international journal of medical informatics xxx (2008) xxx–xxx

journal homepage: www.intl.elsevierhealth.com/journals/ijmi

Adaptable dialog architecture and runtime engine (AdaRTE): A framework for rapid prototyping of health dialog systems

*L.M. Rojas-Barahona**, *T. Giorgino*

Laboratory for Biomedical Informatics, Dipartimento di Informatica e Sistemistica, Università di Pavia, Via Ferrata 1, I-27100 Pavia, Italy

article info

Article history:
Received 20 February 2008
Received in revised form
18 July 2008
Accepted 30 July 2008

Keywords:
Telemedicine
Speech recognition software
Ambulatory care information systems
Telephone
Reminder systems
Medical informatics
Computer

abstract

Spoken dialog systems have been increasingly employed to provide ubiquitous access via telephone to information and services for the non-Internet-connected public. They have been successfully applied in the health care context; however, speech technology requires a considerable development investment. The advent of VoiceXML reduced the proliferation of incompatible dialog formalisms, at the expense of adding even more complexity. This paper introduces a novel architecture for dialogue representation and interpretation, AdaRTE, which allows developers to lay out dialog interactions through a high-level formalism, offering both declarative and procedural features. AdaRTE's aim is to provide a ground for deploying complex and adaptable dialogs whilst allowing experimentation and incremental adoption of innovative speech technologies. It enhances augmented transition networks with dynamic behavior, and drives multiple back-end realizers, including VoiceXML. It has been especially targeted to the health care context, because of the great scale and the need for reducing the barrier to a widespread adoption of dialog systems.

1. Introduction

Automated dialog systems are widely used to provide the public with access to a set of automated services. The opportunities offered by computer-based conversations can be reaped for telemedicine applications, e.g. offering patients self-paced access to an ever-increasing fraction of clinical information. Patients can take the role of either information consumers (e.g. for receiving counsel and education), producers (interviews, long-term monitoring of chronic diseases, symptom reporting, etc.) or both (e.g. as a support to the scheduling of clinical exams and receiving therapy updates).

Adopting dialog systems in the medical domain, however, is especially complex. First, dialogs in the health care context should be designed to maintain a continuous relation with patients through time. Some dialogs have the objective of eliciting changes in the patient's behaviors or habits. Criticality is also present in dialogs used for chronic symptoms monitoring [1]. Last but not least, clinical practices enact complex guidelines, ontologies and procedures [2], which increment the complexity of automated dialogs. A number of these clinical practices in patients' home care require professional assistance to be successfully fulfilled. At the same time, it is simply unfeasible to have the whole medical personnel available to cover patients' demands and, perhaps, this is the main motivation towards the adoption of automatic dialog systems in medicine. Another important motivation is to increase the availability of cost-effective monitoring in disadvantaged geographical regions.

* *Corresponding author*. Tel.: +39 0382 985981; fax: +39 0382 526088.
E-mail address: linamaria.rojas@unipv.it (L.M. Rojas-Barahona).

1386-5056/$ – see front matter © 2008 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.ijmedinf.2008.07.017

This paper presents the AdaRTE framework, devised in order to overcome the issues of the existing dialog management methods. Our interest is mainly focused on health care dialog systems, and therefore our solution is especially targeted at offering rapid prototyping, standards-compliant deployment and experimentation through the incremental integration of other voice formalisms, e.g. NLP based on lexicalized grammars.

AdaRTE's features were designed to reduce the burden of dialog development through reuse, support of augmented transition networks, adaptable decision points and adoption of best practices. This paper shows how AdaRTE implements these features for dialog deployment, and presents the results obtained by prototyping five medical telephony-linked systems. Three of them were inspired by earlier working systems, namely the chronic obstructive pulmonary disease (COPD) care [3], the Homey dialog system for hypertensive patient home management [4], and for diabetes care [5]. Two more dialogs were implemented from scratch to assist patients receiving peritoneal dialysis and oral anticoagulation therapy (OAT); the latter is being prepared for use at the Mondino Neurological Hospital of Pavia (Italy).

2. Background

Computer-based dialog systems have been proven useful for providing the general public with access to telemedicine services. Several studies have discussed their advantages for chronic symptoms monitoring, interviews, counseling and education. Piette [6], Corkrey and Parkinson [7], Krishna et al. [8], Kaplan et al. [9], and Edmonds et al. [10] reviewed interactive voice response systems (IVR) that have been used for interviews, alcohol or drug-abuse support, hypertension monitoring and others, emphasizing the benefits of these systems such as the ability to be in continuous operation whilst offering confidentiality. Migneault et al. [11] summarizes the experience in building over twenty IVR systems for various health purposes focusing on positively influencing patients' health behavior and disease monitoring such as angina pectoris, chronic obstructive pulmonary disease, asthma and others. Several interventions have been reportedly able to improve the quality of service and communication in a cost-effective way. Also, dialogs supporting automated speech recognition (ASR) have been used successfully for health purposes. For instance, a dialog for the management of hypertensive patients supporting high adaptability and restricted mixed initiative is described in [4]. More recently, Levin and Levin presented a system for chronic pain monitoring, tested on volunteers, that profiles users upon their experience, offering varying amounts of help and fallbacks in the case of misunderstanding by the ASR component [12]. Further discussion on the adoption of dialog systems for health communication can be found in a special issue [13].

Despite the growing efforts towards developing user-friendly IVR systems, these kinds of telemedicine interventions have been criticized for their lack of flexibility, mainly due to the fact that, to be effective, they greatly restrict the interaction with the user. This is mostly due to technical reasons: touch-tone-based systems are limited to numeric input and menu-like navigation. On the other hand, speech-based systems have emerged as a very promising solution thanks to the extensive efforts pursued by the speech recognition community. Nevertheless, speech-based systems are still error-prone and their language-understanding component does not cover the whole variety of linguistic phenomena [14].

2.1. *Theoretical approaches*

Attempts towards more sophisticated approaches to dialog modeling in the medical domain have been pursued in Refs. [15,16]. In the former, Allen et al. present a medication advisor "Chester", in which the dialog is seen as a collaborative system where agents work together in order to achieve a common goal. Chester embraces the generic architecture developed for the Rochester Interactive Planning System (TRIPS) [17], which clearly separates domain and dialog representations. It uses a stochastic plan and an intention recognizer in order to infer the user's intention and supports mixed initiatives. In [16], Beveridge and Fox present a decision support dialog system for advising physicians about whether or not a patient should be referred to a cancer specialist. The dialog follows the conversational games theory of dialog modeling, first introduced by Power [18]. Its architecture separates dialog and domain representation and, in particular, the domain and plan representation use ontologies and guidelines, respectively.

In spite of the remarkable endeavors, a considerable effort should still be made in order to solve issues related to language, pronunciation modeling of medical terminology, limited information in knowledge bases, computational costs for reasoning and adaptability. This is even more evident when considering the perspective of implementing robust, functional and complete dialog systems for *real* situations in the medical domain.

A range of approaches is available for dialog modeling. According to the classification made by Allen et al. [19] ordered by increasing complexity, the simplest of these is *finite-state scripts*, also called *dialog grammar*, followed by *frame-based*, *sets of context*, *plan-based* and *agent-based models*. In a *finite-state* script, the dialog is represented as a script of prompts for the user. In *frame-based* systems, questions are asked in order to enable the system to fill slots of entity requirements to perform a task. *Sets of contexts* describe the dialog task and each context is represented using the frame-based technique. Conversely, *plan-based theories* claim that utterances infer acts that are part of a plan; thus, the system should identify the user's underlying plan and respond appropriately [20]. *Agent-based models* are at the highest level of complexity. They consider planning, executing and monitoring operations in a dynamically changing world, possibly involving multimodality. An additional approach, *the conversational games theory*, is presented in [16,18,21]. This approach models task-oriented dialogs and uses techniques from both frame-based and plan-based models. This approach provides a method of modeling mixed-initiative and complex dialogs.

Generally, many of the aforesaid approaches require heavily coded solutions and are not readily suited for small-scale applications in a real-world setting. Additionally, there is not much information available about the time and costs implied

Please cite this article in press as: L.M. Rojas-Barahona, T. Giorgino, Adaptable dialog architecture and runtime engine (AdaRTE): A framework for rapid prototyping of health dialog systems, Int. J. Med. Inform. (2008), doi:10.1016/j.ijmedinf.2008.07.017

in the dialog systems' development process. As a matter of fact, the process of deployment of dialog systems has been considered complicated, costly and time demanding and usually requires speech technology experts. Several toolkits were devised in order to simplify the programming burden; among them, the best-known is probably TrindiKit [22]. Although TrindiKit allows the implementation of dialogs following the theoretical approach of the information state, it is complex to use and requires the proprietary language SICStus Prolog.

2.2. *Standardization of technology*

Recently, the WWW Consortium introduced the web-based "voice browser" (VB) activity [23]. VBs are built around a dialog manager, which retrieves documents over the web and interprets them. Its delivery reduced the proliferation of incompatible dialog formalisms by offering a reference model for voice applications. Dialog systems built with VoiceXML have been published lately, e.g. for the home monitoring of diabetic patients [24]. In spite of the aforesaid advantages, VoiceXML has inherent limitations which are well analyzed in [25,26]. For instance, its structure is declarative and static and it lacks means for efficient and heavy computation, so it is difficult to access remote resources (e.g. databases and ontologies). Also, its support of mixed-initiative interaction is limited. Furthermore, due to the web-based paradigm, VoiceXML documents themselves have to be generated dynamically by other code, which complicates application maintenance. Finally, the strongest limit pointed out by the research community is that neither dynamic natural language understanding, generation nor multimodality are directly supported.

3. Motivation

Early methods for describing the detailed structure of computerized voice interactions followed either the *finite-state* or the *frame-based* approach. Technically, they were implemented either in native computer code, or with custom dialog managers. In the former, the interaction was specified and hard-coded in a custom program, step by step, in a native computer programming language, for example C. Such a program activates speech and telephony-related features on demand according to the point of the dialog, by means of platform-specific application–program interfaces (API), which depend on the particular hardware and software combination. This approach is familiar to computer programmers and allows fine control over all aspects of the interaction. However, it ties implementations to the specific software or hardware vendors, is not portable and puts the dialog flow design mainly in the hands of programmers, rather than domain experts.

3.1. *Context-based approaches*

The fact that most of the telephone interaction boils down to a few steps of basic types, e.g. playing prompts, listening to answers, storing results in variables, and so forth, has motivated dialog engines to be built around *context-based* approaches [27,28]. The definition of "context" differed among platforms, but usually it was characterized by a group of properties, such as: prompt, grammars, variables to hold the result, help and pointer to the next context.

Although this formalism simplifies dialog layout by decomposing it into basic building blocks and provides a higher level of abstraction than low-level computer languages, it still has the disadvantages of being proprietary and strongly tied to the specific dialog manager. Furthermore, it generally lacks the flexibility required for handling special, application-specific cases, e.g. to compute prompts adaptively on the basis of user experience or other factors, to enable or disable confirmation questions, and generally it is difficult to implement strategies beyond those supplied with the platform as built-ins. These refinements tend to be important for a smooth user experience, but often can be expressed readily only through procedural programming (e.g. to keep track of counters and sensible defaults, perform non-trivial string analysis, hold complex states in an object-oriented fashion, and so on). Procedural and object-oriented constructs conflict with the *static declarative* structure of context-based dialog formalisms.

3.2. *Conforming to industry standards*

VoiceXML has been a first step towards a *combined procedural and declarative* approach, because it has foreseen both a form-filling mechanism and a procedural interpreter, in the form of the standard ECMAScript standard language [29]; context variables and scripts belong to the same namespace. In addition, ECMAScript is a powerful general-purpose language, but its usefulness is severely limited by being confined to the client-side. Therefore, this interesting approach is limited by the web-based paradigm, because the document generation and exchange have to be meta-programmed and executed at runtime, and the procedural interpreter is restricted to the browser. Both the high-level dialog management and the resources access take place on the web server, where they have to be programmed with the less-than-straightforward mechanisms like Java Server Pages. Again, this paradigm may be familiar to programmers, but is often outside of the grasp of domain experts for dialog writing or maintenance. For the reasons discussed in the previous section, VoiceXML has a negative impact on the ability to produce telemedicine applications efficiently.

As a consequence, a variety of extensions to VoiceXML have been proposed: for instance, DialogXML (applied to car telematics) extended the VB in order to support NLP KANTOO generated grammars [30]. A prototype of an editor for creating VoiceXML documents is exposed in [31]. Other VoiceXML generative approaches are presented in [32], which follows a database-oriented approach, and in [33], which is seemingly targeted towards customer care tasks with sophisticated call routing. We believe that a big effort should still be made in adapting dialog systems' best practices [34], such as confirmation strategies, adaptability, mixed initiative into VoiceXML-based frameworks, providing usable speech interfaces to users and graphical interfaces to developers.

We devised a dialog representation that overcomes these limits. This work has been motivated by two main factors: (1) reduce the time required to deploy dialog applications, and enable subjects who are not programmers to develop and

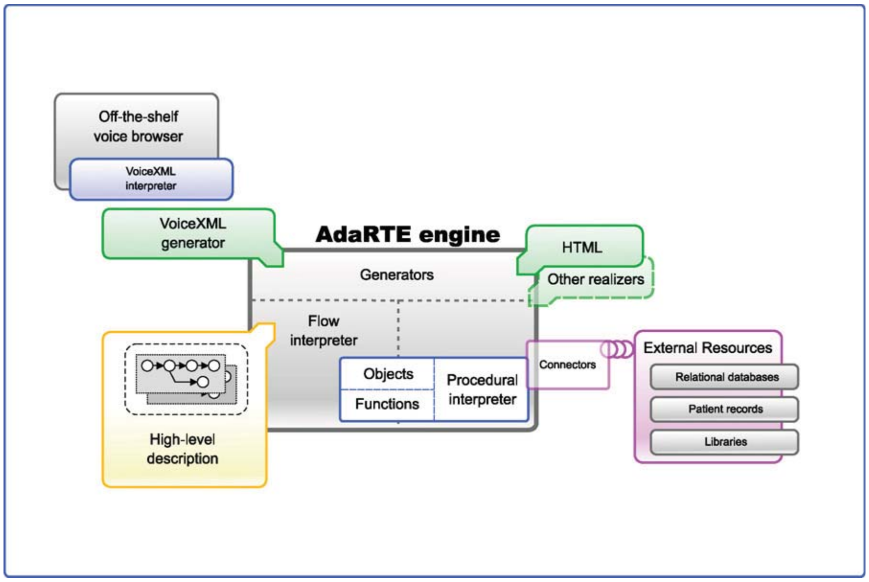

Fig. 1 – Block diagram of the AdaRTE architecture.

test their applications autonomously; and (2) experiment with best practices and alternative dialog strategies, incorporating them in existing systems, where possible, without rewrite. (So-called "best practices" suggest consistent adoption of dialog features like self-revealing prompts, incremental amounts of help, smart recovery from ASR recognition errors, and so on [34].)

4. Adaptable runtime engine

The approach we propose is AdaRTE, a flexible dialog manager patterned around the *combined procedural–declarative approach* to dialog-based interfaces. According to Allen et al.'s classification [19], the new approach falls into the "sets-of-context" category, because it combines the static nature of contexts and slots with the dynamic features of a procedural interpreter. The interpreter provides enough flexibility in execution in order to switch contexts according to arbitrarily complex criteria. The main components of the proposed architecture are (a) a *dialog interpreter*, (b) a *runtime engine* and (c) an *interface media realizer for back-end generation* (Fig. 1). A system typically interacts with three main roles: dialog developers, patients and case managers (e.g. physicians or case manager nurses, typically through a web interface).

The flow of a conversation is structured as a series of augmented transition networks (ATN). Usually, an ATN is associated with a context or a topic; we call this structure *subdialog*. Subdialogs partition a complex application into modules (Fig. 2). This contributes to structuring the conversa-

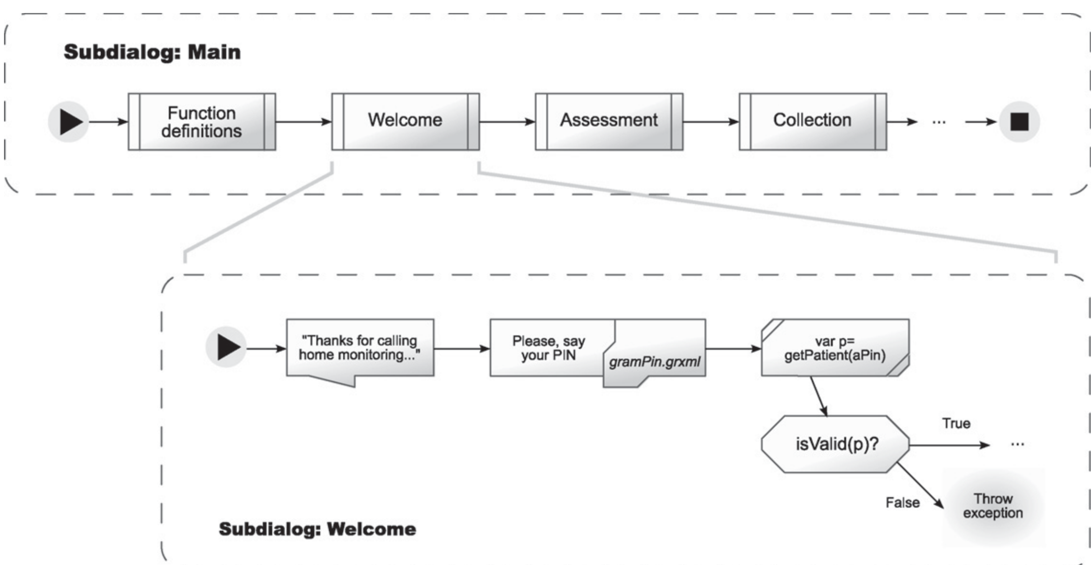

Fig. 2 – Block-based dialog description. Subdialogs are defined by the application developer (shown here as rounded dotted boxes), and can be invoked with a subdialog calling block (shown with a double border).

Please cite this article in press as: L.M. Rojas-Barahona, T. Giorgino, Adaptable dialog architecture and runtime engine (AdaRTE): A framework for rapid prototyping of health dialog systems, Int. J. Med. Inform. (2008), doi:10.1016/j.ijmedinf.2008.07.017

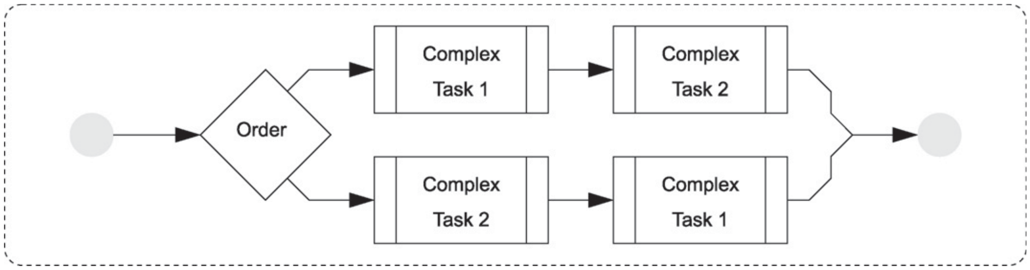

Fig. 3 – Within-application reuse of blocks. The dialog sequence is rearranged without duplication.

tion layout for ease of maintenance, because subdialogs can be reused both within the same dialog, and across different applications (Fig. 3).

Prompts, questions and other elements are the nodes (here named *blocks*) of an ATN that specify the flow of the conversation. Blocks are represented in the description by XML tags. When the system is started, the XML dialog description is read and compiled into an internal representation. When a call reaches the system, the dialog representation is executed. Consequently, AdaRTE activates the dialog blocks, constructs prompts, interprets the answers returned by the caller through the voice platform and interacts with external resources as appropriate.

When a call is set up, the *main* subdialog is retrieved and started; it will, in turn, invoke other subdialogs, and so forth. When the end of each subdialog is reached, the execution flow returns to the caller and, at the end of the main subdialog, the call is terminated. The subdialog flow can be altered by throwing dialog-specific exceptions.

Several blocks are available for building subdialogs, namely: prompt, question, script, decision, exception handler, prompt sets, place-holders, containers and subdialog calling blocks (Table 1).

Containers are another important feature that support the building of natural-sounding interactions; they are used for common tasks in which one of several subdialogs is selected according to a specified policy (Fig. 4). Containers provide a direct way for designers to augment the flexibility of the dialog, for instance allowing prompts to accommodate users' experience with the system. Policies for the activation of blocks inside containers could be: randomly, in sequence, ordered by call number and according to an externally defined schedule. Another policy could be generated, for instance, by performing statistical analysis to classify the level of experience of the user or even the likelihood of their encountering problems on specific parts of the dialog.

Equally central, along with the declarative block structure, is the embedded procedural interpreter. It provides an execution space which is shared with the block structure. The inclusion of procedural code is essential for flexibility, inter-operability and ease of programming. Almost all prompts, questions and other user-visible elements are evaluated dynamically at runtime. Evaluation can include function and method calls; this allows, for example, the modeling of entities (like, for examples, dates, therapy and doses) as complex objects. Therefore, they can be converted to their various natural-language representations via convenient calls to their methods. Furthermore, AdaRTE allows the embedding of larger blocks of code written in the ECMAScript language into *script blocks*. Procedural code has extensive access to external resources, Java standard libraries and user-provided APIs. Note that, in contrast to VoiceXML, the script interpreter belongs to the *server* side, and thus it has access to external resources such as databases, ontologies or any other commodity library.

Standard-conforming VBs provide basic mechanisms for semantic interpretation, e.g. through compliance with the SISR specification [35]. Grammars conforming to the specification construct objects and set their properties according to the parts of the sentence recognized. These objects are transferred to the AdaRTE server; in this way, semantic interpretation can

Table 1 – Main block classes for the AdaRTE dialog formalism

Block class	Description
Prompt	Evaluates its content and realizes the result in speech
Question	Evaluates and plays its content; activates a grammar and binds the return value to an object
Script	Allows bulk evaluation of arbitrary procedural and object-oriented code; may be used, e.g. for defining functions, evaluating complex formulas and accessing external resources and libraries
Decision	If- and switch-like statements evaluate their arguments and consequently alter the block execution flow
Subdialog	Activates a different subdialog, placing it on the call stack
Exception handlers	Exception handlers (catch) and generators (throw) handle out-of-band events, including "no match" and similar and user-defined events
Container	Multiple blocks can be grouped into a container and activated according to a chosen criterion, e.g. incrementally across calls, to provide different information in subsequent contacts, or according to any suitable "profile ability function"
Placeholder	Can be instantiated as no-operation blocks and their implementation deferred to later
Head	One-time declarations, e.g. directions for database access

Please cite this article in press as: L.M. Rojas-Barahona, T. Giorgino, Adaptable dialog architecture and runtime engine (AdaRTE): A framework for rapid prototyping of health dialog systems, Int. J. Med. Inform. (2008), doi:10.1016/j.ijmedinf.2008.07.017

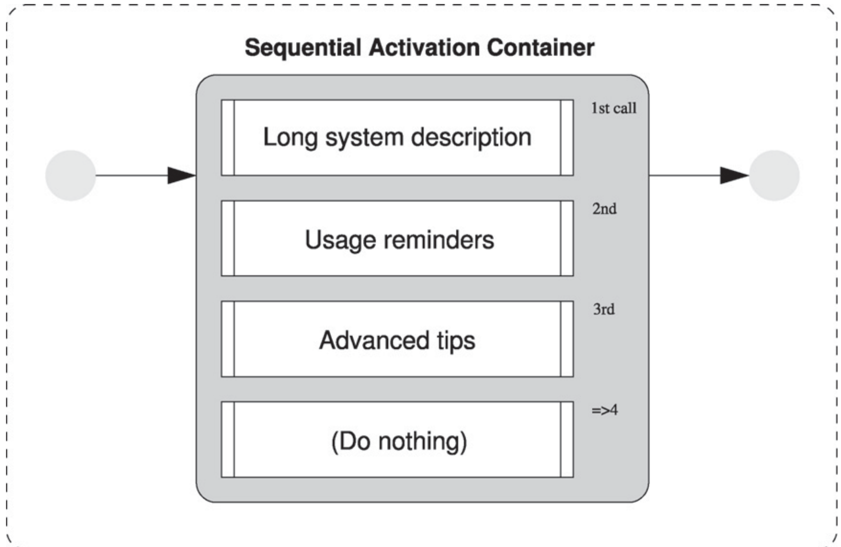

Fig. 4 – Containers automate switching between homologous blocks. Switching happens according to a container-specific policy—in this case, only one contained block is activated per invocation, according to the call number for that patient: a long system description is played on the first call, a brief reminder is given on the second time he calls, and so on. Containers simplify the addition of variability to the dialog.

happen both for the client, according to the SISR rules or be coded in the server, e.g. using regular expressions, statistical text models or NLP libraries.

Vendor-independence is an important consequence of the architecture of AdaRTE; it operates with off-the-shelf speech recognition software. In particular, for telephone interactions, it acts as a web server and dynamically transmits VoiceXML code to a voice browser. The VB, which is in turn connected to other hardware, will be in charge of interpreting documents according to the user's interaction over the phone. The browser captures and recognizes the answers, and sends them back to AdaRTE through HTTP requests. Vendor independence means that VBs can be replaced depending on economic considerations, the quality of the recognition and languages supported, or even outsourced.

A large body of research is available on the optimization of spoken interfaces. Some of the results of the research have been condensed into best practices [36]. More complex confirmation strategies with respect to simple "yes/no" answers, for example, should be adapted according to confidence thresholds and *n*-best lists. The inclusion of such techniques into custom-developed systems is complex. A big advantage in using an interpretable and high-level dialog representation language like the one proposed in this work is that such "dialog practices" can be incorporated seamlessly into the underlying dialog interpretation, removing the burden from the dialog developer. Within the AdaRTE formalism, for example, developers can enable either plain or skip-list-based confirmation steps to questions which are more prone to ASR misrecognitions (e.g. critical questions or those activating more complex language models). These features will be handled internally on the server-side, which will map them into lower-level constructs.

A well-known limitation of conventional ASR grammars is that – *on field* – they do not perform well in some domains, e.g. with lists of drug names. Since AdaRTE is independent of the VB, more complex language models can be employed, as long as they are offered by the underlying platform. This includes, for example, especially trained *n*-grams or speaker-dependent adaptations.

5. Evaluation and results

The AdaRTE framework is currently fully operative and it has been integrated with three voice service providers (VSPs). In this section, we present the evaluation of the framework through the construction of three realistic health care dialog systems derived by actual systems, which had been deployed and validated in the past, and two more test cases for peritoneal dialysis and OAT assistance. The implementation details are also presented in this section. A number of practical examples are displayed, describing the implementation of some of the important features explained in Section 4.

5.1. *Evaluation*

AdaRTE was evaluated by constructing five health-care dialogs, paying special attention to the metrics that allow us to measure each dialog development process. As a consequence, the strength of the framework is demonstrated by describing the variety of supported voice applications. At first, we consider as metrics the time invested, the expertise of developers and the technology and platform used in each developed dialog, as shown in Table 2. The first dialog prototype is based on the TLC-COPD system previously deployed by Young et al. [3]. For this specific example, we used Tellme Studio as the VSP. The implementation of this pilot required less than 2 weeks of man-effort. The fulfilled activities, shown in Fig. 5,

Please cite this article in press as: L.M. Rojas-Barahona, T. Giorgino, Adaptable dialog architecture and runtime engine (AdaRTE): A framework for rapid prototyping of health dialog systems, Int. J. Med. Inform. (2008), doi:10.1016/j.ijmedinf.2008.07.017

Table 2 – Five test-case health dialogs implemented in AdaRTE

No.	Prototype	Development		Mode	Grammar	Platform	Language	Reference
		Time (week)	Prof.					
1	Chronic obstructive pulmonary disease	2	Expert	DTMF	–	Tellme Studio, Loquendo	English	[3]
2	Hypertensive patients management	3	Expert	Speech	GSL, SGRS	Voxpilot, Loquendo	Italian	[4]
3	Diabetes	4	Junior	Speech	SGRS	Loquendo	Italian	[5]
4	Dialysis	4	Junior	Speech	SGRS	Loquendo	Italian	–
5	Oral anticoagulation therapy	4	Junior	Speech	SGRS	Loquendo	Italian	–

The first three prototypes were based on previous implementations (see references). GSL is the Nuance Grammar Specification Language; SRGS is the Speech Recognition Grammar Specification from the WWW Consortium [37].

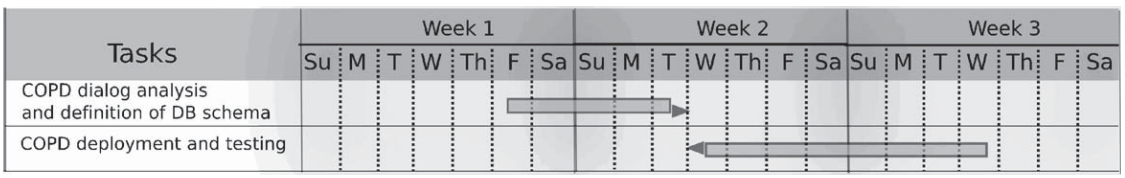

Fig. 5 – Gantt diagram of the COPD dialog pilot prototype implemented in AdaRTE.

included database schema definition, data preparation and dialog deployment. This dialog is in the English language and based on phone keypad interaction (also known as touch-tones, based on dual-tone multi-frequency or DTMF).

The second test case has been the partial re-implementation of the Homey dialog system for the care of hypertension. It included an extensive electronic health records (EHR) system with storage of personal data and profiles, in order to support dialog adaptability. The original system has been used at two Italian hospitals for approximately 2 years [4]. Despite the successful deployment, the time spent in developing the original system (Fig. 6) has been rather long and the result was reusable only to a limited extent. The voice part of the Homey system, for instance, took approximately one man-year to design and implement. Re-engineering the system from the original proprietary dialog manager to the AdaRTE architecture required approximately 3 weeks (11 days of man effort). This valuable test case allowed a side-to-side comparison between different dialog development environments. The redevelopment of this prototype involved the following activities: VSP evaluation, database definition, and grammars and dialog deployment (Fig. 7). Unlike the TLC-COPD pilot, this system makes extensive use of grammars for speech input; grammars were formulated both in the GSL (Nuance 7) and the SRGS grammar formats [37], and the dialog was tested by using the Voxpilot and Loquendo VoxNauta 7.0 VSP platforms.

Dialogs 3–5 in Table 2 have been implemented by external, junior developers; all of them support speech input and are executed in Italian using Loquendo VoxNauta 7.0 as VSP. The third dialog is based on the IVR used in the multi-access service for the management of diabetes mellitus patients (M2DM) project [5]. The goal of this dialog is for diabetic patients to enquire about information regarding insulin and glucose self-measurements. The dialog adapts its interaction according to the patient's therapy. The fourth dialog provides assistance to

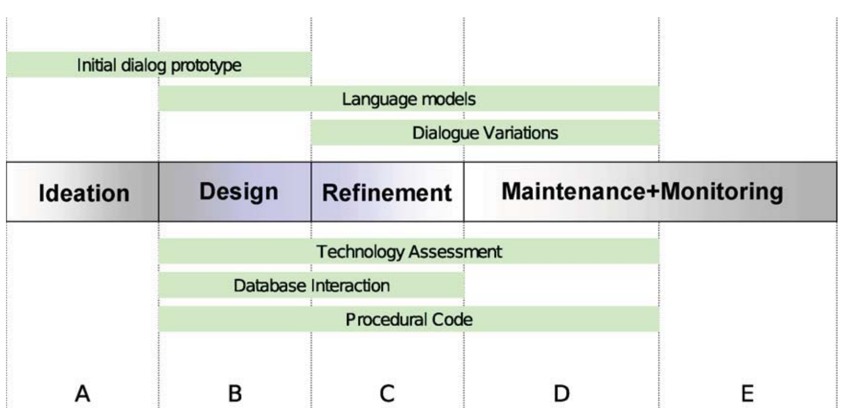

Fig. 6 – Phases in the development cycle of the original context-based Homey system.

Please cite this article in press as: L.M. Rojas-Barahona, T. Giorgino, Adaptable dialog architecture and runtime engine (AdaRTE): A framework for rapid prototyping of health dialog systems, Int. J. Med. Inform. (2008), doi:10.1016/j.ijmedinf.2008.07.017

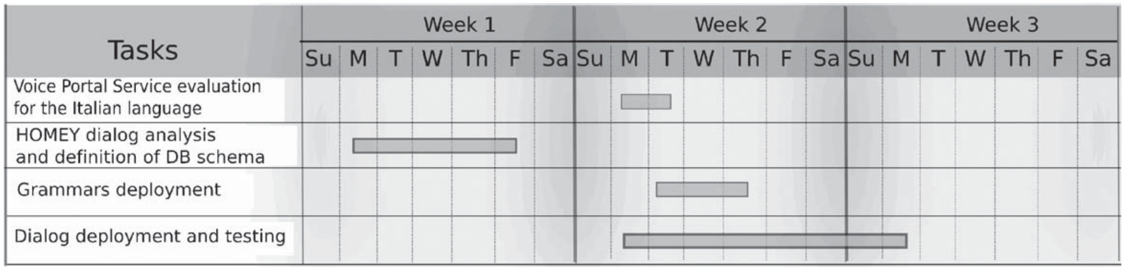

Fig. 7 – Gantt diagram of the Homey pilot developed in AdaRTE.

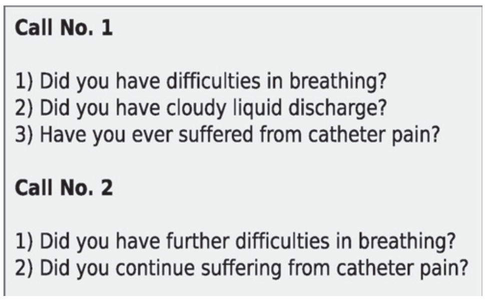

Fig. 8 – Sample of adapting questions across calls for the tele-dialysis dialog prototype.

patients undergoing peritoneal dialysis. It interacts to obtain information concerning the home dialysis process and the health status. The dialog adapts its interaction according to the patient's expertise, answers and history as shown in Fig. 8. The expertise level is calculated on the average of "no match" and "no input" events registered during a call. The patient's history gathers the clinical information, the answers on previous calls and the typology of the dialysis therapy.

The last dialog was designed to communicate the daily therapy to patients under OAT, and to verify that the patient has understood the therapy correctly. This dialog is being designed with the Mondino Neurological Hospital of Pavia (Italy).

Other metrics were used to measure the completeness and usability of AdaRTE, demonstrating that the framework covers all of the features previously introduced in Section 4 and indicating the level of complexity of the dialogs that can be implemented in the framework. Conceptually, the metrics were classified in database complexity (D), language-model complexity (L), application complexity (C) and front-end complexity (F) as shown under the column "type" of Table 3. For instance, the EHR designed for the Homey system has 78 tables and it is only accessed 6 times during the dialog. The explanation for this fact is that the Homey database (like the OAT) is a shared resource, accessed by other interfaces besides the vocal one. In order to evaluate the complexity of the language model, the internal representations of generated and recognized speech were considered. The former indicates whether or not the dialog exploits the SISR object-based mechanisms for the semantic representation of recognition results, while the latter describes whether or not the dialog uses an object-oriented approach to generate the output messages. Grouping prompts into objects turned out to be an especially effective programming technique: often, the same utterance should be conveyed in several styles with slight linguistic variations

Table 3 – The metrics used to represent the complexity of the developed dialogs were grouped into: D = database complexity, L = language model complexity, C = application complexity and F = front-end complexity

Type	Metrics	COPD	Homey	Diabetes	Dialysis	OAT
D	Number of DB accesses	19	6	26	13	10
D	Number of DB tables	12	78	17	31	12
L	Internal representation of speech/semantics	No	Yes	Yes	Yes	Yes
L	Internal representation of generated speech	No	No	No	No	Yes
L	Number of custom grammars	0	13	15	18	3
C	Total code lines	1127	781	2088	2415	882
C	Number of subdialogs	15	21	49	35	9
C	Number of blocks					
	Questions	46	33	41	62	8
	Prompts	34	11	76	31	27
	Scripts	17	14	39	58	17
	Exception handler	1	8	62	32	5
	Containers	0	1	0	0	0
	No-match/no-input	0	6	62	30	4
C	ECMAScript code lines	636	396	1225	1526	623
F	Modality (Voice/DTMF)	DTMF	Voice	Voice	Both	Voice
F	N-best/skip-list adoption	No	Yes	No	No	No

Table 4 – A comparison between the custom context-based Homey dialog and the AdaRTE-based prototype reimplementation

Type	Metrics	Homey: Custom	Homey: AdaRTE
D	Number of DB accesses	2	6
D	Number of DB tables	78	78
L	Internal representation of speech/semantics	Yes	Yes
L	Internal representation of generated speech	No	No
L	Number of custom grammars	~100	13
C	Total code lines	10132	781
C	Number of subdialogs	26	21
C	Number of blocks		
	Questions	–	33
	Prompts	–	11
	Scripts	–	14
	Exception handler	–	8
	Containers	–	1
	No-match/no-input	–	6
C	Procedural code lines	8776	396
F	Modality (Voice/DTMF)	Voice	Voice
F	N-best/skip-list adoption	No	Yes

(brief or verbose, singular or plural, request to repeat, etc.). Representing the utterance with the instance of an object allows the programmer to add more variations as the need arises, still handling the object as a unit.

The number of custom grammars conveys an idea of the domain-specific grammars implemented in each dialog. It is important to mention that many built-in grammars offered by the VB, e.g. numbers, Boolean, etc., were adopted and other custom grammars were reused in dialogs.

The number of code lines of the whole dialog, the length of the embedded scripts and the number of blocks provide an estimate of the level of complexity of each dialog application. Of course, the estimate is just a rough indication, because the dialog code can usually be simplified by expert developers and, on the other hand, expanded by the liberal use of comments. The number of ECMAScript code lines indicates how much the implemented dialogs took advantage of the benefits of the dynamic procedural interpreter; such code is mainly used for external resources access and decision making.

Table 4 compares the code-line counts for the previous proprietary context-based implementation of the Homey system with the AdaRTE-based reengineering. Despite being essentially the same application, a large amount of procedural code was formerly required for a relatively small number of database accesses, which retrieved the patient's profile at the beginning of the dialog and stored call outcomes at its conclusion. This was mainly due to the fact that database access mechanisms were not part of the dialog primitives. Also, lots of code was required for user profiling, because the simple procedural interpreter was somewhat unsuited to the evaluation of the complex criteria required for adaptability (e.g. decision making to alter the dialog flow with respect to user ability, checking ranges with respect to previous readings, and so on).

Table 3 depicts another metric, i.e. the number of blocks adopted by each dialog, grouped by block type. All the implemented dialogs used questions, prompts, scripts and exception handlers' blocks. Not surprisingly, information-gathering and patient-monitoring applications (dialogs 1–4) have a larger fraction of question blocks with respect to the OAT dialog, which is primarily an information-providing application, in which prompt blocks are prominent. The no-match and no-input are exception blocks, relevant for controlling these VB exceptions.

The front-end complexity was measured considering the modality and the adoption of the confidence thresholds and n-best lists. The HOMEY dialog, for example, activated the n-best confirmation strategy inside question blocks. Thus, in the case of ASR misrecognition, the utterance will be added into the skip-list that contains elements to be discarded by the ASR in future recognitions during the whole confirmation process of the question under discussion.

5.2. *Examples*

Practical examples of the implementation of dialogs are presented in Figs. 9–12. Fig. 9 displays the "root" subdialog, *main*, of the COPD dialog description. Note that it invokes the main topics to be addressed in the dialog by subdialog call blocks. Fig. 10 presents the hierarchical structure of the subdialogs

```
<subdef name="main">
    <start id="1" next="2"/>
    <subdialog id="2" next="3" name="identification"/>
    <if id="3" next="4" cond="followUpCALL == '1'">
      <else next="5"/>
    </if>
    <subdialog id="4" next="6" name="FollowUpCall"/>
    <subdialog id="5" next="6" name="dyspnea"/>
    <subdialog id="6" next="7" name="closingStatement"/>
    <catch event="dialog.finishCall" next="7">
      <start id ="1" next="2"/>
      <script id="2" next="3" >
       <![CDATA[
          finishCall = true;
       ]]>
      </script>
      <subdialog id="3" next="4" name="goodbyStatement"/>
      <end id="4"/>
    </catch>
   <end id="7"/>
</subdef>
```

Fig. 9 – Top-level dialog sequence (COPD example).

Please cite this article in press as: L.M. Rojas-Barahona, T. Giorgino, Adaptable dialog architecture and runtime engine (AdaRTE): A framework for rapid prototyping of health dialog systems, Int. J. Med. Inform. (2008), doi:10.1016/j.ijmedinf.2008.07.017

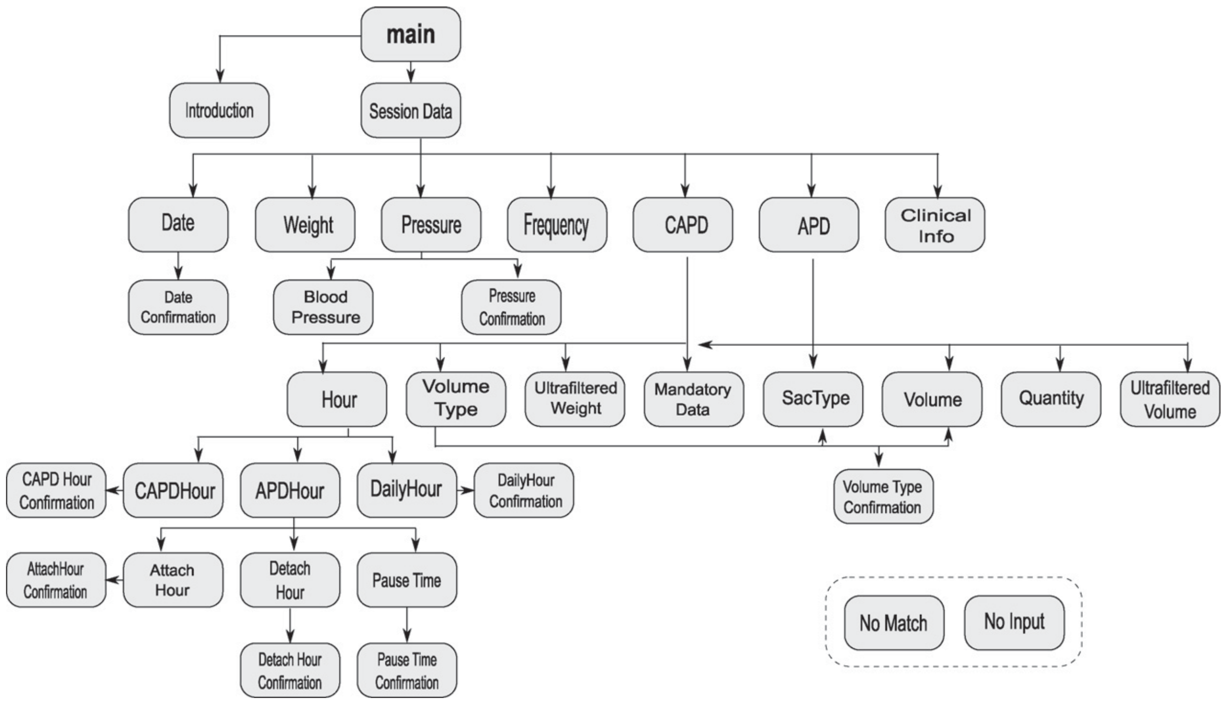

Fig. 10 – Hierarchical structure of subdialogs in the dialysis vocal application.

that make up the dialysis dialog and their flow. Figs. 11 and 12 provide examples of script blocks; they define functions that retrieve some patients' data from the database. Fig. 11 shows a basic method for placing a query: a structured query language (SQL) query statement is constructed and delivered directly to the database via the Java Database Connectivity (JDBC) API [38]. In contrast, Fig. 12 shows a more sophisticated approach adopted in the OAT dialog, which uses the Hibernate persistence framework to access the databases [39]. A persistence framework abstracts the mechanics of the query language, binding database entities to programming language objects.

6. Discussion and future enhancements

We presented five health-care dialogs that have been implemented in AdaRTE. Whilst implementing these applications, the developers profited from the features for rapid prototyping reuse, adaptable decision support and best practices provided by the framework. Through these dialogs, it has been proved that the framework fully supports these features. Moreover, the completeness and usability of the framework has been measured by describing the complexity of realistic applications that could be implemented with it.

```
<script id ="2" next="3">
  <![CDATA[
        initalize_vars();
        function setPatientCode(pin){
              var conn = DBConnection.getConnection("iper");
              var statement = conn.createStatement();
              var resultSet = statement.executeQuery("SELECT nopaziente,
                     CONCAT(CONCAT(nome,' '),cognome) AS nomePaziente
                     FROM anagrafica WHERE cod_telefono like '" + pin + "'" );

              while (resultSet.next()) {
                    nopaziente = resultSet.getInt("nopaziente");
                    nomePaziente = resultSet.getString("nomePaziente");
              }
              ....
  } ]]>
</script>
```

Fig. 11 – Procedural code of a script block that accesses a database through the JDBC API (authentication).

Please cite this article in press as: L.M. Rojas-Barahona, T. Giorgino, Adaptable dialog architecture and runtime engine (AdaRTE): A framework for rapid prototyping of health dialog systems, Int. J. Med. Inform. (2008), doi:10.1016/j.ijmedinf.2008.07.017

```
<script id="10" next="20" >
  <![CDATA[
      .......

      function patientByPin(sPin) {
         var result=null;
         try {
              var nPin=new java.lang.Integer(sPin);
              result=gDialogUtil.getPazienteByPin(nPin);
              myLog("patientByPin("+sPin+") returns "+result.getCognome());
         } catch(err) {
              myLog("Exception "+err);
               result=null;
         }
         return result;
  }
   ......
  ]]>
 </script>
..
```

Fig. 12 – Procedural code in a script block of the OAT dialog. This code retrieves an object representing the patient ("result"), given their personal identification number ("pin"). The actual work is performed by a method of the Java commodity object "gDialogUtil", which accesses the database by means of the Hibernate persistence framework.

In conclusion, the research and development effort provided an operative solution for the rapid prototyping of health dialogs, with a level of functionality that is not attained by current frameworks supporting more complex theoretical approaches to dialog. These frameworks should consider not only the intrinsic complexity of dialog modeling, but also the special requests in the medical domain, in order to offer a leaner development of robust and efficient dialogs. This is an important direction for future research in computational linguistics and medical informatics.

Currently, we have a strong commitment to the integration of a more elaborate semantic interpretation by integrating AdaRTE with an NLP application that supports lexicalized grammars to increase expressivity [40]. In this way, not only would recognition not depend on the grammars supported by VBs, but also more natural interactions will be supported, improving the patient's perception of dialogs.

The inclusion of spoken interfaces' optimization techniques or best practices in custom-developed systems is not straightforward. A big advantage in using an interpretable and high-level dialog representation language like the one proposed in this work is that more "dialog practices" can be incorporated seamlessly into the underlying dialog interpretation, removing the burden from the dialog developer.

Furthermore, extended support to the management of voice projects is foreseen, where a project involves a dialog and its composing subdialogs, together with definitions of templates. High-level templates serve as guidelines in the development of abstract tasks, e.g. for assessing the patient's psychological stage (useful e.g. for behavior-change interventions based on psychologically motivated models).

The AdaRTE system was foreseen not only as a reliable platform for dialog deployment, but also as a framework for incorporating advanced features of speech recognizers as they become available, including increased support to adaptability, and natural language understanding and generation. For instance, so far, AdaRTE supports the same amount of mixed initiative provided by the underlying VoiceXML interpreter. This could be enhanced by introducing stochastic-based grammars in the VSPs in order to increment the variety of possible expressions and the specialized medical terminology. Similarly, a more elaborate semantics representation could be considered in future frameworks for the easy deployment of dialogs in health.

7. Conclusion

We have presented a dialog-interpretation architecture for rapid dialog prototyping. The corresponding engine addresses current barriers to the realization of elaborate telephone-based interactions. AdaRTE differs significantly from other frameworks because it is targeted at the requirements of the chronic-care domain, which typically requires adaptable dialogs with complex structures and enquiry data collection tasks. The new methodology offers developers a high level of flexibility, by allowing access dynamically through the procedural execution environment surrounding the interpreter. At the same time, dialogs can be coded and inspected by developers who are not specifically trained in web-based technologies.

The expressiveness of the dialog representation yielded an important reduction of the time invested in developing a number of real-world prototypes. We have implemented five health-care dialog prototypes and shown that development times were remarkably optimized with respect to earlier development methodologies. Finally, AdaRTE is a standard-compliant architecture for the incremental adoption of and experimentation with advanced dialog formalisms.

Acknowledgments

The authors would like to thank Professor R. Friedman (Medical Information Systems Unit, Boston, MA) for information on the TLC-COPD dialogs; Professor T. Bickmore (College of Computer and Information Science, Northeastern University, Boston, MA) for the discussions that led to the AdaRTE project; Dr. A. Cavallini (IRCCS Mondino, Pavia) for the OAT project; I. Buetti, I. Limongelli, F. Carpignano, S. Derosa and others for contributing to dialog test cases and for the useful discussions.

REFERENCES

[1] T. Bickmore, T. Giorgino, Health dialog systems for patients and consumers, Journal of Biomedical Informatics 39 (5) (2006) 556–571.
[2] R. Lenz, R. Blaser, M. Beyer, O. Heger, C. Biber, M. Bäumlein, et al., IT support for clinical pathways—Lessons learned, International Journal of Medical Informatics Elsevier 76 (2007) S397–S402.
[3] M. Young, D. Sparrow, D. Gottlieb, A. Selim, R.H. Friedman, A telephone-linked computer system for COPD care, Chest 119 (2001) 1565–1575.
[4] T. Giorgino, I. Azzini, C. Rognoni, S. Quaglini, M. Stefanelli, R. Gretter, et al., Automated spoken dialogue system for hypertensive patient home management, International Journal of Medical Informatics Elsevier 74 (2004) 159–167.
[5] C. Larizza, R. Bellazzi, M. Stefanelli, P. Ferrari, P. De Cata, C. Gazzaruso, et al., The M2DM project the experience of two Italian clinical sites with clinical evaluation of a multi-access service for the management of diabetes mellitus patients, Methods of Information in Medicine 45 (2006) 79–84.
[6] J.D. Piette, Interactive voice response systems in the diagnosis and management of chronic disease, American Journal of Managed Care 6 (2000) 817–827.
[7] R. Corkrey, L. Parkinson, Interactive voice response: review of studies 1989–2000, Behavior Research Methods Instruments and Computers 34 (2002) 342–353.
[8] S. Krishna, E.A. Balas, S.A. Boren, N. Maglaveras, Patient acceptance of educational voice messages: a review of controlled clinical studies, Methods of Information in Medicine 41 (2002) 360–369.
[9] B. Kaplan, R. Farzanfar, R.H. Friedman, Personal relationships with an intelligent interactive telephone health behavior advisor system: a multimethod study using surveys and ethnographic interviews, International Journal of Medical Informatics Elsevier 71 (1) (2003) 33–41.
[10] M. Edmonds, M. Bauer, S. Osborn, H. Lutfiyya, J. Mahon, G. Doig, et al., Using the vista 350 telephone to communicate the results of home monitoring of diabetes mellitus to a central database and to provide feedback, International Journal of Medical Informatics Elsevier 51 (2/3) (1998) 117–125.
[11] J.P. Migneault, R. Farzanfar, J.A. Wright, R.H. Friedman, How to write health dialog for a talking computer, Journal of Biomedical Informatics 39 (October (5)) (2006) 468–481.
[12] E. Levin, A. Levin, Evaluation of spoken dialogue technology for real-time health data collection, Journal of Medical Internet Research 8 (4) (2006) e30.
[13] T. Bickmore, T. Giorgino, R. Picard, Guest editorial: special issue on dialog systems for health communication, Journal of Biomedical Informatics Elsevier 39 (2006) 465–467.
[14] A. Zafar, B. Mamlin, S. Perkins, A.M. Belsito, J.M. Overhage, C.J. McDonald, A simple error classification system for understanding sources of error in automatic speech recognition and human transcription, International Journal of Medical Informatics Elsevier 9/10 (2004) 719–730.
[15] J. Allen, G. Ferguson, N. Blaylock, D. Byron, N. Chambers, M. Dzikovska, et al., Chester: towards a personal medication advisor, Journal of Biomedical Informatics 39 (5) (2006) 500–513.
[16] M. Beveridge, J. Fox, Automatic generation of spoken dialogue from medical plans and ontologies, Journal of Biomedical Informatics 39 (5) (2006) 482–499.
[17] J. Allen, G. Ferguson, A. Stent, An architecture for more realistic conversational systems, in: Proceedings of Intelligent User Interfaces (IUI-01), Santa Fe, NM, 2001.
[18] R. Power, The organisation of purposeful dialogues, Linguistics: An International Review La Haye 17 (1/2) (1979) 107–151.
[19] J. Allen, D.M.D. Byron, G. Ferguson, L. Galescu, Towards conversational human–computer interaction, AI Magazine 22 (4) (2001) 27–37.
[20] C.R. Perrault, J.F. Allen, A plan-based analysis of indirect speech acts, American Journal of Computational Linguistics 6 (3/4) (1980) 167–182.
[21] G. Churcher, E. Atwell, C. Souter, Dialogue Management Systems: A Survey and Overview, School of Computer Studies, University of Leeds, 1997.
[22] S. Larsson, D. Traum, Information state and dialogue management in the TRINDI dialogue move engine toolkit, Natural Language Engineering Special Issue on spoken language dialogue system engineering, vol. 6, 2000, 323–340.
[23] S. McGlashan, D.C. Burnett, J. Carter, P. Danielsen, J. Ferrans, A. Hunt, et al. Voice Extensible Markup Language (VoiceXML) Version 2.0, 2004, available from: http://www.w3.org/TR/voicexml20/ (updated 2004; cited 2008).
[24] L.A. Black, M. McTear, N. Black, R. Harper, M. Lemon, Di@log by design: an integrated tele-care communication infrastructure, in: Proceedings of the 26th Annual International Conference of the IEEE EMBS San Francisco, vol. 2, 2004, pp. 3290–3293.
[25] M. Mittendorfer, G. Niklfeld, W. Winiwarter. Evaluation of Intelligent Component Technologies for VoiceXML Applications. Serial on the Internet, 2001, available from: http://speech.ftw.at/download/EvalVoiceXML.pdf.
[26] M. Mittendorfer, G. Niklfeld, W. Winiwarter, Making the voice web smarter-integrating intelligent component technologies and VoiceXML, in: Proceedings of the Second International Conference on Web Information Systems Engineering, vol. 2, 2002, pp. 126–131.
[27] D. Falavigna, R. Gretter, M. Orlandi. A mixed language model for a dialogue system over the telephone. ICSLP 2000, 2000, Beijing, China.
[28] D. Falavigna, T. Giorgino, R. Gretter, A frame based spoken dialog system for home care, in: Proceedings of INTERSPEECH, Lisbon, Portugal, September 4–8, 2005.
[29] Ecma International. Standard ECMA-262: ECMAScript Language Specification, 3rd ed. Serial on the Internet, 1999, available from: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.
[30] N. Hataoka, Y. Obuchi, T. Mitamura, E. Nyberg, Robust speech dialog interface for car telematics service, in: First IEEE Consumer Communications and Networking Conference, 2004, pp. 331–335.
[31] J. Hartman, J.A. Vila, VoiceXML Builder: a workbench for investigating voice-based applications, in: Proceedings of the 31st Annual Frontiers in Education Conference, vol. 3, 2001, pp. S2C6–9.
[32] S. Hamerich, V. Schubert, V. Schless, R. Crdoba, J. Pardo, L. d'Haro, et al., Semi-automatic generation of dialogue

Please cite this article in press as: L.M. Rojas-Barahona, T. Giorgino, Adaptable dialog architecture and runtime engine (AdaRTE): A framework for rapid prototyping of health dialog systems, Int. J. Med. Inform. (2008), doi:10.1016/j.ijmedinf.2008.07.017

applications in the GEMINI Project, in: Proceedings of the 5th SIGdial Workshop on Discourse and Dialogue, Cambridge, Massachusetts, USA, April 30–May 1, 2004.
[33] G. Di Fabbrizio, C. Lewis, Florence: a Dialogue Manager Framework for Spoken Dialogue Systems, in: Proceedings of the 8th International Conference on Spoken Language Processing, Jeju, Jeju Island, Korea, 2004.
[34] B. Balentine, D.P. Morgan, How to Build a Speech Recognition Application. A Style Guide for Telephony Dialogues, 2nd ed., EIG Press, California, 2001.
[35] L. Van Tichelen, D. Burke. Semantic Interpretation for Speech Recognition (SISR) Version 1.0. W3C Recommendation 5, April 2007.
[36] S. Weinschenk, T. Barker, Designing Effective Speech Interfaces, Wiley, New York, 2000.
[37] S. McGlashan, A. Hunt. Speech Recognition Grammar Specification Version 1.0, 2004, available from: http://www.w3.org/TR/speech-grammar/ (updated 2004; cited 2008).
[38] G. Hamilton, R. Cattell, M. Fisher, Jdbc Database Access With Java: A Tutorial and Annotated Reference (Java Series), Addison-Wesley, 1997.
[39] C. Bauer, Java Persistence with Hibernate, Manning Publications, New York, 2006.
[40] L.M. Rojas-Barahona, Adapting combinatory categorial grammars in a framework for health care dialogue systems, in: R. Artstein, L. Vieu (Eds.), Proceedings of the 11th Workshop on the Semantics and Pragmatics of Dialogue, Rovereto Italia, 2007.

Please cite this article in press as: L.M. Rojas-Barahona, T. Giorgino, Adaptable dialog architecture and runtime engine (AdaRTE): A framework for rapid prototyping of health dialog systems, Int. J. Med. Inform. (2008), doi:10.1016/j.ijmedinf.2008.07.017