**QUEEN'S
UNIVERSITY
BELFAST**

Promoting extension and reuse in a spoken dialog manager: An evaluation of the queen's communicator

Hanna, P., O'Neill, I., Wooton, C., & McTear, M. (2007). Promoting extension and reuse in a spoken dialog manager: An evaluation of the queen's communicator. *ACM Transactions on Speech and Language Processing*, 4(3)(3), 1-39. [1255173]. <https://doi.org/10.1145/1255171.1255173>

Published in:

ACM Transactions on Speech and Language Processing

Queen's University Belfast - Research Portal:

[Link to publication record in Queen's University Belfast Research Portal](#)

General rights

Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy

The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Promoting Extension and Reuse in a Spoken Dialog Manager: An Evaluation of the Queen's Communicator

PHILIP HANNA and IAN O'NEILL

Queen's University Belfast

and

CRAIG WOOTTON and MICHAEL MCTEAR

University of Ulster

This article describes how an object-oriented approach can be applied to the architectural design of a spoken language dialog system with the aim of facilitating the modification, extension, and reuse of discourse-related expertise. The architecture of the developed system is described and a functionally similar VoiceXML system is used to provide a comparative baseline across a range of modification and reuse scenarios. It is shown that the use of an object-oriented dialog manager can provide a capable means of reusing existing discourse expertise in a manner that limits the degree of structural decay associated with system change.

Categories and Subject Descriptors: H.5.2 [Information Interfaces and Presentation]: User Interfaces—*Natural language, Voice I/O*; I.2.7 [Artificial Intelligence]: Natural Language Processing—*Discourse, Speech recognition and synthesis*; D.2.11 [Software Engineering]: Software Architectures—*Domain-specific architectures*

General Terms: Human Factors, Design

Additional Key Words and Phrases: Human-computer interaction, speech and language processing, spoken dialog systems, dialog management

ACM Reference Format:

Hanna, P., O'Neill, I., Wootton, C., and McTear, M. 2007. Promoting extension and reuse in a spoken dialog manager: An evaluation of the queen's communicator. *ACM Trans. Speech Lang. Process.* 4, 3, Article 7 (July 2007), 39 pages. DOI = 10.1145/1255171.1255173 <http://doi.acm.org/10.1145/1255171.1255173>

Authors' addresses: P. Hanna and I. O'Neill, The Institute of Electronics, Communications and Information Technology, School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, Northern Ireland Science Park, Queen's Road, Queen's Island, Belfast, BT3 9DT, Northern Ireland; email: {p.hanna,i.oneill}@qub.ac.uk; C. Wootton and M. McTear, School of Computing and Mathematics, University of Ulster, Newtownabbey, BT37 0QB, Northern Ireland; email: {wootton-s1,mf.mctear}@ulster.ac.uk.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2007 ACM 1550-4875/2007/07-ART7 \$5.00 DOI 10.1145/1255171.1255173 <http://doi.acm.org/10.1145/1255171.1255173>

ACM Transactions on Speech and Language Processing, Vol. 4, No. 3, Article 7, Publication date: July 2007.

1. INTRODUCTION

Spoken language dialog systems [McTear 2002] tend to be large, complex software artifacts, and with good reason. Such systems embody a set of algorithmic processes that model the conventions, practices, and safeguards necessary to engage in a spoken dialog. Additionally, dialog systems encapsulate the domain-specific expertise, potentially across several domains, needed to direct the spoken dialog towards useful forms of application. In order to accomplish this task, dialog systems often need to manage interactions and information exchange between several auxiliary software systems.

As with most other software solutions, spoken language dialog systems tend to evolve and change over time. This can encompass an extension of the system into a new domain or new form of application. It can also include the refinement or alteration of existing behavior either in terms of discourse management or domain-specific expertise. Obviously, changes to the system will incur a related cost in terms of the time and effort needed to realize the change.

Studies have shown that modern, large software solutions intended to serve a target user population can expect the majority of effort, accumulated throughout the lifespan of the project, to be allocated to software maintenance [Bennett et al. 2005], that is, modifying existing functionality and adding new functionality. It would appear reasonable to assume that dialog systems with a projected period of maintenance, including those developed as commercial ventures or research platforms, can expect a similar distribution of effort across the lifespan of the project. As such, there is benefit in minimizing the cost of realizing change.

A significant amount of research activity is directed towards improving the maintainability of software (an overview of current practice can be found elsewhere [Pressman 2004; Bosch 2000]). Broadly, the cost of maintenance can be linked to the complexity of the system that is to be maintained (i.e., dependent upon system size, algorithmic complexity, component cohesion/stability, etc.). It is recognized, for example, IEEE [1993], that a key method of minimizing maintenance costs is to ensure that the system is designed in a manner that facilitates change. Towards this goal, a wide range of different software design methods can be used, crudely clustered in terms of those that adopt a structured methodology and those that adopt an object-based methodology. In particular, object-oriented approaches [Booch 1993; Meyer 1997] have been widely employed as they provide a number of recognized benefits in terms of promoting component reuse, extension, and ease of adaptation [Gamme et al. 1995].

An alternative approach that is attracting increasing attention in the research literature is to apply the statistical methods used successfully in speech recognition to the design of other components of a spoken dialog system, such as the speech understanding and dialog management components [Young 2002]. In this approach, there is a minimal dependence on handcrafting explicit rules and a heavy dependence on learning from data.

This article is concerned with outlining and evaluating how an object-oriented approach to architectural design can be applied to dialog management with the explicit aim of facilitating system maintenance, that is, promoting

modifiability and extensibility. More specifically, the article is explicitly concerned with the design of an object-oriented dialog management component that could be expected to interface with other separate components responsible for speech recognition/synthesis and natural language parsing/generation. The examination does not consider low-level implementation specifics; instead it focuses on the high-level architectural design in terms of defining, using, and maintaining discourse-related expertise.

The article is structured as follows. Section 2 provides an overview of related work exploring other approaches to spoken dialog system development and generic models of dialog management. Section 3 provides a brief overview of the Queen's Communicator system and illustrates the framework within which the dialog management component sits. Section 4 details the architectural design of the dialog manager and explores how discourse behavior can be defined, modified, and extended. Section 5 describes how a functionally equivalent application can be developed using VoiceXML. Finally, Section 6 provides details of an experimental evaluation of system maintainability, comparing the two applications developed using the Queen's Communicator architecture and VoiceXML, respectively.

2. RELATED WORK

2.1 Spoken Dialog Systems and Platforms

Spoken dialog systems are increasingly being used in commercial applications. The first systems were mainly telephone-based applications, such as AT&T's Spoken Language Understanding System, an automated call-routing system that fields 15 million calls per-month [Gupta et al. 2006]. More recently, spoken dialog technology has been applied to in-car interactive and entertainment systems, online conversational help services, and computer games. At the same time, there has been widespread interest in academic research in spoken dialog technology as evidenced by the increasing number of sections in conferences devoted to this area as well as a number of special issues of journals. Major research projects include the Galaxy Communicator [Communicator 2007] and TRAINS/TRIPS [Trains 2007] as well as a range of systems developed at research laboratories such as AT&T, MIT, CMU, and CSLI, Stanford in the USA, and Europe projects such as Verbmobil [2007], TALK [2007] and JASPIIS [2007]. While developers in the commercial area tend to focus predominantly on issues such as usability, performance, and standards, academic research aims to advance the current state-of-the-art, for example, with more sophisticated methods for dialog management that emulate human conversational abilities or by applying methods from artificial intelligence and machine learning to model uncertainty in the dialog process and to optimize system performance and design. Currently efforts are underway to bridge the gap between academic and industrial research in dialog technologies (see NAACL-HLT Workshop on this theme [NAACL-HLT 2007]).

A major outcome of research over the past decade has been the development of various toolkits and platforms that can reduce the effort required to develop

Fig. 1. The basic components of a VoiceXML platform.

a spoken dialog system by providing off-the-shelf or easily adapted component technologies. Examples are the Galaxy architecture [Communicator 2007] and related platforms such as the CU Communicator [Pellom et al. 2000], which was used to assist in the development of the Queen's Communicator. The CSLU toolkit [Cole 1999], which has been widely used over a number of years to support learning and research in speech and language, includes a graphical authoring tool (RAD) for design and implementation of spoken dialog systems. Another toolkit used extensively in academic research is TRINDIKIT [2007]. This toolkit was developed within the TRINDI project and is used to build and experiment with dialog move engines and information states, which were the main theoretical contribution of the TRINDI project [TRINDI 2007].

2.2 VoiceXML

VoiceXML (Voice Extensible Markup Language) is a markup language based on XML (extensible Markup Language) for creating spoken dialog systems. VoiceXML was developed within the Voice Browser Working Group of the World Wide Web Consortium (W3C) in order to provide a common language that would support developers familiar with the Web paradigm for application development and to promote code portability and reuse. Version 2.0 of VoiceXML achieved W3C Recommendation status in March 2004 (see <http://www.w3.org/TR/voicexml20/>).

The basic components of a VoiceXML platform are shown in Figure 1. As shown in Figure 1, a user interacts with a VoiceXML application using a telephone and the public services telephone network. This involves calling the VoiceXML browser which is located on a remote server, unlike a Web browser that is normally located on the user's device (such as a PC). The VoiceXML browser includes an interpreter that processes the VoiceXML code for the application and additional supporting components for automatic speech recognition (ASR), text to speech synthesis (TTS), audio files, DTMF (for touch-tone input), and the required telephony infrastructure. The VoiceXML browser also communicates through an application Web server for backend functions such as transaction processing and database interface.

The concept of dialog within VoiceXML is analogous to that of a form on a Web page. A Web form contains a number of fields to be filled by the user, for example,

by entering text into text input boxes or making choices using radio buttons or select fields. Similarly, a VoiceXML form defines a spoken interaction that collects values for one or more fields. Each field in a VoiceXML form contains a system prompt that requests input from the user and a grammar that specifies the set of permissible user inputs for that field.

Dialog control in VoiceXML is implemented using the Form Interpretation Algorithm (FIA). The FIA enables the developer to provide a mainly declarative description of the dialog by simply specifying the fields of a form, including the prompts and recognition grammars. The processing of the form is controlled by the FIA. In the simplest case, the FIA progresses sequentially through each field in a form, executing the prompts and processing the user input. In a more complex case, there may be fields that already have defined values, in which case the FIA will not execute those fields. For example, in a mixed-initiative form, the user can provide values for more than one field within the same utterance as in an utterance such as ‘A three-star hotel in Belfast’ which would provide values for the fields ‘Hotel Class’ and ‘Location’. In this case, these fields would not have to be visited again as their values are already defined, and the FIA would continue to elicit values for any remaining fields. Thus the FIA frees the developer from having to write procedural code to process the various items required to fill the fields of a form at the cost, however, of the greater flexibility that can be provided by a Dialog Manager using more sophisticated methods for dialog control.

VoiceXML also provides a mechanism for handling various types of events, such as “noinput” when the system does not detect any speech from the user, and “nomatch” when the user’s input does not match any item within the recognition grammar for that particular prompt. There is also a facility to define additional events using the <throw> and <catch> elements. Moreover, events can be declared at different levels of scope within a VoiceXML application. For example, events declared within a root document have application-level scope and are inherited at lower levels of scope such as document or dialog (i.e. form) unless a specific event of the same type is declared at such a lower level. In this way, error-handling behavior can be implemented using standard object-oriented concepts such as inheritance.

Forms and fields may also include one or more <filled> elements which may contain blocks of procedural code (in ECMAScript) that execute when field values have been assigned. Generally this code takes the form of conditional statements, for example, to provide integrity checking on the user input or to submit the elicited data to a Web server.

VoiceXML is supplemented by several additional markup languages, including SRGS (Speech Recognition Grammar Specification), SSML (Speech Synthesis Markup Language), SISR (Semantic Interpretation for Speech Recognition), and CCXML (Call Control Extensible Markup Language) (also see the list of specifications on the VoiceXML Forum site [VoiceXML Forum 2007]).

VoiceXML can be generated dynamically from a Web server using technologies such as JSP and PHP. For example, the results of a database lookup can be incorporated into a template for a prompt and spoken to the user as required. It is also possible to dynamically construct recognition grammars by

obtaining values from a database at runtime and incorporating these into a grammar template. In this way recently updated information, such as names of clients, can be obtained from a database and used as recognition elements in the VoiceXML application.

VoiceXML reusability can be facilitated through components such as reusable dialog components (RDCs) and other components that provide functionalities similar to the service and support agents of the Queen's Communicator (see Section 4.2). The Jakarta RDC taglib project provides a wide range of basic reusable components for simple functions such as collecting a date, time, expiration date, or alphanumeric string, as well composite components for set-piece dialogs such as collecting mortgage information, including type of mortgage and percentage down-payment, and collecting credit card information, including type of card, card number, expiration date and security code [Jakarta 2007]. Similar reusable components are provided by Nuance [2007] and VoiceObjects [2007]. In addition to encapsulating basic reusable elements of dialog, these components support an object-oriented development method as they allow other resources such as prompts and grammars to be packaged together with the reusable objects.

2.3 Dialog Management

The dialog manager (DM) is the central component of a spoken dialog system, accepting input from the user, producing messages to be output to the user, communicating with external knowledge sources, and generally determining the dialog flow. Dialog management can involve complex decision-making, such as whether it is preferable to proceed with the next stage of the dialog or whether to clarify or confirm the user's intentions, for example, in the case of a poor speech recognition result.

Broadly speaking, three different approaches have been used in dialog management: finite state-based, frame-based, and agent-based [McTear 2002]. In a finite state-based approach, the states of the dialog, which may represent exchanges involving a system prompt and the possible user responses to that prompt, are determined in advance and can be represented graphically as a directed graph with transitions denoting alternative paths through the dialog graph. This is the simplest method of dialog control which is used extensively in commercial applications to implement system-directed dialog. Frame-based dialog control stores the items to be acquired from the user in a form (or frame). These systems are more flexible as the order in which the items are elicited can vary, and users can specify more than one item within a single utterance as opposed to the state-based method where each item is elicited one-by-one in a predetermined sequence. The frame-based method has been used widely in a number of systems and is also used to implement mixed-initiative dialogs in VoiceXML. Finally, agent-based approaches incorporate a wide variety of approaches that use techniques from Artificial Intelligence to produce more intelligent systems, for example, by using plan-based dialog management [Allen et al. 2000] or some agent-based approach [Turunen and Hakulinen 2003; Bohus and Ridnicky 2003]. The Queen's Communicator system is an example of an

agent-based system as it incorporates mixed initiative interaction in which the user can change or correct items as well as switch domains, and it makes complex decisions on how to ground the dialog based on the current context and the dialog history [McTear et al. 2005].

In its current form, VoiceXML is mainly finite state-based. In directed-dialog mode, the developer specifies the conversational exchanges as fields that are executed in sequence. A limited type of mixed initiative dialog is also possible (see Section 2.2). Currently dialog control in VoiceXML uses the Form Interpretation Algorithm, which frees developers from the necessity of having to explicitly encode dialog control but with the disadvantage of inflexible dialogs. For example, it is difficult to develop a dialog system in VoiceXML in which the user can take the initiative to correct or request clarification of some information or change topic. However, there are efforts within the Advanced Dialog Working Group, which is part of the VoiceXML Forum Tools Committee, to extend the functionality of VoiceXML [VoiceXML Forum Tools 2007]¹. Some of the issues discussed at a recent workshop include the specification of state transitions at a higher level of abstraction, the integration of rules and goal-driven methods into VoiceXML, and the application of probabilistic modeling within VoiceXML applications [VoiceXML Forum Dialog 2007].

Until recently, spoken dialog systems have been developed using handcrafted methods for spoken language understanding and dialog control. For example, handcrafted grammars are used in the MIT TINA system [Seneff 1992] and in the Phoenix parser [Phoenix 2007] which is also used for spoken language understanding in the Queen's Communicator system. More recently a data-driven approach has emerged, drawing on methods used widely in information extraction and question-answering (for a recent collection of papers on data-driven spoken language understanding, see Bangalore et al. [2006]). Currently the grammar rules in the Queen's Communicator are handcrafted. However, there is no reason why the grammar rules used in the Queen's Communicator could not be learned from data. Similarly, machine learning has also been applied to dialog management. In this case, the dialog system is modeled as a Markov Decision Process and reinforcement learning is applied to find the optimal dialog management policies [Young 2002; Walker 2000; Levin et al. 2000]. This approach could also be applied in the Queen's Communicator, for example, to determine the best policy for handling errors. Currently the rules for error handling are handcrafted, but they could in principle be learned from data. The expected benefits of the data-driven approach in comparison with handcrafted and knowledge-based methods are a reduction in development time, greater robustness, and less developer bias. The main disadvantage is the requirement for a large amount of suitable training data along with the need for annotation using human labelers.

¹Nuance is currently developing xHMI, an XML-based dialog configuration language intended to enable efficient development of more natural, conversational applications. However, as xHMI has not yet been published it is not possible to include a discussion of it within this review (see <http://www.nuance.com/xhmi/faqs.asp>).

Fig. 2. Architectural overview of the Queen's Communicator.

2.4 Summary

There is a wide range of approaches to the design and implementation of spoken dialog systems. The current article illustrates a knowledge-based approach in which the use of an agent-based approach is considered to provide a number of benefits in terms of improving the flexibility and adaptability of the dialog manager through mechanisms such as the: separation of task and discourse behavior within and between agents, use of distributed processing to support dynamic forms of inquiry and/or leverage of individual agents, and the ability to employ a plug-and-play capability whereby new agents can be added into the system.

3. OVERVIEW OF THE QUEEN'S COMMUNICATOR

An architectural overview of the Queen's Communicator system can be seen in Figure 2. The system consists of three main nodes with a number of different deployment options. A Java-based client program (i.e., potentially embedded within a Web page) provides a means of capturing user input speech which is then encoded and sent via a TCP/IP connection to the speech server where it is recognized by a SAPI compliant speech recognizer before being sent as a string of recognized text to the dialog server node.

The dialog server is comprised of a natural language parser (the Phoenix parser [Phoenix 2007]), dialog manager, backend database server, natural language generator (NLG) and an affective mark-up generator (using the Affective

Fig. 3. Illustrative overview of the dialog manager component.

Presentation Mark-up Language (APML) [De Carolis et al. 2003; Humaine WP8 2007] which augments the NLG output with appropriate affective tagging. The server makes use of a hub-and-spoke communications infrastructure, made possible through the use of the Galaxy Communicator [Communicator 2007] architecture, to facilitate interaction between components. The output from the dialog server is sent to the client and consists of the generated textual response from the dialog manager and associated APML mark-up to be used to determine how the utterance will be spoken and expressed by the embodied conversational agent (ECA). Once received by the client, an APML wrapper is used to handle the translation of the APML mark-up into facial expression directives for the Haptek ECA [Haptek 2007] and the subsequent phone-based alignment of the ECA against the synthesised speech (obtained using the MS Speech Synthesizer).

Within the Queen’s Communicator, the dialog management component is intended to permit cross-domain, mixed initiative spoken exchanges that deal with inquiry or transactional forms of discourse, that is, encompassing arranging bookings, information lookup enquiries, etc. By means of illustration, an overview of the key subcomponents of a cinema booking system can be seen in Figure 3. In particular, the dialog manager is comprised of a number of inquiry agents, such as cinema booking experts and payment experts, which interact together to provide an intended form of discourse. In this regard, some of the inquiry agents may adopt a support-oriented role, offering discourse services to other agents. Specialized forms of inquiry agent inherit a rule-based engine and define a number of expert rule collections that can be used to direct and control the discourse. In addition, all inquiry agents inherit a common core of discourse management behavior, thereby ensuring all agents employ a similar discourse style. A domain spotter object is used to manage interactions between the user and agents. Finally, a shared discourse history is used to provide an evolving account of the dialog.

The dialog manager was developed using Java (as is the majority of other components). Further information concerning the structure and operation of the Queen’s Communication can be found elsewhere [O’Neill et al. 2003, 2005].

Fig. 4. Principle, top-level objects defined within the dialog manager.

4. DIALOG MANAGER ARCHITECTURAL DESIGN

4.1 Encapsulating Expertise within Objects

As with any object-oriented approach, a key initial step is to identify the principle top-level objects that need to be introduced in order to appropriately model the problem. This represents an important phase of the design in that decisions made at this point will direct much of the subsequent development. In particular, there is a danger that inaccurate or unnecessarily limiting assumptions introduced at this point will negatively impact on the capabilities and extensibility of the developed system.

Within the Queen's Communicator, the selection of top-level objects has attempted to define a structure that encapsulates the notion of multiple, domain-specific topics of discourse. While not directly impacting the selection of top-level objects, the development has also assumed that an individual topic of discourse will be transaction or inquiry-based.

The principle top-level objects shown in Figure 4 were selected. Central to Figure 4 are the InquiryExpert objects, also referred to as expert objects or more simply as agents, which represent the basic element of encapsulated discourse expertise. Conceptually, each expert object can be considered as an instantiation of a particular form of agent, that is, an autonomous entity which can serve a human interlocutor through spoken dialog towards completion of some shared task. In order to facilitate reuse, it is intended that each agent will encapsulate a well defined and bounded role, for example, collecting a fixed segment of information or performing a well understood action. Examples of such roles include collecting a telephone number and deciding upon a particular time and/or date. As such, the dialog system can be broadly viewed as consisting of a collection of agents that, through collaboration, are capable of offering one or more services which can be combined together to provide complete domain-specific transactions and inquiries.

While each agent will independently engage in discourse, a mechanism is needed whereby collaboration/interaction between agents can be managed.

A number of different forms of collaborative agent behavior can be identified [Parunak et al. 2003], for instance, centralized/decentralized or cooperative/contentious. Within the Queen's Communicator, it was decided that a controlling agent would be used to manage a form of centralized collaboration between agents. The controller, as represented by the DomainSpotter object in Figure 4, has overall responsibility for handling both user interaction and interobject requests. In particular, the controller provides a mechanism whereby collaborative requests from agents and requests from the user can be competitively assigned to the most capable agent. As such, the controller agent does not initiate collaborative behavior but rather determines how collaborating agents are selected. A controller object was favored as it provides a straightforward mechanism whereby application-specific discourse management behavior can be readily incorporated within a single object, that is, providing a single point of contact for all interexpert interaction.

Finally, a DiscourseProduct object was introduced to provide a common model of the entire discourse which is shared between the controller and agent instances.

In the following sections, each object identified in Figure 4 is further explored.

4.2 Agents: Expert Objects

As outlined, the dialog manager consists of a collaborating set of autonomous agents, with each agent offering a measure of expertise within a particular domain. A number of different approaches can be adopted towards meaningfully organizing this set of agents, for example, in accordance with area of expertise, level of specialization, etc. Within the Queen's Communicator, agents have been separated into two categories, namely, service agents and support agents. Service agents are classified as those agents that provide complete frontline services to the user, that is, forms of interaction that encapsulate stand-alone transactions and/or inquiries. Examples include booking cinema tickets, arranging hotel accommodation, scheduling a journey, etc. In contrast, support agents are intended to provide generic supporting services, for example, eliciting a well defined segment of information that is needed within a number of wider inquiries.

In broad terms, support agents can be viewed as providing set-piece dialog assemblies that are of use within a number of different inquiries/transactions and, as such, represent portable reusable elements of discourse functionality. Service agents can be viewed as encapsulating domain-specific transactions/inquiries that can draw upon a library of reusable supporting agents. A support agent may also draw upon other support agents as necessary. Hence, within a typical dialog, the user may wish to complete a number of specific inquiries/transactions, for instance, booking cinema tickets and a table at a nearby restaurant, with each type of inquiry/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 other support agents in order to fulfil its objectives (e.g., a support payment agent may request that address information is collected by an address agent).

Fig. 5. Generalized form of agent hierarchies and example service and support hierarchies.

Other than separating agents into service and support categories, agents within each category may be further refined into domain-specific hierarchies. Each hierarchy structures a number of agents that share a common domain of expertise according to their level of specialization within the shared domain. For example, a number of different support agents dealing with payment might be introduced, some of which might deal with very specific forms of payment, for example, a certain form of store credit card, while others might deal with payment at a more general level. Figure 5 shows the general case and also provides an illustrative example².

The structuring of agents into hierarchies of increasing specialization provides a mechanism that permits an inquiry to evolve from a more general form into a more specific form, for example, as additional information is collected from the user, the discourse evolves. Alternatively, an unsuccessful inquiry might back off from a specific form of agent to a more general form of agent. While not explicitly visible at this level of description, the representation of the agent hierarchy in terms of an underlying object-oriented inheritance hierarchy provides a means of sharing common domain expertise which can then be readily augmented through the addition of more specialized domain expertise (see Section 4.7).

4.3 Managing a Collection of Agents

As mentioned, a primary design goal of the Queen's Communicator is the production of a dialog manager that facilitates the straightforward extension, modification and deployment of expertise. The segmentation of domain expertise into a number of discrete expert objects represents an initial step towards the realisation of the design goal, however, in order to more fully realize the goal, it will be necessary to ensure that agents can be added, removed, or modified

²The current structuring of agents into service and support groupings, alongside the formation of agent specialization hierarchies, provides a basis through which agents can be organized and managed. It is likely that, for large numbers of agents, an extension to the current model will be desirable in terms of providing a more easily managed/maintained structure, for example, making use of the standard software object decomposition into packages of related agent hierarchies.

in a manner that has minimal interdependence upon other agents. As a means of minimizing the degree of dependence, it was decided that agents should act autonomously and not require any specific hardcoded knowledge of any other agent. A number of mechanisms were used to achieve this.

- Centralized Control*. The DomainSpotter will act as an intermediary for all interagent collaboration, that is, all requests for supporting agent services will not involve direct contact between agents, thereby lessening the degree of coupling and hence interdependence between agents.
- Request in Terms of Desired Expertise*. All interagent discourse requests are structured in terms of a statement of desired expertise, that is, desired behavior/functionality, and not in terms of a specific requested agent.
- Registration of Expertise*. On system commencement, all agents will be required to register themselves with the DomainSpotter, thereby making available their expertise to the wider system.
- Agent Self-Selection*. Each agent is responsible for ascertaining and indicating if it can adequately satisfy a request for a certain form of expertise.

Taken together, the mechanisms ensure that agents do not directly refer to other agents and that new agents can be added to the system and automatically make available their expertise to any requesting agents, that is, the operation of the system is dependent on the available registered expertise and not dependent upon registered agents. Based on the service/support agent hierarchies outlined in Figure 5 and the organizing mechanisms outlined previously, a number of different forms of agent interaction arise, for example, selecting a service agent to fulfil a user's request, contracting a support agent on behalf of another service/support agent, transferring control to another/previous agent, evolving a more general form of inquiry into a more specialized form of inquiry, etc. Further details concerning the different forms of agent interaction can be found elsewhere [O'Neill et al. 2005]. Common to the various forms of algorithm that enable agent interaction is the underlying selection of a service or support agent to fulfill a user or agent request for certain discourse expertise. An overview of the employed algorithm is shown in Figure 6.

An example of service agent selection following a fictitious user travel request is shown in Figure 7. By expressing agent selection requests in terms of an expression of desired expertise and providing a controller agent, that is, the DomainSpotter object, both system maintainability and extensibility are improved as the operation of the system is not dependent on the agents that are registered but is instead dependent on the expertise that registered agents make available. A general area of expertise can be extended to include more specialized expertise through the introduction and linkage of appropriately refined agents with associated selection matched to whatever agent can best fulfil the user's request as the inquiry evolves. Additionally, complementary/competing areas of expertise can also be readily incorporated and selected as necessary, for example, in Figure 7, additional agents that offer other forms of travel expertise might be introduced with the polling mechanism ensuring that the most appropriate agent is selected to handle the user's inquiry.

Fig. 6. Overview of polling algorithm used to select agents to fulfil discourse expertise requests.

Fig. 7. Example of service agent selection given a user travel request.

4.4 Developing a Dialog Product

The design of a discourse product was approached from the perspective of promoting extensibility, maintainability, and reusability through the use of an object-based approach. As the design of the Queen's Communicator has proceeded on the assumption that discourse will be transaction or inquiry-based, a slot-filling approach was selected as an appropriate means of representing the information set associated with each inquiry or transaction. More specifically, it was assumed that a frame comprised of a number of slots containing items of information would provide a sufficient basis on which to model the dialog product.

Fig. 8. Class structure used to model a frame of information.

An overview of the class structure used to represent `DialogueFrame` objects within the Queen's Communicator can be seen in Figure 8. `DialogueFrame` objects are decomposed into a collection of attribute objects which encapsulate discourse-specific information, for example, discourse status, system intention, a numeric pegged confidence used to provide a measure of grounding confidence in the user supplied slot value, alongside the value(s) stored within the slot. Table I and Table II provide an outline of the different confirmation statuses and system intentions used within the system. Additional information concerning the class structure shown in Figure 8 can be found elsewhere [O'Neill et al. 2005].

As shown in Figure 8, attributes within a frame can be classified in terms of type as either user or system attributes. User attributes are intended to represent slots that can be directly populated using information supplied by the user. In contrast, system attributes are intended for internal use and cannot be populated by the user, providing a mechanism for representing information that has currency within a particular inquiry/transaction but is not directly supplied by the user. Some examples of system attributes are slots that hold information inferred from user utterances to be used to control the order in which

Table I. Slot Confirmation Statuses Used within the Queen's Communicator

Confirmation Statuses	
NEW_FOR_SYSTEM	Initial user provided value for a given slot
INFERRED_BY_SYSTEM	Value deduced by system based on other populated slots
REPEATED_BY_USER	Existing value repeated/confirmed by user (either implicitly or explicitly)
MODIFIED_BY_USER	Value modified by user
NEGATED_BY_USER	Value negated by user
MODIFIED_BY_SYSTEM	Value modified by system (most likely to enable dialog furtherance given an impasse arising from the previous value)
NEGATED_BY_SYSTEM	Value negated by system (most likely to enable dialog furtherance given an impasse arising from the previous value)
CONFIRMED_BY_SYSTEM	Value confirmed by system (most likely via a successful database lookup)

Table II. Slot System Intentions Used within the Queen's Communicator

System intentions	
CONFIRM	Implicitly confirm the stored value
EXPLICIT_CONFIRM	Explicitly confirm the stored value with the user
REPAIR_CONFIRM	Confirm that the user has repaired a value
EXPLICIT_REPAIR_CONFIRM	Explicitly confirm a repaired value with the user
REPAIR_REQUEST	Request that the user repair a negated value
REFORMULATION_REQUEST	Request that the user reformulate their inquiry (only used when the system is unable to proceed with the inquiry)
SPECIFY	Request that the user specify a particular value
SPECIFY_ALTERNATIVE	Request that the user specify a value other than the current value
UNSPECIFIED	No system intention

subsequent slots are filled and Boolean flags that record discourse progression or database confirmation of user supplied values.

The refinement of slots into those holding information considered to be of direct interest to the user and those holding background or supporting information provides a means of cleanly encapsulating all pertinent information relating to a particular inquiry/transaction or well defined subdialog, that is, set-piece dialog assembly, within a single frame object. In a manner analogous to the support agent hierarchy outlined in Section 4.2, the encapsulation of information relating to set-piece dialog assemblies within frame objects provides a means of reusing common segments of information.

The encapsulation of transaction/inquiry information into frame objects also provides a clean mechanism where the expertise encapsulated within agents can be mapped onto the information that needs to be collected to fulfil a particular transaction or inquiry. A number of different forms of agent-to-frame linkage can be adopted. Examples include multiple agents collaborating to populate a given frame with information, one agent responsible for populating several different frames of information, etc. Within the current implementation of the

Queen's Communicator, it was decided to opt for a simple form of agent-frame relationship whereby at most one agent can be given responsibility to populate a given frame of information. This approach was adopted as it was considered to provide an extensible relationship between agents and frames.

- Given the identification and selection of an agent to fulfil a request (either originating from the user or from another agent), the selected agent will be requested to identify the frame of information that will be populated in response to the request. The agent will be given exclusive responsibility to populate the selected frame.
- An agent may request that control be relinquished and that responsibility to populate the selected frame of information be passed to another agent (e.g., as the dialog progresses, the agent may decide to cede control to a more specialized variant).
- By exploiting the inheritance hierarchy between frames, that is, where one frame can be extended to provide a more specialized frame variant, if control from one agent is passed to a more specialized variant of that agent, then the frame associated with the inquiry may also evolve to a more specialized frame variant. For example, a dialog managed by a `cardPaymentExpert` agent populating a `cardPayment` frame might evolve to a dialog managed by a `visaCardPaymentExpert` populating a `visaCardPayment` frame.

From a design perspective, the expertise needed to fulfil a particular request is encapsulated within an agent, while the information associated with the particular request is encapsulated within a corresponding frame object. By separating expertise and information in this manner, it becomes possible to permit one agent to deal with any number of different instances of a particular inquiry/transaction at the same time, for example, as might arise when dealing with payment across a number of different transactions.

4.4.1 Linking Frames Together to Form Larger Discourse Entities. Having introduced a frame as encapsulating the information associated with the operation of a particular agent, it becomes necessary to define how instances of frame objects collectively represent the evolving discourse product (including dialogs that involve a number of different inquiries/transactions managed across a number of different agents). Within the Queen's Communicator, it was decided that individual frames of information could be linked together to provide composite discourse objects that model full inquiries/transactions (i.e., including any well defined subdialogs that can be encapsulated within a single frame).

Linkage between frames is achieved through the use of linked-frame attributes which provide a mechanism whereby a particular frame can indicate that additional frames of information may need to be collected and attached. For example, a theater inquiry may require linkage to a credit card frame representing payment information, which in turn may require linkage to frames holding address and telephone contact information. This example is illustrated in Figure 9.

In order to ensure that newly introduced and revised agents are given an opportunity to deploy their expertise, the linkage between frames is not

Fig. 9. Example dialog frame linkage.

predetermined. Instead each linked-frame attribute is described in terms of a statement of requested content with the agent responsible for populating the frame determining when a suitable linked frame should be attached (i.e., through the identification of a suitable agent that will be given responsibility to populate the linked frame). The polling mechanism outlined in Figure 6 is used to select the agent that will be given responsibility to both identify and populate the linked frame.

The representation of the discourse product associated with a particular inquiry/transaction in terms of a hierarchy of linked frames may be viewed as useful from a number of different perspectives. In particular, it: introduces a mechanism where a complex inquiry/transaction can be broken up into a number of smaller-sized chunks, which facilitates both the delegation and delimitation of agent responsibility in terms of tasking each agent to collect a certain frame of information, and it provides an open-ended structure that can dynamically grow and adapt depending on the objectives set by the user, for example, through the use of dynamic linkage, it becomes possible to incorporate information imparted by the user in the agent polling process, thereby ensuring that the most appropriate frame of information is incorporated within the dialog.

However, making use of a hierarchy of linked frames is not without its challenges. In particular, the division of an inquiry/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 in future, that is, currently unlinked, frames. In particular, if the user returns to a previously completed frame, any system attributes must be either reset, retested, or regenerated to ensure that any new information introduced by the user is suitably propagated

Fig. 10. Overview of how an evolving dialog product is recorded and maintained.

throughout the dialog (as explored in Section 4.8, this can be readily achieved through the use of housekeeping expert rules). In order to determine if information is of relevance to an as yet, unlinked frame, it is necessary to perform a forward recursive search through all potentially introducible frames and to introduce and link such frames to the current frame where necessary. Details of how this is accomplished within the Queen's Communication are reported elsewhere [O'Neill et al. 2005].

4.4.2 *Modeling the Evolving Dialog.* Figure 10 illustrates how a record of the evolving dialog is kept via two objects: a DiscourseStack and an ExpertFocusStack.

During each dialog turn, a new frame will be generated by evolving the previous state of the current inquiry/transaction with the most current user utterance and the corresponding system response. This new frame is pushed onto the discourse stack, that is, the discourse stack provides a turn-by-turn history of discourse development. No restriction is placed on the order in which different types of frames can be added to the stack, for example, frames associated with two different inquiries will be intermixed if the user jumps back and forth between inquiries. The expert focus stack is comprised of a number of pointers to frames within the discourse stack that identify the most recent frames associated with ongoing inquiries/transactions. As such, whenever the current active inquiry/transaction is completed, the next frame indexed by the expert focus stack is extracted and pushed onto the top of the discourse stack.

The use of these two stack-like structures provides a simple and effective means of dealing with several different interleaving inquiries/transactions. This is illustrated in Figure 11 which shows a situation where the user has left an uncompleted theater booking and started a new cinema booking. While the discourse stack provides a complete account of turn-based interaction, the expert focus stack references only the most recent frames associated with ongoing

Fig. 11. Example DiscourseStack and ExpertFocusStack for a cinema and theater inquiry.

transactions or inquires, in this example, the current cinema booking and the recent theater booking. Once the cinema booking has been completed, or should the user indicate they wish to return to the theater booking, the theater frame referenced by the expert focus stack will be popped onto the top of the discourse stack, thereby enabling the discourse to continue from the original point of departure.

4.5 Internal Agent Structure

While the design of agents as autonomous entities that can individually engage the user in dialog results in a simplified architectural model of the system, it requires that agents must independently deal with the issues surrounding spoken dialog. For example, in addition to undertaking domain-specific 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-initiated repairs/negations are appropriately executed, etc.

Through the use of object inheritance, it becomes possible to define a structure that permits the common discourse behavior (e.g., the broad mechanism where information is confirmed or repaired) to be separately encapsulated and inherited by all agents. Additionally, the use of inheritance also provides a mechanism through which the system can create more specialized agents building upon the defined expertise of more generalized agents. Figure 12 provides an overview of the class hierarchy employed within the Queen's Communicator.

4.6 Discourse Manager: Encapsulation of Generic Discourse Behavior

The DiscourseManager class has responsibility for the overall management of turn-based interaction 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 dialog, alongside common forms of discourse behavior such as grounding, repairing, and arbitrating between different dialog intentions. In other words, the discourse manager encapsulates

Fig. 12. Internal agent class structure.

the common discourse behaviour that is inherited and used by all enquiries and transactions. The discourse manager will only permit the execution of domain-specific expertise if it is satisfied that all previous system intentions have been acted on with appropriate grounding where necessary. An overview of the system-turn process is shown in Figure 13.

An agent invokes its own system-turn process once the semantic parse of the user's input utterance has been converted into a corresponding frame object holding the user supplied information. The input frame evolves against the last frame pushed onto the discourse stack (i.e., representing the latest state of the inquiry/transaction). The evolve process determines how the user's input changes the previous discourse state. Within the Queen's Communicator, the evolve process is controlled by a number of update rules that are conditional on the new and previous contents of a slot combined with the previous, if any, system intention associated with the slot. For example, if a particular slot contains an existing value and the user's input supplies a different value that populates this slot, then, depending upon whether the previous system intention for the slot was to ask the user to supply a different value, the status of the slot will be set to either *Modified.By.User* or *New.For.System*. Once evolved, a second rule-based process assigns generic system intentions to each slot, depending on the status of the slot (e.g., a *Modified.By.User* status will likely attract a system intention of *Repair.Confirm*). The set of generated generic intentions will determine if domain-specific expertise can be invoked. More specifically, provided

Fig. 13. Overview of discourse manager system turn.

none of the slots contained within the frame need to be specified, repaired, or explicitly confirmed, then domain-specific expertise can be later invoked later to further the discourse.

A final rule-based process determines the semantic content of the system's response based on the generated set of intentions for the newly evolved frame. The rules dictate the precedence of different system intentions and also limit the amount of new information that is either provided or requested during a single turn. For example, the rules determine the maximum number of slots that the system can ask the user to populate in a single turn, or the maximum number of values that should be implicitly confirmed in a single turn. If confirming several values, a number of interpretative rules are used in conjunction with the rules concerning implicit confirmation to cater for situations where an uncertain number of slots have been negated. Additionally, if the user populates several slots following a system request, the parser-to-slot mapping will be subject to the normal confirmation strategy, that is, the user will be given an opportunity to rectify any incorrect slot values.

By varying the rule parameters that direct the generic system-turn process, it becomes possible to alter the nature of the dialog, for example, from a discourse style that is highly restrictive and cautious to a style that is more flexible and user-led. Once the system turn has completed, the evolved frame will be pushed

onto the discourse stack and control passed back to the agent which will then invoke its domain-specific expertise if appropriate.

4.7 Enquiry Expert: Encapsulation of Domain-Specific Expertise

In order to exploit the generic turn-based discourse management provided by the DiscourseManager, it is necessary to introduce a mechanism where domain-specific expertise can be added to provide meaning to the contents of each slot and purpose to the execution of system intentions.

Given the use of a slot-filling approach within the Queen's Communicator and the desire to promote the reuse and extension of domain expertise, it was decided that a rule-based approach would be used to encapsulate domain expertise (the mechanisms that facilitate rule reuse and extension are detailed in Section 4.8). In particular, it was decided that expert rules would be defined in terms of a number of preconditions and a number of actions. Preconditions are expressed in terms of conditional properties of specified attributes, for example, the attribute's value or confirmation peg, or the system intention associated with that attribute. In turn, actions associated with a particular rule are concerned with discourse furtherance. This can include changes to the properties of an attribute, for example, the system intention, and also encompass the initiation of database inquiries, NLG utterances, discourse flow changes, inter-agent collaboration, etc. In broad terms, the preconditions of a rule describe a particular situation, for example, the user has provided a cinema name and a particular time, while the actions describe what should be done to appropriately further the dialog, for instance, a database inquiry should be initiated to establish which movies are showing at the stated time. Hence, 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.

In order to improve code reuse, all common functionality concerning the sequencing, testing, and execution of expert rules is extracted and encapsulated within an inheritable rule-based engine. This is identified as the InquiryExpert class shown in Figure 12, which can be viewed as containing a number of sequences of expert rules which are tested and executed as appropriate. Classes which extend InquiryExpert inherit both the turn-based discourse management functionality of DiscourseManager and the rule-based engine of InquiryExpert and, in most cases, need only define the sets of rules that encapsulate the functionality needed to execute the inquiry/transaction offered by the agent. From the perspective of the dialog developer, the introduction of a new agent requires little effort beyond the creation of appropriate expert rules, that is, the InquiryExpert class encapsulates and defines the forms of task-specific behavior that extending agents can employ.

An overview of the rule-based system-turn process is shown in Figure 14 (Section 4.8 provides details on the different types of expert rules available within the system).

The InquiryExpert system-turn process initially invokes the discourse management behavior inherited from DiscourseManager. Following this, the agent

Fig. 14. Overview of inquiry expert system turn.

is then given an opportunity to test and fire all defined housekeeping rules. If the initial discourse management indicates that domain-specific expertise should be employed to further the discourse, then the agent will consider all defined discourse furtherance rules. Unlike the housekeeping rules where any number of rules is permitted to fire, only one discourse furtherance action will be selected to fire. Where necessary, an updated system utterance will be generated to convey the discourse furtherance action undertaken by the expert. The final updated frame can then be stored and the system utterance forwarded to the NLG for generation. Finally, post-system turn actions are considered (e.g., initiating an interagent transfer, dispatching a database inquiry, etc.).

4.8 Enquiry Expert: Rule Type and Structure

As indicated, the notion of an expert rule is central to the encapsulation of domain-specific expertise within the Queen's Communicator. A rule can be defined as consisting of one or more preconditions expressed in terms of slot

Table III. Expert Rule Syntax and Example Conditions and Actions

Expert rule syntax			
{Condition ₁ , ..., Condition _n } → {Action ₁ , ..., Action _m } where n,m ≥ 1 Condition _i ≡ ConditionType, {[NOT]Value ₁ , ..., [NOT AND OR] Value _j } Action _i ≡ ActionType, {Value ₁ , ..., Value _j }			
Example Conditions		Example Actions	
Status, Intention	Current slot status and associated system intention	Set_Intention, Set_Status, Set_Value	Set identified slot property to specified value
Value_Within, Value_Contains, Value_Is	Open and closed tests based on the current slot content value	DB_Request	Issue a database lookup request to populate/confirm a slot
PEG_GreaterThan, PEG_LessThan, PEG_EqualTo	Magnitude tests based on the numeric confirmation peg value	Expert_Transfer, Contract_Expert	Transfer control to another expert. Contract another expert to collect some information
Unspecified, Specified, Confirmed	Commonly employed slot conditions	Response	Semantic utterance to be forwarded to the NLG

properties, combined using standard logical operators, and one or more associated actions describing a course of action should the preconditions be met. An overview of the rule syntax and example conditions and actions are shown in Table III.

It is likely that a complex inquiry/transaction will require the development of a sizeable number of expert rules to adequately model the necessary domain expertise. This was felt to be potentially problematic, in that as the number of expert rules increases, it is likely that the associated cost of maintaining/extending the rule collection (e.g., adding new rules, modifying behavior, etc.) will also increase. As a means of limiting this cost, and thereby improving overall maintainability and extensibility, the notions of rule grouping and rule categorization were introduced.

Rule groupings, as facilitated through the ExpertRuleSequence object identified in Figure 12, provide a straightforward means of grouping together a number of related rules into a sequence that represents a larger chunk of expertise. For example, expert rules governing the booking of a cinema ticket might be grouped into separate rule sequences that deal with the elicitation and verification of different transactional phases, for example, selecting the movie, selecting the cinema, selecting the desired date/time, collecting party information, etc.

Rule categorization provides a means of grouping together rule sequences that share a common functional form into a higher-level organizational structure. Currently, within the Queen's Communicator, three different rule categories are defined.

—*Discourse furtherance rules* are concerned with progressing the discourse in a manner appropriate to the current discourse state as encapsulated within the

latest evolved dialog frame. Examples of common actions include prompting the user to supply more information and initiating a database confirmation or lookup.

- Housekeeping rules* are concerned with maintaining the integrity of the inquiry/transaction, for example, resetting confirmation flags and system-inferred values should the user change a related data item (thereby enabling the discourse furtherance rules to regenerate/reconfirm system attributes).
- Database furtherance rules* are specifically concerned with recovering from unsuccessful database lookups by progressively relaxing the associated lookup constraints in a domain-specific manner until a potential lookup value can be returned. For example, given a failed hotel accommodation lookup for a four-star hotel in Belfast, a database furtherance rule may request that the accommodation class constraints (i.e., four star) be relaxed in order to determine if a database match can be found for the remaining constraints (i.e., hotel in Belfast). Structured as such, the database furtherance rules represent recovery strategies that enable the system to offer potentially viable alternatives to a user inquiry that cannot be satisfied.

The grouping of related rules into expert rule sequences and the use of different rule categories provides a mechanism to decompose a potentially large and complex collection of expert rules into a number of smaller functional segments which tend to afford more straightforward maintenance and extension. Through the use of inheritance, this principle can also be extended to permit more specialized agents to inherit and refine the rule sequences of more general agents. Hence, it becomes possible to define generic or general rule collections that can be shared across a number of different forms of inquiry or transaction. Figure 15 provides an example of how this might be applied within a hierarchy of telephone number agents.

Each extending agent inherits the discourse furtherance, database furtherance, and housekeeping rule sequences from all defined forms of more general agent (i.e., all superclass agents). The agent may then extend the inherited rules sequences by introducing its own specialized sets of rules. When determining which rules should be tested and potentially fired, the more specialized rule sequences are tested first before backing off to more general rule sequences. The first discourse furtherance or database furtherance rule to fire will stop the search process, whereas all defined and inherited housekeeping rules will be tested and fired as appropriate. Defined as such, this structure permits an agent to both introduce new expertise and refine inherited expertise by overriding inherited rules (i.e., it operates in a manner similar to the normal rules of inherited behavior within object-oriented languages). In effect, the use of inheritance provides an additional mechanism where complexity can be managed across a number of different layers and also helps promote the reuse of discourse expertise. In addition, the use of inheritance provides a mechanism to encapsulate cross-domain behavior within a generic, cross-domain agent which can be inherited and further specialized by domain-specific agents.

Fig. 15. Example expert rule inheritance hierarchy for a number of telephone agents.

5. THE VOICEXML APPLICATION

As a precursor to the experimental investigation, this section describes how a functionally similar system to the Queen's Communicator (QC) can be developed using VoiceXML. It should be noted that the basic version of VoiceXML 2.0 (<http://www.w3.org/TR/voicexml20/>) was used without the additional components and objects (RDCs, Speech Objects, Voice Objects) described in Section 2.2. This is because the purpose of the evaluation is to examine issues of maintainability and re-usability in respect to additional functionalities of the dialog management component that are potentially more easily supported in QC, such as the provision of cross-domain and advanced mixed-initiative dialog. The evaluation does not address the comparison of reusable components and objects with QC's service and support agents since it is straightforward to create additional objects to match QC's agents and vice versa.

A VoiceXML version of a cinema booking and inquiry system was designed and implemented on the basis of the functional and design specifications of a system implemented using the Queen's Communicator architecture in order to provide a basis for the evaluation of the QC architecture in terms of maintainability and reusability (see Section 6). Although not all of the functionalities of the QC version of the application could be easily implemented in the VoiceXML

version, it was possible to implement sufficient elements of functionality to provide a basis for evaluation as detailed in the following.

User input. The basic idea of a dialog exchange involving a system prompt and a user response is fundamental to any spoken dialog system. These elements were straightforward to implement in VoiceXML, including recognition grammars to process the user input. These grammars are roughly equivalent to the semantic grammars used by the Phoenix parser in QC, although Phoenix provides more robust parsing for the irregular input typical of spoken language. In principle, it would be possible to construct VoiceXML grammars to handle irregular input but at the cost of large numbers of alternative rules that would ultimately lead to unacceptable recognition times.

Dialog Initiative. A major feature of QC is the ability to switch between mixed initiative and system initiative. In VoiceXML, a limited form of mixed initiative is provided in which, at the initial prompt in the dialog, the user can include any number of the required items of information in the response. However, following this, the system reverts to system initiative mode to elicit any remaining items one-by-one. For example, if there are required fields for cinema, film, day and time, and the user says the name of the film and the day, then the system will ask for the name of the cinema followed by a prompt for the time. It is possible to specify several <initial> elements in VoiceXML to allow for continued mixed initiative. However, this requires specifying all the different combination of potential user input in the grammars as well as the conditions for each mixed initiative field, whereas the architecture of QC lends itself to a more concise and manageable approach to mixed initiative dialog.

Error Handling. VoiceXML includes a number of built-in mechanisms for handling various types of events, such as noinput and nomatch as described. These have to be specified explicitly in QC. In addition, the VoiceXML application included an event to cover cases where no records were returned from the database. A similar mechanism is defined in QC. Also, as mentioned earlier, events can be inherited in VoiceXML using the mechanism of different levels of scope.

Domain Spotting. An important feature in QC is the spotting of the domain relevant to the user's inquiry, for example, hotel reservation, cinema inquiry, theater booking. In QC, the Domain Spotter is defined explicitly as an object within the Dialog Manager (see Figure 4). Moreover, the Domain Spotter incorporates various behaviors such as selecting and polling agents (see Figure 6). There is no mechanism for specifying complex objects such as this in VoiceXML so the function of domain spotting was approximated through the use of semantic tags that identified a domain by matching elements of the recognized input with a list of domains and using rules in the form of nested conditional statements to approximate the polling of agents.

Agents. Similarly, it was not possible to implement agents in the VoiceXML application as agents in QC are essentially objects within an object hierarchy (see Figure 5). For the same reason, it was not possible to specify inheritance relationships between objects. An approximation of the QC approach

was achieved using VoiceXML form templates to represent the behaviors of the agents. These forms could be reused as required within the application, similar to the invocation of agents in QC.

Frames. In QC, frames are used to encode the information about the dialog, such as the status of the information being elicited and the system's intentions relative to this information (see Table I and Table II). Basic information about the slots to be filled is encoded in the specification of field variables in VoiceXML, and the developer is not required to write code to update these values since this is handled automatically by the Form Interpretation Algorithm. However, the downside of this feature is that it is not easy to encode in a general way the more complex information and behaviors that are included in the QC frames (as shown, for example, in Figure 8 and Figure 9). Procedural code can be attached to field variables in the form of conditional statements, but these would have to be specified individually for each field.

Dialog Control. As indicated earlier, most basic dialog behaviors can be easily specified in VoiceXML without requiring the developer to write procedural code, for example, to control the dialog flow through a form. More complex dialog control is made possible in QC through the use of objects and rules (see Figure 13, Figure 14, and Figure 15). As indicated earlier, QC rules can be approximated in VoiceXML with appropriate procedural code, but this is an ad-hoc approach that violates the principle of declarative coding supported by VoiceXML, and it is not possible to provide the more general, reusable solutions available in QC.

In summary, a number of the functions for dialog control that are provided in QC can be implemented more or less directly in VoiceXML, and some others can be approximated with procedural code. In this way, VoiceXML meets the requirements of its designers in providing a platform that is easy to use for the design of simple dialog while containing features that also support more complex interactions. However, as indicated earlier, some of the more sophisticated elements of QC that support more advanced dialogs cannot be implemented easily in VoiceXML if at all, and, even so, the methods used are ad hoc and lacking in generality. Nevertheless, there is sufficient similarity between both versions of the system to enable the experimental evaluation of the two platforms in the following sections.

6. EXPERIMENTAL EVALUATION

6.1 Experimental Setup

The IEEE Standard for Software Maintenance [IEEE 1993] defines maintenance as the modification of “a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment”. In turn, maintainability can be defined in relation to the associated cost with which a software artefact can be extended, adapted, or corrected. Cost, in this context, can be described in terms of the amount of effort required to plan, implement, and monitor the maintenance, alongside the degree of structural decay associated with the maintenance [IEEE 1993]. In other words, the cost of maintainability can be expressed in terms of how easy

change can be introduced and to what extent the change is likely to deteriorate the design in a manner that makes future maintenance more problematic.

A number of different types of metric have been proposed as providing an estimate of the likely cost of maintaining a software artifact. Broadly, the metrics attempt to formally encapsulate measurements related to complexity, comprehensibility, reusability, and adaptability. Examples of different approaches range from simple source metrics, such as the number of lines of code, to more complex assessments of algorithmic complexity (e.g., Halstead [1977], McCabe and Butler [1989], and Welker and Oman [1995]), and to those exploring class and package-level complexity (e.g., Chidamber and Kemerer [1994], Martin [2003]). In relation to architectural design, the use of metrics tends to be highly dependent on the context and form of predicted change, that is, an evaluation of the cost of architectural change must also consider the predicted likelihood that such a change will occur. A number of different scenario-based approaches for evaluating maintainability have been proposed (e.g., Bosch [2000], Kazman et al. [1996], Kazman et al. [1998]).

In order to evaluate the maintainability of the dialog manager architecture, it was decided that a scenario-based approach would be used, based on the following forms of maintenance: modification of existing domain-specific expertise, extension and reuse of existing domain-specific expertise, and modification of discourse management behavior. In order to provide a measurement of the cost associated with a particular form of change to the discourse manager, the following metrics are proposed as providing an adequate means of estimating the overall effort:

- measurement of change to domain-specific expertise*: the number of new/modified expert rules and the associated number of new/modified expert objects;
- measurement of change to discourse management behavior*: the number of new/modified discourse manager parameters/rules;
- where change cannot be fully measured in terms of the previous metrics*: the approximate number of additional lines of Java code (LOC).

Similarly, suitable VoiceXML-based metrics had to be derived that could also provide an adequate means of estimating the overall effort so that a comparison could be made between the two systems. The criteria to be used for measuring the functionally similar VoiceXML metrics are:

- measurement of change to domain-specific expertise*: the number of new and modified fields within a dialog form;
- measurement of change to discourse management behavior*: unlike the QC and related systems mentioned in Section 2, VoiceXML does not separate the domain management and the discourse management expertise. As mentioned before, the FIA implements dialog control in a fairly rigid and inflexible sequential manner so discourse management is usually controlled by utilizing conditional statements held within the VoiceXML `<filled>` elements which execute code once a value has been elicited. Likewise, `<filled>` is also used to control the domain expertise within a VoiceXML system along with the

Fig. 16. Queen's Communicator expert objects supporting a cinema booking inquiry.

<block> element which can also contain conditional statements. Therefore a separate metric cannot be derived solely for discourse management, and the number of new and modified fields within a dialog form is once again appropriate;

- *where change cannot be fully measured in terms of the previous metrics*: as the number of fields added or amended can be quite a crude figure, additional metric lines of code (LOC) will once again be provided which is an approximation of the number of lines of both VoiceXML and PHP code which have been added or altered within the system.

6.2 Base Scenario

A cinema-booking inquiry was selected to provide the base scenario on which various forms of maintenance could be explored. In particular, a dialog system was constructed that would collect information concerning the show (film name, cinema name, show date and time, number of tickets, ticket price) and related booking information (credit card number and expiration date, contact name, address, and telephone number). The base scenario assumes that the dialog management process will employ grounding and confirmation strategies throughout the discourse, for example, using the database to confirm a film name as soon as the name is provided by the user. Additionally, where possible and appropriate, information supplied by the user should be used to constrain the remaining items of information that need to be collected, for instance, if the user has supplied a film and cinema name, then only valid show times should be presented to the user. Finally, the user should be able to change previously given items of information. An overview of the agents introduced to support the outlined cinema booking transaction is shown in Figure 16.

An overview of the structure of an equivalent VoiceXML is shown in Figure 17. Figure 17 shows the dialog represented as VoiceXML, and the fields within each form are illustrated to the right of the forms. These fields must be elicited in either a mixed or system initiative fashion before progressing onto the next form. The forms themselves are designed to be encapsulated, so that they can be reused in future dialog systems that require the particular functionality. Additionally, during the booking dialog, there is an additional validation process made up mainly of PHP scripts that confirms that the elicited

Fig. 17. VoiceXML setup supporting cinema booking inquiry.

film/cinema/date/showtime combination is valid. When system initiative is utilized, this is a step-by-step process where the value elicited by the user is used first to derived a possible subset of allowed values based upon that input for example, if the user elicits “Movie House Belfast” then only available movies showing at that particular cinema will be included in the dynamic grammar for the following question.

6.3 Maintenance Scenarios

A number of maintenance scenarios are outlined in the following, alongside an analysis of the associated cost of introduction assuming the Queen’s Communicator and VoiceXML baseline systems outlined Figure 16 and Figure 17 respectively (see Tables IV, V, VI).

7. ANALYSIS

As outlined in the IEEE Standard for Software Maintenance, maintainability can be measured in terms of the amount of effort required to plan, implement, and monitor the maintenance, together with the degree of structural decay associated with the maintenance [IEEE 1993].

7.1 Cost of Planning and Implementation

With regard to the amount of effort required to plan and implement change, it is the assessment of the authors that the overall cost is similar between

Table IV(A). Modification of Existing Domain-Specific Expertise

Fictional Case Study 1				
Analysis has shown that in most cases the user wishes to see the film on the same day as their booking. The system should be changed to assume the current date as the desired booking date. The user should be informed of this assumption and permitted to change the date if desired.				
Queen's Communicator cost analysis				
A single additional user focussed rule is required to prepopulate the date slot upon initial commencement of the cinema expert. The inherited discourse management functionality will implicitly confirm the selection and permit the user to change the slot value if desired.				
Expert Rules		Expert Objects		Extra LOC
New	Modified	New	Modified	
1	0	0	1	
VoiceXML cost analysis				
One field is added to confirm that the user wishes to use today's date as default. If 'yes', the booking date is set to current, otherwise booking date is left undefined, so that the user can specify this during the dialog.				
Fields			Extra LOC	
New	Modified			
1	0			

(B).

Fictional Case Study 2

As a means of increasing maximum seating levels the fictional Budget cinema chain is to introduce two-tiered theaters. Bookings to Budget cinemas retain the same characteristics as normal cinema bookings but should additionally enquire if the user wishes to book seats on the upper or lower tier.

Queen's Communicator cost analysis

A new BudgetCinemaExpert, extending CinemaExpert, will need to be introduced. The expert will inherit all previously defined cinema expertise. New rules will need to be added to obtain/suggest/confirm the seating tier. Additionally, some database back-off rules will need to be modified to take into account the unavailability of a requested tier. A new rule will need to be added into CinemaExpert to evolve a basic cinema inquiry into a Budget cinema inquiry if the user selects a Budget cinema.

Expert Rules		Expert Objects		Extra LOC
New	Modified	New	Modified	
3	3	1	1	0

VoiceXML cost analysis

Two new fields are required, first to prompt the user if a budget cinema is requested, and if so, then whether the lower or upper tier is requested. The selection is then added to the global variables, and later, if a budget cinema was requested, the tier choice is added to the database with the payment details. Additional PHP scripting is also required to verify that there is suitable seating available on the selected tier, and if not, to suggest the alternative tier to the user.

Fields		Extra LOC
New	Modified	
3	0	~70

the VoiceXML and Queen's Communicator developments with two notable exceptions. In both systems identifying and planning the necessary change was unproblematic and mostly consisted of selecting new rules, fields, and experts. As the case studies are fictitious, no attempt was made to measure the cost of monitoring and assessing the impact of the maintenance.

Cost within the VoiceXML system can be broadly described in terms of the effort required to design new fields and associated VoiceXML/PHP scripts and the effort required to integrate the new elements into the existing forms. Within the Queen's Communicator, cost can be broadly described in terms of the effort required to introduce new rules and the effort required to package changes into new/existing expert objects.

Case Study 5 (Table VI) illustrates a situation where the cost of planning and implementation differ significantly between the two systems. In particular, as indicated in Section 5, it is not readily possible to employ inheritable dialog control strategies across VoiceXML forms. This entails that modification of the underlying dialog behavior is likely to require change across all applicable forms. Within the Queen's Communicator, as the discourse control is inherited from the discourse manager, any change in behavior will be inherited by all extending experts.

Notable variation between the systems was also observed in Case Study 4 (Table V(B)). As can be seen, the ability to inherit generalized functionality and reuse existing expert objects ensures that the cost of introduction remains low, both in terms of effort and the number of new rules, etc. that need to be added to the system. While the VoiceXML system was able to reuse some existing fields, the overall cost was higher. In particular, the amount of new code that was required, and hence the amount of additional code that will need to be maintained in the future, was significantly higher.

It is important to note that the base scenario and case studies used in the comparison were of modest complexity and size. As such, the corresponding cost of change is likely to be similarly modest. However, it is reasonable to expect that as the complexity of the base system and change requests increase, then so will the cost of maintenance in terms of planning and implementation. Based on patterns of maintainability within traditional forms of software engineering, it may be the case that as the complexity of the dialog system increases, then the advantages of encapsulating behavior into discrete expert objects will result in a significantly lower comparative cost of planning and implementing change.

7.2 Structural Decay

There is a notable difference between the Queen's Communicator and VoiceXML systems in terms of the degree of structural decay following the implementation of changes associated with a number of the case studies (most notably, Case Studies 2 (Table IV(B)), 3 (Table V(A)) and 5 (Table VI)).

The majority of change within the Queen's Communicator was introduced through the development and addition of new expert objects which typically extended an existing component. As such, changes tend to be encapsulated within defined expert objects and have limited/no impact on other experts, that is existing expert objects tend not to be subjected to change or decay. Likewise, established forms of interdependence between expert objects are not altered. Introducing extending experts that refine and specialize existing experts enables reuse scenarios to employ whichever expert fits best, that is, selecting an expert that matches the level of specialization appropriate to the form of reuse.

Table V(A). Extension and Reuse of Existing Domain-Specific Expertise

Fictional Case Study 3				
A new form of payment is to be introduced, called reserve-and-collect. The user will be provided with a reservation number and be required to pay upon collection. User contact information will continue to be collected. This form of payment is to be offered alongside credit card payment.				
Queen's Communicator cost analysis				
A new expert, extending PaymentExpert, will need to be added. The expert will inherit rules that instigate the collection of contact information, that is the payment expert needs to define three rules: one to provide a booking reference and another two to repeat the reference if needed. A new expert rule action will be needed to generate a booking reference (this will require Java coding). The expert selection process will ensure that the new payment expert will be automatically selected as appropriate.				
Expert Rules		Expert Objects		Extra LOC
New	Modified	New	Modified	
3	0	1	0	~100
VoiceXML cost analysis				
A field is added before the commencement of the credit card dialog. The field prompts the user to continue with the credit card payment now, or to reserve the tickets and pay later. Either the dialog will continue or the user's details will be passed to the Web server where existing PHP scripts will store the details in the database.				
Fields				Extra LOC
New		Modified		
1		0		24

(B).

Fictional Case Study 4				
A new form of concert ticket booking transaction is to be introduced. Users will be asked to specify their intended venue, show, date and given the option to purchase tickets using only their credit card. Where appropriate and possible, discourse expertise developed for the cinema booking transaction should be reused.				
Queen's Communicator cost analysis				
A new form of inquiry expert, extending the event expert will need to be introduced. General event handling expertise is provided through inheritance. As such, a total of 10 new user-focused rules and 4 database rules tailored to concert booking will need to be added to the concert expert. Credit card payment facilities can reuse the payment experts and associated support experts created for the cinema expert. One additional rule will need to be added to the concert expert to explicitly transfer control to the credit card expert as the only form of supported payment.				
Expert Rules		Expert Objects		Extra LOC
New	Modified	New	Modified	
15	0	1	0	
VoiceXML cost analysis				
New dialog forms are required, one to handle mixed initiative allowing the system to elicit the venue, show and date, and others to provide a refined subset from the database based upon previous utterances. Approximately 700 LOC are required to achieve this, including database rules and logic. As previous code is specific to cinema forms, fields and database tables, it cannot be reused. Certain parts of the cinema dialog, such as requesting if a user is a first-time user or whether he/she is using credit card payments, can be reused.				
Fields				Extra LOC
New		Modified		
16		0		
				~ 700

Table VI. Modification of Discourse Management Behavior

Fictional Case Study 5				
Upon session commencement, the user should be asked if they have previously used the system. If not, then all information supplied by the user will be explicitly confirmed with a maximum of one item confirmed each dialog turn. Otherwise, the system will adopt a default unrestricted implicit confirmation strategy.				
Queen's Communicator cost analysis				
Two expert rules will need to be added to ask if this is a new user and act accordingly. New code will need to be added to track if the user is new. A discourse rule will need to be updated to use an explicit confirmation action given a new user (otherwise defaulting to implicit) and two discourse parameters will be needed to track the maximum number of slots to implicitly/explicitly confirm each dialog turn.				
Expert Rules		Expert Objects		Extra LOC
New	Modified	New	Modified	
2	1	1 (rule) 1(parm)	1 (parm)	~ 10
VoiceXML cost analysis				
One field is added at the start to prompt the user if they are a new or existing user. Confirm fields are then added throughout the dialog to explicitly confirm each uttered detail if they are a new user.				
Fields				Extra LOC
New		Modified		
18		4		~ 170

The use of centralized interagent control, via the DomainSpotter object also provides a mechanism where coupling between expert objects remains weak and new forms of experts can be incorporated into the system without impacting upon existing experts, that is, new forms of agents can be dropped into the system and their functionality automatically offered and selected as appropriate in response to other agent requests.

Within the VoiceXML system, changes tend to be integrated into existing forms and consequentially change or enrich the functionality or purpose of the form. However, such a type of change is likely to have an impact on other components that currently depend on the form or, alternatively, intend to make use of the functionality offered by the form. For example, in Case Study 3 (Table V(A)), the incorporation of a reserve-and-collect method of payment into the payment form will ensure that any reuse of the payment form in other types of dialog application will need to consider if this method of payment is appropriate and, if necessary, an adapted form without a reserve-and-collect method of payment may have to be maintained. The payment form may in effect become less reusable following the introduction of the new functionality.

More generally, the use of an object-based approach, employing strong encapsulation and the ability to inherit and specialize functionality, provides both a ready means of isolating change and a mechanism for component reuse. The more procedural and form-based operation of systems such as VoiceXML, while offering mechanisms for reuse, do not lend themselves so strongly to easy and effective reuse. However, with specific regard to VoiceXML, this difference is likely to become less marked as reusable VoiceXML components, which currently encapsulate individual areas of expertise, become organized into hierarchies of interrelated objects. In summary, over a period of time and system

change, it is likely that object-based dialog systems will offer strong resilience to forms of structural decay. To a good extent, this suggestion mirrors the relationship found within the broader software engineering domain concerning the advantages of an object-oriented development methodology.

8. CONCLUSIONS

This article has provided details of how an object-oriented approach can be applied to the architectural design of a dialog management component with the aim of facilitating the modification, extension, and reuse of discourse-related expertise.

A functionally similar VoiceXML system was constructed and used as a baseline against which to compare the maintenance cost of a number of fictional case studies. The cost of planning and implementing change was broadly similar between the two systems except for those case studies involving either reuse of existing discourse expertise or a change of dialog control strategy. In these case studies, the encapsulation of expertise into reusable expert objects and the inheritability of dialog control strategies resulted in a lower cost of implementation within the Queen's Communicator.

The case studies also demonstrated significant differences between the two systems in terms of the degree of structural decay following implementation of change associated with the case studies. The ability to extend existing expert objects, inheriting their defined functionality and encapsulating change within more specialized experts, typically ensured that existing expert objects were not subjected to structural decay. Additionally, the introduction of specialized experts enabled reuse scenarios to employ whichever expert object offered the best match to the required level of generality or speciality. Finally, the use of weak coupling between objects, with a domain spotter providing centralized interagent control, enabled new agents to be added to the system and their functionality automatically offered to existing agents. While the case scenarios could be successfully implemented in VoiceXML, in its current form, the implementation was such that the overall cost of future maintenance/reuse would be greater. The use of reusable components in VoiceXML and the possibility of organizing these components into object hierarchies would help to address this issue. Furthermore, while the case studies focused on scenarios that involved the implementation of change to an existing system, the authors would argue that the architectural benefits relating to agent reuse would also be realizable within scenarios involving the reuse of cross-domain expertise for the development of new types of inquiry or transaction.

More generally, this article has shown how the generalized benefits of an object-oriented design methodology can be realized within the design and implementation of a dialog management component in terms of improved maintainability, extensibility, and reusability. This is likely to be of significance within dialog systems where the majority of effort will occur after the initial development, for example, within research systems intended to provide a platform on which to explore new approaches or within commercial systems where it may be desirable to reuse and/or extend an existing solution. Furthermore, while

the dialog system outlined within this article employed a knowledge-based approach to dialog management, the authors would argue that the advantages afforded through an object-oriented design methodology should be realizable in the development of data-driven dialog systems in terms of providing a ready means of reusing, refining, and specializing datasets.

In conclusion, through appropriate design, an object-oriented dialog manager can provide a means of limiting the degree of structural decay as the system evolves and changes over time. Additionally, an object-oriented approach provides a means of readily reusing existing discourse expertise. As such, the advantages provided by an object-oriented approach are likely to prolong the useable lifespan of a dialog manager.

REFERENCES

- ALLEN, J., BRYON, D., DZIKOVSKA, M., FERGUSON, G., GALESCU, L., AND STENT, A. 2000. An architecture for a generic dialog shell. *J. Nat. Lang. Engin.* 6, 3, 1–16.
- BANGALORE, S., HAKKANI-TUR, D. AND TUR, G. 2006. *Speech Comm.* (Special Issue on Spoken Language Understanding in Conversational Systems) 48, 3–4, 233–462.
- BENNETT, K., GOLD, N., AND MOHAN, A. 2005. Cut the biggest IT cost. *ITNOW* 47, 1, 20–21.
- BOHUS, D. AND RUDNICKY, A. 2003. RavenClaw: Dialog management using hierarchical task decomposition and an expectation agenda. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, Geneva, Switzerland. 597–600
- BOOCH, G. 1993. *Object Oriented Analysis and Design with Applications*, 2nd Ed. Benjamin-Cummings Publishing Co., Inc. Redwood City, CA.
- BOSCH, J. 2000. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison Wesley Longman, Reading, MA.
- CHIDAMBER, S. AND KEMERER, C. 1994. A metrics suite for object oriented design. *IEEE Trans. Softw. Engin.* 20, 6, 476–493.
- COLE, R. 1999. Tools for research and education in speech science. In *Proceedings of the International Conference of Phonetic Sciences (ICPhS)*, San Francisco, CA. 1277–1280.
- COMMUNICATOR. 2007. Galaxy Communicator. <http://communicator.sourceforge.net/>.
- DE CAROLIS, B., PELACHAUD, C., POGGI, I., AND STEEDMAN, M. 2003. APMML: A markup language for believable behaviour generation. In *Life-like Characters. Tools, Affective Functions and Applications*, Prendinger, H. and Ishizuka, Eds, Springer, Berlin.
- GAMMA, E., HELM, R., JOHNSON, R., AND VISSIDES, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Reading, MA.
- GUPTA, N., TUR, G., HAKKANI-TUR, D., BANGALORE, S., RICCARDI, G., AND GILBERT, M. 2006. The AT&T spoken language understanding system. *IEEE Trans. Speech Audio Process.* 14, 1, 213–222.
- HALSTEAD, M. 1977. *Elements of Software Science*. Elsevier Science Inc. New York, NY.
- HAPTEK 2007. Haptek. <http://www.haptek.com/>.
- HUMAINE WP8. 2007. Humaine WP8 Emotion in Communication. <http://emotion-research.net/ws/wp8/proceedings-wswp8.pdf>.
- IEEE. 1993. IEEE Standard for Software Maintenance, STD 1219-1993, Institute of Electrical and Electronics Engineers, Inc.
- JAKARTA 2007. The Jakarta Project. <http://jakarta.apache.org/taglibs/doc/rdc-doc/intro.html>.
- JASPIS. 2007. Jaspis. <http://www.cs.uta.fi/research/hci/spi/Jaspis/>.
- KAZMAN, R., ABOWD, G., BASS, L., AND CLEMENTS, P. 1996. Scenario-based analysis of software architecture, *IEEE Softw.* 13, 6, 44–55.
- KAZMAN, R., KLEIN, M., BARBACCI, M., LONGSTAFF, T., LIPSON, H., AND CARRIERE, J. 1998. The architecture trade-off analysis method. In *Proceedings of the 4th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE Computer Society Press, 68–78.
- LEVIN, E., PIERACCINI, R., AND ECKERT, W. 2000. A stochastic model of human-machine interaction for learning dialog strategies. *IEEE Trans. Speech Audio Process.* 8, 1, 11–23.

- MARTIN, R. 2003. *Agile Software Development: Principles, Patterns and Practices*. Prentice Hall, Upper Saddle River, NJ.
- MEYER, B. 1997. *Object-oriented Software Construction* 2nd Ed. Prentice-Hall, Inc., Upper Saddle River, NJ.
- MCCABE, T. AND BUTLER, C. 1989. Design complexity measurement and testing. *Commun. ACM* 32, 1415–1425.
- MCTEAR, M. 2002. Spoken dialog technology: Enabling the conversational user interface. *ACM Comput. Surv.* 34, 1, 90–169.
- MCTEAR, M. O'NEILL, I. HANNA, P., AND LIU, X. 2005. Handling errors and determining confirmation strategies—An object-based approach. *Speech Comm.* 45, 3, 249–269.
- NAACL-HLT. 2007. Workshop on bridging the gap: Academic and industrial research in dialog technologies. <http://www.cs.rochester.edu/meetings/hlt-naacl07/workshops/dialog.html>.
- NUANCE. 2007. W3C SpeechObjects Specification V1.0. <http://www.w3.org/TR/speechobjects/>.
- O'NEILL, I., HANNA, P., LIU, X., AND MCTEAR, M. 2003. The Queen's Communicator: An object-oriented dialog manager. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, Geneva, Switzerland. 593–596.
- O'NEILL, I., HANNA, P., LIU, X., GREER, D., AND MCTEAR, M. 2005. Implementing advanced spoken dialog management in Java. *Sci. Comput. Program.* 54, 1, 99–124.
- PARUNAK, H., BRUECKNER, S., FLEISCHER, M., AND ODELL, J. 2003. A preliminary taxonomy of multi-agent interactions. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Melbourne, Australia. 1090–1091.
- PELLOM, B., WARD, W., AND PRADHAN, S. 2000. The CU communicator: An architecture for dialog systems. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Beijing, China. 9–16.
- PHOENIX. 2007. The Phoenix System. <http://cslr.colorado.edu/~whw/Phoenix.htm>.
- PRESSMAN, R. 2004. *Software Engineering: A Practitioner's Approach* 6th Ed. McGraw Hill, New York, NY.
- YOUNG, S. 2002. Talking to machines (statistically speaking). In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Denver, CO. 9–16.
- TALK. 2007. Talk and look: Tools for ambient linguistic knowledge. <http://www.talk-project.org/>.
- SENEFF, S. 1992. TINA: A natural language system for spoken dialog applications. *Comput. Linguis.* 18, 1, 61–86.
- TRAINS. 2007. The TRAINS Project: Natural spoken dialog and interactive planning. <http://www.cs.rochester.edu/research/trains/>.
- TRINDI. 2007. Trindi. <http://www.ling.gu.se/projekt/trindi/>.
- TRINDIKIT. 2007. TrindiKit. <http://www.ling.gu.se/projekt/trindi/trindikit/>.
- TURUNEN, M. AND HAKULINEN, J. 2003. Jaspis2—An architecture for supporting distributed spoken dialogs. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, Geneva, Switzerland. 1913–1916.
- VERBMOBIL. 2007. Verbmobil. <http://verbmobil.dfki.de/>.
- VOICEOBJECTS. 2007. VoiceObjects. <http://www.voiceobjects.com/>.
- VOICEXML FORUM. 2007. The VoiceXML Forum. <http://www.voicexml.org>.
- VOICEXML FORUM DIALOG. 2007. Workshop on advanced dialogs. <http://www.chicagodavid.com/dialog/index.htm>.
- VOICEXML FORUM TOOLS. 2007. The Advanced Dialog Working Group (part of the VoiceXML Forum Tools Committee). <http://www.chicagodavid.com/dialog/ADgroup.htm>.
- WALKER, M. 2000. An application of reinforcement learning to dialog strategy selection in a spoken dialog system for email. *AI Resea.* 12, 387–416.
- WELKER, K. AND OMAN, P. 1995. Software maintainability metrics models in practice. *J. Defense Softw. Engin.* 8, 11, 19–23.

Received October 2006; revised March 2007; accepted April 2007 by Marcello Federico