, 2002)). However, whereas the ‘blackboard’ represents a shared information repository that various agents periodically consult, the forum facilitates more direct interaction between the system’s object-components: members of the same forum can interact with one another. For example, the Generation Manager, which is responsible for processing any acts that require to be formed or generated before they are output, is associated with the Generation forum, as is an NLG Agent, which, via the shared forum, is able to provide the Generation Manager with well-formed phrases and sentences whenever they are required. The main features of the QuADS architecture are outlined in Figure 4.

From a historical perspective, the assembly-and-forum architecture of QuADS supersedes the hub-and-spoke architecture used by the Queen’s Communicator, which, though not a DARPA project itself, used the same Galaxy
Communicator hub as was used by the dialogue systems participating in the DARPA Communicator project\(^1\). (The Galaxy Communicator hub was developed by the Spoken Language Systems group at MIT and subsequently extended and released as an open source package by the MITRE Corporation (Bayer et al., 2001).) In the ‘Communicator’ configuration, system components (similar to the system components shown in Figure 3) interact with each other using information that they package and send through the hub as ‘hubframes’. A ‘hub-script’ maintains correct sequences of interaction. Other applications have made use of similar configurations, sometimes referring to the single or multiple hub-like co-ordinating entities in their architectures as ‘facilitators’, ‘facilitator servers’ or even ‘facilitator agents’. Cheyer and Julia (1995) describe their use of the then relatively new Open Agent Architecture (OAA) and its Interagent Communication Language (ICL) (Cohen et al., 1994) to enable task-specific agents to communicate with each other via facilitator agents in the context of a multimodal travel planning application. From a multimodal perspective Cheyer and Julia’s work was particularly interesting in that it tackled the problem (an on-going challenge!) of resolving references that were supplied in different modalities (e.g. speech and gesture) and handled by different modality agents, but that referred to the same real-world object. MATCH (Johnson et al., 2002), is a further example of a system built around a central software ‘facilitator’ (in this case a message-passing component known as ‘MCUBE’). Developed at AT&T, MATCH (Multimodal Access to City Help) also set out to address the problems of appropriately generating, receiving and fusing (for better understanding) information in different modalities, on this occasion taking restaurant and subway information as its application domain. (The developers of MATCH (Johnson et al., ibid.) liken the functionality of their MCUBE facilitator specifically to the functionality provided by the OAA and the Communicator hub.\(^1\)

Architectures involving a central hub or one or more facilitators have much to commend them as a means of coordinating a range of dialogue system components, each with its own task to perform in its own particular modality. However, in moving away from a hub- or router-based configuration, we are exploiting instead the very free, method-calling interaction that characterises object-oriented systems. The use of forums in association with an assembly, as a means of bringing together collaborating system components, opens up a number of interesting possibilities, creating more flexible and richer interactions than the DialogueManager - Problem Solving Manager interaction that we proposed in Chu et al. (2005), an earlier exploration of an extension to the Queen’s Communicator object hierarchy. In the new QuADS architecture, different Managers might, for example, have different Agents at their disposal, so that tasks may be completed in a number of ways. For instance, requests for output might be reinterpreted by Agents that represent human-computer interfaces with very different capabilities, some biased towards use of text, others towards use of audio or video, and so on. Moreover, the forum-based architecture means that information exchange in the dialogue cycle is not necessarily swept all in one direction, from input to output: by making use of Agents in a forum other than the one with which they might most immediately be associated, Managers can incorporate information from any stage of the dialogue cycle into their own decision-making. If, for example, the Dialogue Manager was informed that audio output was being heavily used, it might start

---

\(^1\) [http://groups.csail.mit.edu/sls/technologies/galaxy.shtml](http://groups.csail.mit.edu/sls/technologies/galaxy.shtml)
to gear its dialogue acts towards simple visual output, and so on. And while the QuADS architecture supports predominantly server-side functionality, that receives, processes and outputs information coming from and going to a range of front-end devices, these front-end clients are easily interfaced to the server via IP and port addresses, in the manner typical of current client-server configurations.

In the present paper we have already mentioned use of agents in multimodal systems. On previous occasions we have alluded to examples of their use, and pointed out that, in contrast to the approach we adopted in the Queen’s Communicator and now in ISIS-NL (where a single agent-object might embody expertise for a substantial real-world, user-system interaction – e.g., ticket-booking), agents in other dialogue systems sometimes perform very simple tasks. Turunen and Hakulinen (2001), for example, describe simple generic error-handling agents that ask the user to repeat misunderstood input; Cheyer and Julia (1995) describe collaborating ‘macro agents and ‘micro agents’, the former having more substantial knowledge and ability for domain-specific reasoning, the latter typically handling fine-grained input in a particular modality. Elsewhere we have described how the domain experts or agents in the Queen’s Communicator – each of which was, via inheritance, a ‘dialogue manager’ in its own right – used their sequences of ‘expert rules’ to ask the user for the information needed to complete routine, frame-based transactional inquiries (O’Neill et al. 2003, 2004, 2005).

In the Queen’s Communicator our concern to support maintainability and extensibility, through the use of inheriting and collaborating agent-objects, was characteristic of the software engineer (Hanna et al., 2007). In QuADS, our attempt to capture dialogue acts as families of objects that can be interpreted and acted upon by assemblies of Managers and their Agents, is similarly motivated by an aspiration to good software design (Hanna et al., 2009). In terms of functionality, our main concern in both the Queen’s Communicator and ISIS-NL is with the successful completion of the underlying real-world task: the system performs successfully if its information- and confirmation-requests can be understood and acted on by the user, and if the user is presented with information (details of a completed hotel- or theatre-booking in the case of the Queen’s Communicator, relevant video footage in the case of ISIS-NL), that matches the supplied or inferred constraints of his or her inquiry.

However, other researchers have been more closely concerned with refining communicative efficiency, and in so doing increasing the naturalness, of the agent-based dialogue system’s utterances. One notable example in this regard is PARADIME (Parallel Agent-based Dialogue Management Engine), funded by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO) as part of the IMIX information extraction multi-project. Bunt’s DET taxonomy captures the many dimensions into which dialogue acts may fall: task-related questions, instructions, requests; auto-feedback; allo-feedback; and so on. In the course of dialogue management in PARADIME, specialist Dialogue Act Agents each have the opportunity to generate a candidate dialogue act for the next system turn in their own area of competence: an Evaluation Agent then determines how the acts should be scheduled, perhaps over several turns, or whether some acts should be dropped from the candidate list (since they are implied by other candidate acts), or indeed whether several acts can be combined into what becomes a single multidimensional system utterance (Keizer & Bunt, 2006, 2007). The developers also take into consideration the possibility that an act might be realised non-verbally – by some event on a graphical user interface, for example.

While ISIS-NL does not yet embody PARADIME’s sensitivity to dialogue act realisation, its architecture provides considerable scope for the incorporation of such expertise. The assembly-forum architecture of QuADS envisages a rich collaborative interaction between components. Thus, for example, a Generation Manager (possibly one that learns and adapts) might resolve questions concerning choice of modality by assessing the options proposed in the Generation Forum by a team of information- and context-sensitive Generation Agents.

2.3.2 Reacting to input

In our current QuADS-based implementation, ISIS-NL, for each concept that is recognised, the NLU Manager creates a new dialogue act, from within the taxonomy of act classes available to the system. Let us consider the case where a bus route number is spotted in the input phrase

“... A person who got on the number 57 bus...” (1)

First a problem-solving Resource object is created: a BusRoute, that in this example takes as an attribute the spotted route (57). Again, in a very basic dialogue scenario, where the universe of discourse is closely bounded and a limited range of dialogue acts is expected from the user, simple word- and phrase-spotting is sufficient to identify the

2 http://wwwhome.ewi.utwente.nl/~hofs/imix/index.html
dialogic force behind the user’s utterance: the ISIS-NL system is intended to retrieve footage that satisfies certain constraints, and so phrase (1) may be safely interpreted as the user informing the system about the kind of footage it should retrieve. The BusRoute (with its route attribute) itself becomes an attribute of a Dialogue Act object of type Inform. Figure 5 outlines this relationship. Our use of resources for problem solving in dialogue is influenced by the work of Blaylock et al. (2003, 2005). The following pseudocode represents the process of creating an object-based Inform act, from information ‘spotted’ in the user utterance.

Create new BusRoute instance called busRouteSpot;
Set Route number in busRouteSpot to the busNumber spotted;
Create new Inform instance called busRouteInform;
Set data Element in busRouteInform to busRouteSpot;
Add busRouteInform to the Acts created for this user-turn;

How does the Dialogue Manager respond when it receives such an Inform act? The system has to be able to cope with quite a fluid evolution of the dialogue, where, for example, the task-supporting elements that the user supplies in his or her Inform utterances may contradict or negate what went before (“I meant the number 39 bus.” “Not the number 57.”) To deal with these situations the system, along with its advancement and realisation policies, has an integration policy that enables it first to check whether value types provided by the user can be mapped neatly on to slots relevant to the current tasks, or whether the significance of the values provided must be disambiguated – the latter requiring further interaction with the user. The rules of the integration policy further enable the dialogue manager to deal with those questions of user-supplied values that change from turn to turn: in some cases the system has to make the user aware that it has noticed the change, while in other cases the Dialogue Manager might simply assimilate the change and allow the dialogue to proceed.

From the point of view of multimodal output generation, the system’s behaviour is perhaps at its most interesting when it has acquired a “full set” of information, enough to attempt the information retrieval task and let the user know, in whatever modalities are appropriate and possible, what it has found or, indeed, what alternatives it has to suggest when a user requirement cannot be satisfied.

2.3.3 Realising output

Assuming that the system has found information that it wants to show to the user, it must decide, in the current configuration, whether to output its commentary on that information as text or speech, or some combination of these. Though the development system is PC-based, we consider the possibility that it may eventually run on or be interfaced to a variety of devices, including mobile or in-vehicle devices, which, because of a very small or basic screen (an LCD for instance), may be able to display only a limited number of words per turn. The system therefore takes into account the capabilities (modalities) and capacities (maximum output per turn) of the device, before it realises output for a particular turn. It also takes into account any preferences for a particular modality that the user may have expressed, or that may be indicated by the user’s profile. In our development system, both user- and device-profiles are passed to the QuADS-based Dialogue Manager by the front-end client.

In the current system, for a particular dialogue act (for example a Specify act, where the system wishes to present the user with a number of options to select from), all the Elements (options, in this instance) that the Dialogue Manager has included in the act are generated and prepared for output. However, before the act is output or ‘realised’, the Dialogue Manager applies its modal policies to the act. These modal policies take into account device capabilities and user preferences, and enable the Dialogue Manager to determine which modality will be used and how much information will be conveyed to the user on that turn. A single system turn may comprise several dialogue acts, each of which will be generated in the most appropriate modality for user and system. Currently, if different acts are to be generated in the same turn and require the same modality, those acts are generated sequentially in the particular modality: we do not yet attempt to rationalise utterances by identifying opportunities for ‘simultaneous multifunctionality’ – where, for example, a phrase like ‘let me see’ might realise a turn-taking act and a request for some thinking time (Bunt, 2009b).

In an alternative configuration, it is true, the Dialogue Manager might at the very start of each turn restrict the potential complexity of the turn; or it might give the NLG Manager a dialogue act that comprises multiple elements, but instruct the NLG Manager to generate the act in chunks that can be output over several system turns. While we recognise these as valuable options, which are likely to be the subject of experimentation in the future, our present approach is sufficient to illustrate the concept of adaptation to user and system: the system (internally) generates one
complete dialogue output for the particular turn and then uses this output – in particular its size – to work out how the information will be presented to the user.

2.3.4 Selecting the output modality

At present our research and development concerning modality selection has concentrated on one particular facet of the dialogue scenario: the manner in which the system presents alternatives to the user if an initial user-inquiry has failed. The fact that the system has to present to the user one or more alternatives (or potentially apologise and offer none) makes this a useful and manageable context in which to experiment with chunking information over several turns and across different modalities. In formulating an initial, potential response to the user, the Dialogue Manager internally generates an array of dialogue acts. These are subjected to the Dialogue Manager’s modal policies, so as to determine the manner in which each act will be realised and the amount of information that will be presented to the user by each act. Again, it may be that several dialogue acts are realised in a single system turn. In practice dialogue acts are relatively short – typically informing the user of decisions that the system has made and asking the user to specify additional information. Enactment of our modal policy includes the following steps – and here we will focus on the modality selection process for a Specify act that potentially entails a number of options:

1. The system determines the modalities that can be used to realise the act. Each type of Resource that can be associated with an act (e.g. a BusRoute that the system prompts the user to Specify) has, as an attribute, one or more modalities that may be used to realise it (unless stated otherwise, text and speech are the default modalities for realisation; a non-verbal Resource, like video, will require the corresponding non-verbal modality for realisation). Thus the modal policy considers the Resources associated with a particular act with a view to identifying modalities suitable for realising the act. At this stage appropriate modalities are identified, regardless of their availability for a particular device, and regardless of suitability for a particular user.

2. The system confirms which modalities are available for the device that it is running on or interfaced to. This may mean that a modality that, in theory, could be used for a Dialogue Act, is now deemed unavailable in a particular system environment.

3. The system works out how much information can be handled by each modality on this device. In our initial implementation this is quite a simple metric – representing on a scale of 0 to 1 the degree to which the internally generated act (the potential output for the act) can be accommodated by the particular modality. Let us consider the case of a string that may be realised as text or speech. Since the internally generated string has a length measured in characters, and the modality has an available length that it can accommodate (set as a number of characters by the developer), it is easy to calculate the degree of information load according to the following sigmoid function (in the case of still images to be displayed as a bank of ‘thumbnails’, the number of images may be used as units of length; likewise, if it is possible to stream videos simultaneously, the number of videos may be used):

\[
\text{information load} = \frac{1.0}{1.0 + e^{-5.0 \times (1.0 - \frac{\text{internally generated length}}{\text{available length}})}}
\]

4. The system next considers the user’s preferences for each of the available modalities (represented in each case as strongly like, like, neutral, dislike, strongly dislike and never use.) Each of these preference types is associated with a scaling factor, a simple multiplier, that is applied to the information load value to create a load preference value: less-liked modalities receive proportionately lower load preference values. The load preference values for each modality are compared, and the winning modality – the primary modality in which the act will be realised – is the one with the highest score. (The system may be configured to allow secondary modalities to be used alongside this primary modality, so that, for example a textual output may also be spoken.)

5. In a case where the information load (from step 3 above) is less than 0.5 (which we take as an indication that there is too much information to convey to the user in a single turn in this modality), the system will reduce the information content of the act for this turn – setting the remaining original content temporarily
aside with a view to making it available, if the user wishes to see or hear it, over what might become several
turns.

For example, if the system was planning to ask the user to choose between several options in this turn –
each option being represented by a Resource within the Specify act – it may shorten the content of the act by
progressively removing Resources from the act and generating and re-examining the act until eventually it
may decide to ask the user to choose or reject just one option on the turn. An Inform act accompanying the
newly shortened Specify act will make it clear to the user that the system is presenting just the first of
several options, so affording the user the opportunity to ask for further options, i.e. the Resources that have
been (temporarily) removed on this system turn. Only at this point is the realisation process set in train,
using the selected modality, with an appropriate amount of content, for the particular user. If, because of
the influence of the user’s preferences, a modality is selected that will require the dialogue act to be split
over several turns, that is an acceptable outcome: the user’s preferences are accommodated, even if a more
complex sequence of dialogue interaction ensues.

The effect of the modality selection algorithm and the information load calculation, will, depending on the user’s
preferences and the system’s capabilities, give rise to outputs like Large Capability – Sequence A and Limited
Capability – Sequence A (shown below), which are taken unedited from the current implementation. This example is
for one modality (text) and shows the effect that large and limited output capability in the selected modality has on
the system’s realisation of the dialogue acts. The system is announcing that it has been unable to find an exact match
for the user’s request (which it echoes for grounding purposes) but, by relaxing some of the user-supplied
constraints, has been able to find other matches that the user may wish to consider. In each of the Sequence A
transcriptions, turn System 1b (Inform) is the realisation of a system-internal Report that is intended to provide the
user with a commentary on the system’s actions: in the Large Capability sequence, this Report corresponds to the
concept ‘Specify Choice Provided’ (i.e. all options are being presented); while in the Limited Capability sequence
the Report corresponds to the concept ‘Specify Choice Partial Initial’ (i.e. the first of a range of options are being
presented).

**Large Capability – Sequence A**

System 1a (Inform): So that’s entering the number 45 bus, starting search at 1:00, ending search at 18:15 on
the fifth of October.

System 1b (Inform): - Sorry. I'm unable to match that, but I have 2 alternatives.

System 1c (Specify): Can you tell me if you want: option 1, an alternative bus event; or option 2, an alternative
bus route?

**Limited Capability – Sequence A**

System 1a (Inform): So that’s entering the number 45 bus, starting search at 1:00, ending search at 18:15 on
the fifth of October.

System 1b (Inform): I couldn't find an exact match for your request. Here is the first option I have to suggest.

System 1c (Specify): Can you tell me if you want: option 1, an alternative bus event?

The system can be adjusted to produce even sparser output for a device of limited capability. Omitting the opening
confirmation turn from a limited capability sequence, and generating ‘Specify Choice Partial Initial’ still more tersely
(in turn System 1a), gives the following output (Sequence B):

**Limited Capability – Sequence B**

System 1a (Inform): No exact match found. Options follow.

System 1b (Specify): Can you tell me if you want: option 1, an alternative bus event?
3 LOOKING AHEAD

This research marks an early stage in the process of assessing the ways in which the QuADS architecture might be exploited and extended, and although our implementation for ISIS-NL answers user-inquiries through a combination of speech, text, still image and video, we have yet to explore the most effective balance of these modalities for different user-types in different contexts. At present, for example, it is assumed that the user will always want to see an overview (in the form of small video stills or ‘thumbnails’) of the video content that matches his or her inquiry, before choosing a portion of video to play. However, even this process of video selection may be subject to a number of multimodal permutations: the user might want to read or hear a synopsis of what has been found, before examining thumbnails or videos; or the user might want to read a synopsis alongside each thumbnail; or hear a synopsis for each thumbnail on request.

The most appropriate, indeed the most satisfactory functionality will differ between applications, system configurations, deployment contexts and of course the possibly idiosyncratic preferences of individual users. Part of our ongoing work will be to put in place the mechanisms that allow multimodal output to be realised as flexibly as possible, and then to give the user the opportunity to tailor the system to their particular requirements. While a GUI-based options panel would give a user an immediate means of communicating presentation preferences, in the longer term it is likely that the system’s understanding of user preferences will become more closely tied to the user-system dialogue itself, whether the system asks for guidance directly (‘How am I doing?’), or whether it ‘intelligently’ infers best practice from an analysis of user profiles and user preferences observed from live transactions. Already in the QuADS architecture many important building blocks are in place to help realise these more advanced system behaviours.

REFERENCES


