JASPIS^2 – AN ARCHITECTURE FOR SUPPORTING DISTRIBUTED SPOKEN DIALOGUES

Markku Turunen and Jaakko Hakulinen

Speech-based and Pervasive Interaction Group, Tampere Unit for Computer-Human Interaction
Department of Computer and Information Sciences, University of Tampere, Finland
{mturunen,jh}@cs.uta.fi

Abstract

In this paper, we introduce an architecture for a new generation of speech applications. The presented architecture is based on our previous work with multilingual speech applications and extends it by introducing support for synchronized distributed dialogues, which is needed in emerging application areas, such as mobile and ubiquitous computing. The architecture supports coordinated distribution of dialogues, concurrent dialogues, system level adaptation and shared system context. The overall idea is to use interaction agents to distribute dialogues, use an evaluation mechanism to make them dynamically adaptive and synchronize them by using a coordination mechanism with triggers and transactions. We present experiences from several applications written on top of the freely available architecture.

1. Introduction

Most current speech applications are focused on the telephone and desktop environments and are typically information services between a single user and a computer. Other popular application areas, such as information kiosks often share these same characteristics. New challenges arise when speech is used more and more in pervasive computing applications. In mobile and ubiquitous applications dialogues can be distributed, concurrent, open ended, dynamically constructed, and involve multiple participants in dynamic environments. Examples of these applications and their challenges are presented in [4], [7] and [11], while [5] introduces challenges with near-future applications. At the same time, there is a need to make speech interaction more natural in traditional application areas as well. Issues such as back-channeling bring similar challenges, as presented in [1]. To support the interaction needed in these areas we need better interaction methods for managing the interaction. We consider the key issues to be:

- *Distributed nature of dialogues*: Dialogues can be distributed in many senses. First, the dialogues can take place in multiple physical places, such as in different rooms in ubiquitous computing applications. Second, dialogue management may be distributed over different components instead of a single dialogue manager. Agent-based systems are examples of this approach. In both of these cases, users should hear the conversation as one unified dialogue.
- *Coordination and synchronization of concurrent dialogues*: Multiple dialogues appear at the same time. As they are possibly related, coordination and synchronization are important elements and the efficient sharing of information between the system components is vital.
- *Adaptability of interaction*: Systems should be able to adapt their behavior to the different needs and communication patterns of users and to different environments. In order for the adaptation to be efficient, it should be done at the system level.

Current approaches to distributed dialogues include agent-based systems, such as TRIPS [1]. They still have many limitations. One is the monolithic nature of agents, which makes the systems' ability for true distribution rather limited. The second is adaptivity which is not modeled at system level, but instead implemented in ad-hoc ways in different system components. The third is the coordination and synchronization of agents, which is usually either not modeled at all or is implemented by using one central component which is often the bottleneck of the system. The fourth is information management which usually does not support shared knowledge.

We introduce a spoken dialogue architecture which addresses these challenges by presenting a general infrastructure for building distributed spoken dialogue systems. The proposed Jaspis^2 architecture is based on our previous work with adaptive agent-based speech systems [12] and provides mechanisms for distribution, coordination and dynamic selection of agents. It also provides a common structure for shared information management. The architecture does not assume any special dialogue paradigm and it can be used with different dialogue control strategies. The implementation of the architecture provides concrete tools for building speech systems. The architecture is freely available and it has been used in several practical applications, such as [13] and [3].

The rest of this paper is organized as follows: first, we provide an architecture overview, then describe the information and interaction management components in detail. After that, application issues are discussed. Finally, a summary and conclusions are presented.

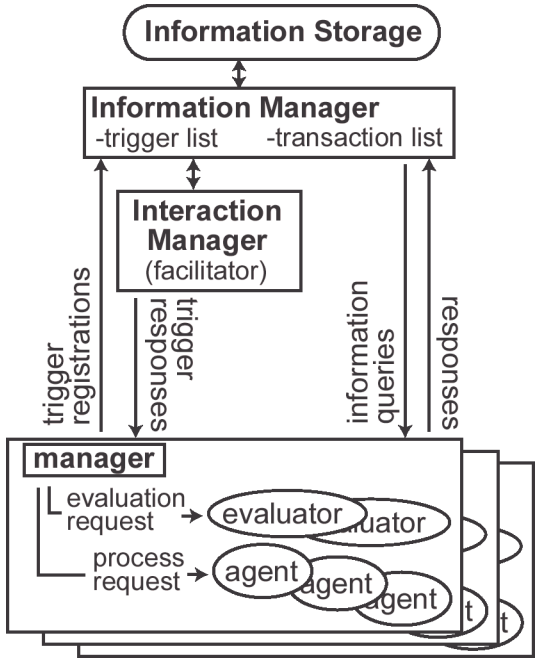
2. Architecture Overview

The Jaspis^2 architecture includes two logical levels: the core infrastructure and the collection skeletons (modules). Here we will focus on the core infrastructure. It includes components and mechanisms for the distribution, coordination, selection and execution of the system components, and the information exchange between these components. It does not define what the individual components do, or what they are. In this sense, the core architecture is flexible and in many senses similar to Galaxy-II [10] and many agent-based infrastructures, such as OAA [6].

The core infrastructure which is illustrated in Figure 1, is based on two principles. First, it combines the benefits of the blackboard-based and the message/event-based approaches into an information and interaction management scheme

which supports distribution and coordination. This is realized using the *Information Storage* and *Information Manager* components. System activity is based on *triggers* which react to changes in Information Storage. The triggers can activate multiple components at the same time and therefore systems can be concurrent. *Transactions* are used to synchronize concurrent actions. Secondly, the architecture provides an explicit model for the synchronized distribution of interaction components and adaptability. This is realized by using *managers*, *agents* and *evaluators*. Agents are the components that handle interaction situations. Evaluators are used to choose the most suitable agents for each situation. Managers are used to coordinate these components.

Figure 1: Architecture overview.



On top of the basic infrastructure, the architecture contains *collection skeletons* (modules) for tasks such as dialogue management, multimodal input and output processing, and the presentation of system messages. These general components provide support for the coordination and implementation of different interaction tasks. Interfaces for speech recognizers and synthesizers are included in these components, as well as support for development tools. These components contain most of the functionality needed in practical speech systems and they are included in the applications in a plug-and-play fashion. New collections can be introduced and standard ones can be replaced with alternative solutions when necessary. These are presented in more detail in [12] and [14].

3. Information Management

Centralized information management addresses the need for coordination that is necessary for distributed and concurrent dialogues. Information management is based on the *Information Storage* and *Information Manager* components which together provide shared, blackboard-type information storage. Blackboard architectures have several advantages, including the ability to use de-centralized control structures and the option for every system component to utilize all the information that the system contains. It is also possible to add new components to the system without defining strict relations (and corresponding system interfaces) between the components. From the application developer's point of view, this offers rich possibilities for iterative system design and development tasks. The benefits of the blackboard approach have been identified in many spoken dialogue systems (e.g. [5] and [7]). In addition, many systems that are based on the message-passing approach use shared information resources at least to some extent (e.g. [1]). The drawback of the blackboard approach is considered to be that it does not provide coordination when it is needed [6]. We address this issue by using managers and transaction for coordination and synchronization.

3.1. Shared System Context: Information Storage

Information Storage offers methods for storing and manipulating shared system information. Since different applications may require different information storing methods, the architecture does not define how the information is actually stored, but defines how it is accessed. This is defined by the *Information Access Protocol*, which is an XML DTD. XML-RPC is used for infrastructure level communication and Annotation Graphs [2] for storing and exchanging linguistic information. The usefulness of XML in NLP applications has also been noticed by other researchers [16].

The standard reference implementation of the storage uses XML for information storing as well. Information is organized hierarchically so that different information resides in different branches of the hierarchy. This makes the management of the storage fast and efficient, especially when only small fractions of the storage are manipulated at a time.

3.2. Information Access: Information Manager

Although Information Storage is seen logically as one component, in practice multiple storages are needed in many cases. Information Manager coordinates different storages and provides an image of a single storage to the rest of the system. The hierarchical structure makes it efficient to distribute the storage to multiple sources, i.e. different branches of the storage can reside in different places. Information Manager also offers methods for other components to access certain information directly without the need to know where the information is stored in the information hierarchy. Different implementations of the Information Manager are used to provide access to heterogeneous components written in different programming languages. In general, it provides a convenient and unified way to access the system context. Information Manager is also used to maintain *triggers.*

3.3. Communication and Synchronization: Triggers and Transactions

Triggers are used to provide indirect messaging between the system components and enable concurrency. Any manager or agent can register triggers. When a trigger becomes active, the manager that has registered it is acknowledged so that it can evaluate the situation and give its agents a possibility to react. It should be noted that even if an agent had registered a trigger, the local *manager* would be notified. This makes system level coordination and adaptation possible.

Triggers are based on the information in Information Storage. When a manager registers a trigger, it gives Information Manager a set of rules that need to be met for the trigger to become active. The rules contain pointers to positions in the information storage and operators for these values.

Synchronization is done by using *transactions*, a concept borrowed from database management systems. Related triggers are combined into transactions, which are activated together. When agents operate, they usually start a transaction, do all their processing within that transaction, and close it when they end processing. Agents can use multiple transactions if they want to provide the system with more concurrency. The transaction mechanism makes sure that the triggers work at correct times, and that the triggers that are closely related (i.e. activate during the same transaction) are handled in a synchronized way. It also reduces the number of times the trigger rules need to be evaluated.

Triggers and transactions are implemented by the Information Manager, because they can reference to information found inside multiple information storage components.

4. Interaction Components

As described in the previous sections, Jaspis interaction components consist of *agents*, *evaluators*, and *managers*.

4.1. Distributed Interaction Tasks: Agents

The distribution of components is based on small stateless agents. Agents are often understood to be autonomous and the tasks that they handle rather complex. For example, a speech-based system can contain such agents as a dialogue agent, a parser agent and a generation agent. The number of agents in a typical setup is rather small (for example, see [4]). The interface to these agents is usually described by means of information exchange languages, such as KQML.

The Jaspis[2] architecture uses a different approach. First, it has been designed with compact agents in mind. By compact agents we mean agents that handle atomic tasks. Agents are small and typical systems contain large numbers of them. For example, in our telephone-based e-mail application [13], there are no single dialogue and generation agents, but instead we have thirty dialogue agents and even more presentation (generation) agents. The agents are independent and stateless, which makes the dialogue highly distributed. This approach differs greatly from many other agent systems, where agents are in fact monolithic software modules.

Typical agents are specialized in certain tasks, which can be domain specific, or general problem solving tasks. In addition to this co-operational approach, agents can be specialized in the same tasks with varying behaviors. This allows application developers to experiment with different approaches and make the system dynamically adaptive. This feature can be used e.g. for multilingual and other personalized applications.

The second important feature of Jaspis agents is that they do not communicate directly with other agents. Instead, they use shared information storage for *indirect communication*. They also keep all the relevant information in the shared storage. Since the Jaspis agents are selected by local managers, they are coordinated, but not controlled, and stateless, but not incapable of communicating and exchanging information. Both of these features are crucial: only small components can be distributed efficiently and the dynamic selection (adaptation) of components can be done efficiently only when the overall context is known.

4.2. System Level Adaptation: Evaluators

Jaspis supports adaptation at architecture level with components called *evaluators*. Evaluators are used to assist managers in choosing which agents are the most suitable for handling different tasks. Evaluators are *always* used when managers need to choose between agents. It is noteworthy that there is no single component which would decide how the interaction should proceed, as is the case in many other systems, such as [5]. When both the agents and the evaluators are compact, we can construct highly dynamic and adaptive systems.

Like agents, evaluators are compact and specialized in atomic tasks. When agents are evaluated, there are multiple evaluators involved. Different evaluators evaluate different aspects of the agents, and each one of them gives a score for each agent. Evaluators can access the shared system context (i.e. the system information storage), which means that they can utilize all the information the system contains, such as the dialogue context and the user model.

Evaluators can form a hierarchy, which means that higher-level evaluators operate from the base of lower-level evaluators. In this way, both detailed and overall evaluations can be performed. For example, higher level evaluators may emphasize the scores of lower-level agents in various ways. Different approaches, such as machine learning, can be used in evaluation functions, as presented in [3]. Evaluators can also be used to provide resource management for complex applications, where agents are shared between dialogues.

4.3. Coordination: Managers and Collections

Interaction management (coordination and synchronization) in the Jaspis architecture contains two layers. The topmost level contains a component called *Interaction Manager*. This facilitator type component handles the overall coordination in the system. Feature-wise, Interaction Manager is similar to HUB in the Galaxy-II architecture [10], and the Facilitator in the TRIPS architecture [1], but it acts more like a coordinator than a controller. In a nutshell, it is aware of the other components in the system and takes care of system bookkeeping and scheduling.

Interaction Manager gives the local managers possibilities to act based on the trigger events received from Information Manager. Events are based on the *triggers* registered by a local manager and its agents. Interaction Manager notifies triggered managers according to their *priorities*. Multiple triggers can become active during a single *transaction* and Interaction Manager is needed to solve the order in which the local managers are notified.

Other components in the system are grouped into *collections* which is the second interaction layer. Each collection contains components related to similar tasks. For example, agents used in dialogue tasks belong to a dialogue component collection. The collections can be seen as “modules” or “agent communities”. Each collection contains one local manager which coordinates agents inside that collection.

The Jaspis architecture supports an unlimited number of managers and does not limit the type of managers in any way. It is also possible to share components between the collections, i.e. agents can belong to more than one collection.

These features make it possible to handle multiple dialogues inside the same system in a modular way, and at the same time share a common system context between the dialogues. This is important in a ubiquitous system in which we may need hundreds, or even thousands of concurrent dialogues.

5. Discussion

The Jaspis architecture has been used in many speech applications, including the e-mail application Postimies/Mailman [13], a Bus-timetable information service [3], and a ubiquitous computing application Doorman [8]. The basic infrastructure has been flexible enough for most interaction tasks, such as handling errors efficiently by using agents [15]. The original architecture was not able to cope with concurrent dialogues, which is crucial in ubiquitous computing applications. For example, in the Doorman application there can be multiple users, and each of these users can have multiple dialogues going on. This complexity led to the development of a new version of the architecture. Currently, we are extending the Doorman system to take full advantage of the Jaspis^2 architecture for managing multiple concurrent dialogues.

One important thing to notice is that the Jaspis^2 architecture is a general infrastructure and it does not define how distributed and concurrent dialogues should be constructed. Neither does it force individual components to follow any predefined paradigms. For example, the dialogue components can be implemented by using any dialogue model, such as the dynamic dialogues provided by the agenda model [9]. Various different dialogue strategies and dialogue control models are used in the mentioned Jaspis-based applications, and still several components are shared between applications.

In the future, we will investigate what kind of methods can be used to share agents efficiently between multiple dialogues. We will also implement interfaces to access systems which use other infrastructures, such as Galaxy-II [10] or OOA [6].

6. Conclusions

In this paper, we have presented a new architecture to support distributed spoken dialogue applications. The Jaspis^2 architecture supports *distributed dialogues and components* by using small stateless agents. *Coordination* and *synchronization* are achieved with shared information management with a trigger and transaction mechanism and the use of managers. *Adaptability* is supported with use of evaluators that are used to select the agents for different situations. In all, the architecture offers a distributed and concurrent environment with system level adaptation, coordination, synchronization and shared system context management.

The architecture is used in many spoken dialogue applications and is freely available. Technology-wise, the architecture is based on Java and XML, and uses emerging standards for communication and information manipulation. For more information, see http://www.cs.uta.fi/hci/spi/Jaspis/.

7. References

[1] Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L. and Stent, A. "Towards Conversational Human-Computer Interaction". *AI Magazine*, 2001.

[2] Bird, S., Liberman, M. “A Formal Framework for Linguistic Annotation”. Speech Communication, 2002.

[3] Jokinen, K., Kerminen, A., Kaipainen, M., Jauhiainen, T., Wilcock, G., Turunen, M., Hakulinen, J., Kuusisto, J., Lagus, K. “Adaptive Dialogue Systems - Interaction with Interact”. *Proceedings of the 3rd SIGdial Workshop on Discourse and Dialogue, 2002*.

[4] Lemon, O., Bracy, A., Gruenstein, A. and Peters, S. "The WITAS Multi-Modal Dialogue System I". *Proceedings of Eurospeech 2001*, 2001.

[5] Luperfoy, S., Duff, D., Loehr, D., Harper, L., Miller, K., Reeder, F. "An Architecture for Dialogue Management, Context Tracking, and Pragmatic Adaptation in Spoken Dialogue Systems". *Proceedings of the Association for Computational Linguistics Annual Meeting*, 1998.

[6] Martin, D. L., Cheyer, A. J. and Moran, D. B. "The Open Agent Architecture: A framework for building distributed software systems". *Applied Artificial Intelligence: An International Journal. Volume 13, Number 1-2, January-March 1999*: 91-128, 1999.

[7] Matsusaka, Y., Fujie, S. and Kobayashi, T. "Modeling of conversational strategy for the robot participating in the group conversation". *Proceedings of Eurospeech 2001*: 2173-2176, 2001.

[8] Mäkelä, K., Salonen, E.-P., Turunen, M., Hakulinen, J. and Raisamo, R. “Conducting a Wizard of Oz Experiment on a Ubiquitous Computing System Doorman”. *Proceedings of the International Workshop on Information Presentation and Natural Multimodal Dialogue*, 2001: 115 - 119.

[9] Rudnicky, A. and Xu W. “An agenda-based dialog management architecture for spoken language systems”. *IEEE Automatic Speech Recognition and Understanding Workshop*: I-337, 1999.

[10] Seneff, S., Hurley, E., Lau, R., Pao C., Schmid, P., Zue, V. “Galaxy-II: a Reference Architecture for Conversational System Development”. *Proceedings of ICSLP98*, 1998.

[11] Stent , A., Dowding, J., Gawron, J., Bratt, E. and Moore R. "The CommandTalk Spoken Dialogue System". *Proceedings of the Thirty-Seventh Annual Meeting of the ACL*: 183-190, 1999.

[12] Turunen, M. and Hakulinen, J. “Jaspis - A Framework for Multilingual Adaptive Speech Applications”. *Proceedings of 6th International Conference of Spoken Language Processing (ICSLP 2000)*, 2000.

[13] Turunen, M. and Hakulinen, J. “Mailman - a Multilingual Speech-only E-mail Client based on an Adaptive Speech Application Framework”. *Proceedings of Workshop on Multi-Lingual Speech Communication (MSC 2000)*, 2000: 7-12.

[14] Turunen, M. and Hakulinen, J. “Agent-based Adaptive Interaction and Dialogue Management Architecture for Speech Applications” .*Proceedings of the Fourth International Conference TSD 2001*: 357-364, 200.

[15] Turunen, M. and Hakulinen, J. Agent-based Error Handling in Spoken Dialogue Systems. *Proceedings of the Eurospeech 2001*: 2189-2192.

[16] Zadrozny, W., Budzikowska, M., J. Chai, Kambhatla, N., Levesque, S. and Nicolov, N. “Natural Language Dialogue for Personalized Interaction”. *Communications of the ACM (CACM) Volume 43, Number 8*: 116-120, 2000.