Persistent Information State in a Data-Centric Architecture*

Sebastian Varges, Giuseppe Riccardi, Silvia Quarteroni
Department of Information Engineering and Computer Science
University of Trento
38050 Povo di Trento, Italy
{varges|riccardi|silviaq}@disi.unitn.it

Abstract

We present the ADAMACH data centric dialog system, that allows to perform on- and off-line mining of dialog context, speech recognition results and other system-generated representations, both within and across dialogs. The architecture implements a "fat pipeline" for speech and language processing. We detail how the approach integrates domain knowledge and evolving empirical data, based on a user study in the University Helpdesk domain.

1 Introduction

In this paper, we argue that the ability to *store and query* large amounts of data is a key requirement for data-driven dialog systems, in which the data is generated by the spoken dialog system (SDS) components (spoken language understanding (SLU), dialog management (DM), natural language generation (NLG) etc.) and the world it is interacting with (news streams, ambient sensors etc.). We describe an SDS that is built around a database management system (DBMS), uses the web service paradigm (in contrast to the architecture described in (Varges and Riccardi, 2007)), and employs a Voice XML (VXML) server for interfacing with Automatic Speech Recognition (ASR) and Text-to-Speech (TTS) components. We would like to emphasize upfront that this does *not* mean that we follow a VXML dialog model.

*This work was partially supported by the European Commission Marie Curie Excellence Grant for the ADAMACH project (contract No. 022593) and by LUNA STREP project (contract no33549).

The data centric architecture we adopt has several advantages: first, the database concentrates heterogeneous types of information allowing to uniformly query the evolving data at any time, e.g. by performing queries across various types of information. Second, the architecture facilitates dialog evaluation, data mining and online learning because data is available for querying as soon as it has been stored. Third, multiple systems/applications can be made available on the same infrastructure due to a clean separation of its processing modules (SLU, DM, NLG etc.) from data storage and persistency (DBMS), and monitoring/analysis/visualization and annotation tools. Fourth, there is no need for separate 'logging' mechanisms: the state of the SDS is contained in the database, and is therefore persistently available for analysis after the dialog ends.

As opposed to the presented architecture, the Open Agent Architecture (OAA) (Martin et al., 1999) and DARPA Communicator (Seneff et al., 1998) treat data as peripheral: they were not specifically designed to handle large volumes of data, and data is not automatically persistent. In contrast to the CSLI-DM (Mirkovic and Cavedon, 2005), and TrindiKit (Larsson and Traum, 2000), but similar to Communicator, the ADAMACH architecture is server-based, thus enabling continuous operation.

To prove our concept, we test it on a University helpdesk application (section 4).

2 Dialog System Architecture

Figure 1 shows our vision for the architecture of the ADAMACH system. We implemented and evaluated the speech modality based core of this system

Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue, pages 68–71,
Columbus, June 2008.

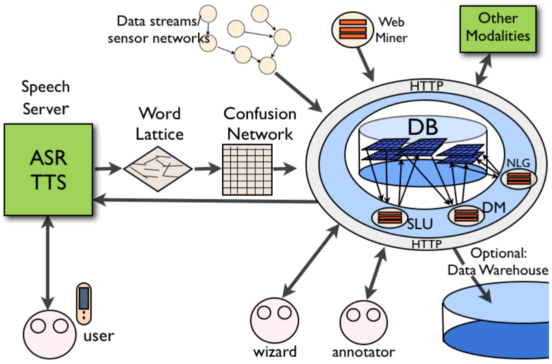

Figure 1: Architecture vision

(figure 2). A typical interaction is initiated by a phone call that arrives at an telephony server which routes it to a VXML platform. A VXML page is continuously rewritten by the dialog manager, containing the system utterance and other TTS parameters, and the ASR recognition parameters for the next user utterance. Thus, VXML is used as a low-level interface to the ASR and TTS engines, but *not* for representing dialog strategies. Once a user utterance is recognized, a web service request is issued to a dialog management server.

All communication between the above-mentioned components is stored in the DBMS: ASR recognition results, TTS parameters and ASR recognition parameters reside in separate tables. The dialog manager uses the basic tables as its communication protocol with ASR and TTS engines, and additionally stores its Information State (IS) in the database. This means that the IS is automatically *persistent*, and that dialog management becomes a function that maps ASR results and old IS to the TTS and ASR parameters and a new IS. The tables of the database are organized into turns, several of which belong to a call (dialog), thus resulting in a tree structure that is enforced by foreign key constraints in the relational database.

The VXML standard is based on the web infrastructure. In particular, a VXML platform can issue HTTP requests that can be served by a web server just like any (HTML) page. The VXML server only sees the generated VXML page, the 'return value' of the HTTP request. This allows us to organize the processing modules of the dialog system (SLU, DM, VXML generator) as web services that are invoked by the HTTP request. As a consequence, each system turn of a dialog is a separate, *stateless* request. The state of the dialog is stored in the database. Furthermore, by threading the VXML session ID through the processing loop (including the VXML pages generated on-the-fly) and distinguishing entries in the DB by sessions, the SDS is inherently parallelizable, just as a conventional web server can serve many users in parallel. Figure 2 shows how information is processed for each turn. The HTTP requests that invoke the processing modules pass on various IDs and parameters, but the actual data is stored in the DB and retrieved only if a processing module requires it. This effectively implements a 'fat pipeline': each speech, language and DM module has access to the database for rescoring and modeling (i.e. data within and across dialogs). At the implementation level, this balances a lightweight communication protocol downstream with data flowing laterally towards the database.

3 Dialog Management

Dialog management works in two stages: retrieving and preprocessing facts (tuples) taken from the database, and inferencing over those facts to generate a system response. We distinguish between the 'context model' of the first phase and the 'dialog move engine' (DME) of the second phase (Larsson and Traum, 2000).

The first stage entails retrieving from the persistent Information State the following information: all open questions for the current dialog from the database, any application information already provided by the user (including their grounding status), the ASR recognition results of last user turn, and confidence and other thresholds. The context model that is applied when retrieving the relevant dialog history from the database can be characterized as a 'linear default model': application parameters provided by the user, such as student ID, are overridden if the user provides a new value, for example to correct a previous misunderstanding. Task boundaries are detected and prevent an application parameter from carrying over directly to the new task.

The second stage employs an inference engine to determine the system action and response: SLU rules match the user utterance to open questions.

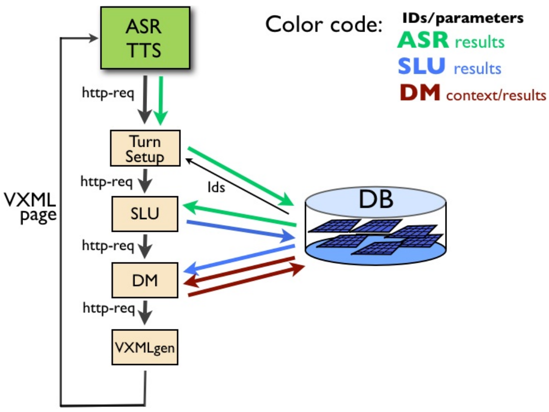

Figure 2: Turn-level information flow

This may result in the decision to verify the application parameter in question, and the action is verbalized by language generation rules. If the parameter is accepted, application dependent task rules determine the next parameter to be acquired, resulting in the generation of an appropriate request. For reasons of space, we cannot provide more details here.

4 Experiments

Our current application is a University helpdesk in Italian which students call to perform 5 tasks: receive information about exams (times, rooms ...), subscribe/cancel subscriptions to exams, obtain exam mark, or request to talk to an operator. Following experimentations, we annotated the dialogs and conducted performance statistics using the system's built-in annotation tool.

Two Italian mothertongues were in charge of manually annotating a total of 423 interactions. Each annotator independently annotated each dialog turn according to whether one of the five available tasks was being requested or completed in it. To compute inter-annotator agreement, 24 dialogs were processed by both annotators; the remaining ones were partitioned equally among them.

We computed agreement at both turn and dialog level. Turn level agreement is concerned with which tasks are requested and completed at a given dialog turn according to each annotator. An agreement matrix is compiled where rows and columns correspond to the five task types in our application. Cohen's κ (Cohen, 1960), computed over the turn matrix, gave a turn agreement of 0.72 resp. 0.77 for requests resp. completions, exceeding the recommended 0.7 threshold. While turn-level agreement refers to *which* tasks occurred and at what turn, dialog level agreement refers to *how many* task requests/completions occurred. Also at the dialog level, the κ statistic gave good results (0.71 for requests and 0.9 for completions).

General dialog statistics The average duration of the 423 annotated dialogs is 63.1 seconds, with an average of 7.43 turn (i.e. adjacency) pairs. 356 of the dialogs contained at least one task; the majority (338) contained exactly one, 17 dialogs contained 2 tasks, and one dialog contained 3. In the remaining 67 dialogs, no tasks were detected: from the audio files, it seems that these generally happened by accident or in noisy environments, hence noinput/hangup events occurred shortly after the initial system prompt.

Furthermore, relative frequencies of task requests and task completions are reported in Table 1. In total, according to the two annotators, there were 375 task requests and 234 task completions. Among the requested tasks, the vast majority was composed by "Get exam mark" –a striking 96%– while "Exam withdrawal" never occurred and the three others were barely performed. Indeed, it seems that students preferred to use the system to carry on "informative" tasks such as obtaining exam marks and general information rather than "active" tasks such as exam subscription and withdrawal.

Table 1: Task request and completion frequencies (%)

Task	**Request**	**Completion**
Get exam mark	96 (360)	96.6 (226)
Info on exam	1.9 (7)	1.7 (4)
Exam subscription	1.1 (4)	0.4 (1)
Exam withdrawal	0.0 (0)	0.0 (0)
Talk to operator	1.1 (4)	1.3 (3)
Total	100 (375)	100 (234)

Task and dialog success Based on the annotation of task requests and completions, we defined task success as a binary measure of whether the request of a given task type is eventually followed by a task completion of the same type. Table 2 reports the average success of each task type according to the annotators[1]. Our results show that the most frequently requested type, "Get exam mark", has a 64.64% success rate (it seems that failure was mostly due to the system's inability to recognize student IDs).

Table 2: Top: annotator (sr_M) and automatic (sr_A) task success rates. Mean $\pm$ binomial proportion confidence interval on the average task success (α= 95%) is reported. Bottom: mean annotator (dsr_M) and automatic (dsr_A) dialog success rates $\pm$ normal law c.i. (α= 95%).

Task	sr_M(%)	sr_A(%)
Get exam mark	64.64	77.97
Info on exam	57.14	71.43
Exam subscription	25	100
Exam withdrawal	-	-
Talk to operator	75	75
Average	64.17±4.96	78.06±4.28
Dialog	dsr_M(%)	dsr_A(%)
Average	64.47±4.95	88.31±9.2

In fact, while it is straightforward to obtain task success information using the manual annotation of dialogs, when the dialog system cannot rely on human judgments, unsupervised approaches must be defined for a rapid (on-line or off-line) evaluation. For this purpose, an automatic approximation of the "manual" task success estimation has been defined using a set of database queries associated to each task type. For instance, the task success query associated to "Info on exam" checks that two conditions are met in the current dialog: 1) it includes a turn where an action is requested the interpretation of which contains "information"; 2) it contains a turn where the concept `Exam_Name` is in focus.

Automatic task success rates have been computed on the same dialogs for which manual task success rates were available and are reported in Table 2, col. 2. The comparison shows that the automatic metric sr_A is more "optimistic" than the manual one sr_M. Indeed, automatic estimators rely on "punctual" indicators (such as the occurrence of confirmations of a given value) in the whole dialog, regardless of the task they appear in (this information is only available from human annotation) and also of the order with which such indicators appear in the dialog.

[1]As several task types occur seldom, we only report the confidence intervals on the means relating to the overall ("Average") task success, computed according to the normal law.

As a by-product of task success evaluation, we defined dialog success rate (dsr) as the average success rate of the tasks in a dialog: $dsr = \frac{\sum_{t_i \in T} sr(t_i)}{|T|}$, T being the set of requested tasks. Depending on whether sr_M or sr_A is used, we obtain two metrics, dsr_M resp. dsr_A.

Our dialog success results (last row of Table 2) are comparable to the task success ones; also, the difference between the automatic and manual estimators of dialog success is similar to their difference at the the task level. This is not surprising when considering that most of the dialogs contained only one task.

5 Conclusions

We have presented a data-centric Spoken Dialog System whose novel aspect is the storage and retrieval of dialog management state, ASR results and other information in a database. As a consequence, dialog management can be lightweight and operate on a turn-by-turn basis, and dialog system evaluation and logging are facilitated.

Acknowledgments

We would like to thank Pierluigi Roberti for helping with the speech platform and annotation tools, and LOQUENDO for providing the VXML platform.

References

J. Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20:37–46.

S. Larsson and D. Traum. 2000. Information State and dialogue management in the TRINDI Dialogue Move Engine Toolkit. *Natural Language Engineering*, 6(3–4):323–340.

D. L. Martin, A. J. Cheyer, and D. B. Moran. 1999. The Open Agent Architecture: A framework for building distributed software systems. *Applied Artificial Intelligence: An International Journal*, 13(1-2):91–128.

D. Mirkovic and L. Cavedon. 2005. Practical Plug-and-Play Dialogue Management. In *Proceedings of PACLING*, Tokyo, Japan.

S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue. 1998. GALAXY-II: A reference architecture for conversational system development. In *Proc. of ICSLP 1998*, Sydney, Australia.

S. Varges and G. Riccardi. 2007. A data-centric architecture for data-driven spoken dialogue systems. In *Proceedings of ASRU*, Kyoto, Japan.