

# Jaspis - A Framework for Multilingual Adaptive Speech Applications

Markku Turunen and Jaakko Hakulinen

Human-Computer Interaction Group  
University of Tampere  
Department of Computer and Information Sciences  
33014 University of Tampere, Finland

## ABSTRACT

We introduce Jaspis, an open framework for adaptive speech applications. Jaspis is designed to support distributed, highly context-sensitive applications that adapt to a user and an environment. Jaspis is especially designed for multilingual applications. In this paper we introduce the fundamental principles of the Jaspis framework, including shared information management, evaluator-based agent frameworks (for dialogue management and speech outputs) and input agent based communication management. A multilingual speech e-mail client, Postimies (Mailman), is described as an example application. Jaspis is written in Java and will soon be freely available for researchers and application developers.

## 1. INTRODUCTION

One of the main factors behind the difficulty of constructing advanced speech applications has been the lack of proper tools and frameworks. Several application protocol interfaces, rapid application development environments, standardization attempts and general markup-languages for high-level development have been introduced. However, most of these solutions are in-house tools, tied to particular products or custom API's, are very low-level and thus hard to learn and use or are too constrained for large-scale applications. Furthermore, most of the existing approaches do not provide support for multilingual, adaptive applications and are very naïve when handling speech outputs.

We introduce Jaspis (Java-based adaptive speech user interface development system), an open framework to build adaptive spoken dialogue applications. In Jaspis adaptive interaction methods are in focus – the whole framework has been designed to enable adaptation of system components, including output generation, input handling and dialogue management to adapt to different users and situations. Jaspis is designed especially with multilingual applications in mind. An example of a multilingual application is Postimies (Mailman), a multilingual speech e-mail client built on top of Jaspis. Technically Jaspis is based on Java and XML, standard platform independent solutions. We will provide Jaspis as a freely available framework to support spoken language research and development.

This paper is organized as follows. First, we introduce the overall architecture of Jaspis including how two of its key concepts, information management and evaluators work. Next, the input/output subsystem is presented. Following that, dialogue and presentation agents are described. Then we discuss briefly possible application areas and introduce Postimies (Mailman), an example application written on top of Jaspis. Finally, discussion is provided and ideas for future work are presented.

## 2. JASPIS ARCHITECTURE

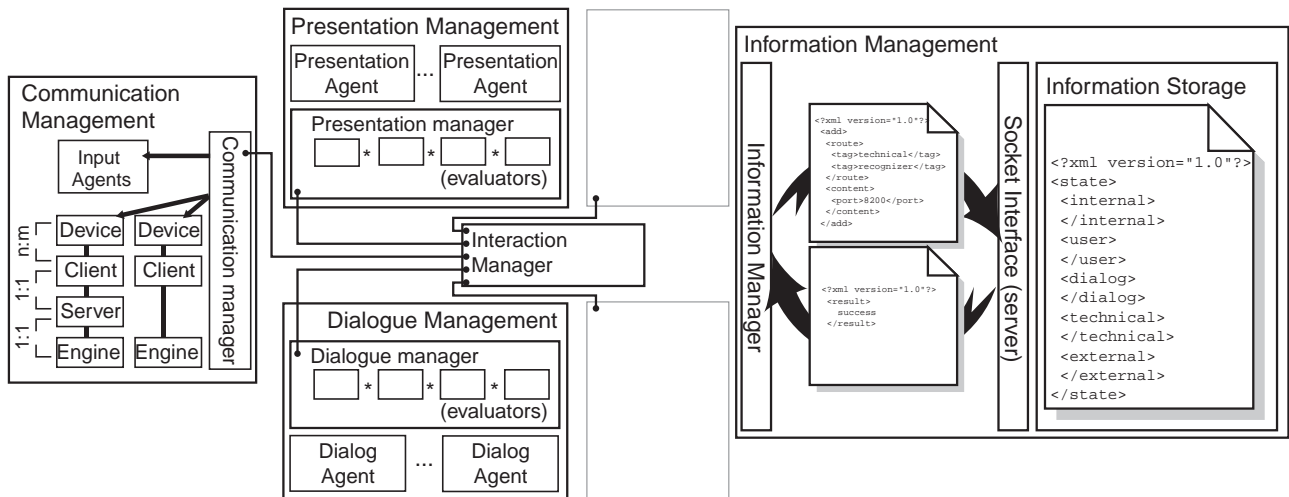
The overall architecture of Jaspis is presented in Figure 1. The main modules are *communication management*, *information management*, *presentation management* and *dialogue management*. All modules have one special component called *manager*. A manager is responsible for the coordination of the other components in a module. The *interaction manager* is responsible for managing the overall system coordination. The amount of managers (and modules) connected to the system is not limited – an application can register any number of managers. There can even be several managers for the same purpose. Other types of possible managers and components include for example database connections.

Program flow control is simplified. The interaction manager keeps a list of available managers and every time when a manager has finished, the information manager gives each of the other managers a possibility to take over the control. The interaction manager asks in a sequential order the other managers if they are willing to take control. In this way only one manager is controlling the program flow in a given situation. It is also the managers' own responsibility to decide when they should take the control. This decision should be based on the information found from the system's information storage. This kind of simplified and extensible architecture has been found to be efficient and it is also flexible enough for adaptive applications.

Jaspis has two features that are important for adaptive, distributed applications. First, all information is stored in a single (abstract) place called *information storage*. Information storage is accessed via the *information manager*. Since all parts of the system store all their information in a single place and each of them has access to all the information, there is no need to pass information between different parts of the system. The second important feature is the concept of *evaluators*. The evaluators provide flexibility for adapting to different situations. They are components that try to choose the best possible system component to handle a given situation. Next we introduce how information management works in Jaspis. Then we describe details of the evaluator concept.

### Information Storage

Jaspis framework is built following the principle of stateless sub-components, which has been acknowledged as a current trend in speech systems [3]. Stateless components are realized by having a single, shared storage for all dynamic information. This centralized *information storage* is accessible to all system components for storing and retrieving information and also for communicating with each other.



**Figure 1:** Jaspis Framework Overview

The information storage, which can be seen as a blackboard, can contain rather large amounts of information. Therefore it is important to keep the content of the storage well arranged. Jaspis arranges the information into a hierarchical tree structure. This way a single component can ignore information that is out of its interest by just ignoring uninteresting nodes in the information structure. Each component should know where information of its interest lies in the document structure. The required information can then be found by specifying its position in the hierarchy. In its current implementation Jaspis stores the content of the information storage in an XML document, which in its own structure supports hierarchical arrangement of information. Listing names of XML tags in a document then identifies a position of any information in the document. It should be noted that the information storage is not forced to be an XML document - it could be in any form, like for example a relational database.

We have defined basic groupings for the information to be stored in the information storage. In the root information is grouped in five categories. These are *internal context*, *dialogue*, *user model*, *external context* and *technical features*. Inside these categories information is divided into *application specific* and *generic information*. This grouping is briefly discussed in [1].

Other components of the system access the information storage through the *Information manager*. The information manager sends XML documents specifying the wanted operations (given by a component) to the information storage. The information storage then returns another XML document as a result to the query it received. These documents can be transferred over a network connection so that information storage can be located in a physically different computer and computer platform than the components that are using it. As XML documents are basically plain text strings, it is easy to write interfaces for accessing the storage from different environments and languages.

We can have different implementations of the information manager in different platforms all connecting to the same information storage using the XML interface. These different information managers can also provide different, perhaps easier interfaces to access the information storage than the powerful

but rather minimalist XML query language. For example, it is possible to build query-based interfaces that seek information based on logical queries rather than defining the exact path to the information.

### Evaluator concept

*Evaluators*, together with *agents* enable more adaptive systems. Jaspis uses similar evaluator-based architectures symmetrically in dialog and presentation modules. Both of these modules have a set of agents (presentation and dialog agents). Agents are used to provide the actual functionality of a module. There can be several possible agents, either for the same or other situations. Evaluators are used to select the best one out of several possible agents to do a task. The process of evaluating the agents is split into several small, preferably atomic evaluations. We have a set of small evaluators, each working on its own subject. Each evaluator gives a score for each of the possible agents under evaluation. Scores are between 0 and 1. If an agent receives 0 from an evaluator it is considered useless for the task in hand. The best score 1 means that this evaluator does not see any reason not to choose that particular agent. When the scores of all evaluators are multiplied we get a result between 0 and 1, and an agent with the best total score, as long as the score is more than 0, should be selected. Agents and evaluators are described in more detail in the following sections.

### 3. INPUT/OUTPUT ARCHITECTURE

Input/output architecture of Jaspis is based on *engines*, *servers*, *clients*, *devices* and *input agents*. The *communication manager* coordinates these components and is the main provider of information between the user, environment and the system. The architecture is distributed and (currently) works over TCP/IP networks using sockets. Next we describe each of the main components of this communication subsystem.

*Engines* are communication devices in a traditional sense and handle all input and output actions at low-level. Speech recognizers, speech synthesizers, telephony cards and different sensors are typical engines of speech applications. Engines are only components tied to particular implementations, protocols

and application programming interfaces. Jaspis supports currently several well-known devices, for example SAPI-compliant speech engines and Dialogic telephony cards. Although individual engines are tied to particular implementations, same kinds of engines provide a coherent interface to servers. For example, all kind of speech synthesizers can be controlled using synthesizer servers of the same type.

The next layer in the input/output structure is formed by *servers*, which besides providing a common interface to different engines also offer a distributed networked environment for speech applications. Servers communicate with *clients*, which can be situated in the same or different physical computer as servers. Servers and clients communicate using common protocols and methods. Currently we have implemented plain socket connections on top of TCP/IP. In the future we will explore the possibility of other methods like JavaSpaces. An application can have any number of servers and clients. A single client can be connected to multiple servers and vice versa. In this way complex, distributed systems that include many users and applications are possible. It is also possible to share certain components, like recognizers, between multiple applications. We have also built direct memory calls to speed up communication for situations where latencies should be minimized. In these cases engines are controlled directly by clients and can't be shared with other applications.

Engines, servers and clients operate on very concrete level and are not seen directly by an application. More abstract *virtual devices* interact between the user and the system in the application level. Devices are small and specialized for certain kinds of interaction tasks. For example, in a case of telephone key input Jaspis has a specialized DTMF device to take care of all telephone key inputs regardless of the context. We believe that it is efficient to separate different kind of inputs from each other. In this way it is possible to write more general solutions in other parts of the system.

Jaspis supports grouping of devices into meaningful units and higher-level devices. Simple devices can be seen as building blocks for more complex devices. As an example, a telephone recognizer device is combined from several devices including a telephone recorder device and a recognizer device. Since combining the devices can be done in runtime it is possible to construct dynamic devices that adapt to the current situation. Building simple and generic devices supports also reusability of components.

The *communication manager* handles the overall coordination of devices by performing basic tasks like starting devices when an input/output act is requested and stopping devices when needed. The operating mode of a device can be either single running or continuous. If a device is continuous it keeps running all the time and provides information continuously. Devices are running in a parallel manner and provide information via callback functions.

When the communication manager receives information from a device it passes that information to a set of input agents. *Input agents* interpret and conceptualize these raw information streams. They also coordinate different input streams including the coordination of multimodal issues. Input agents are invoked in a sequential order. First agents try to generalize and conceptualize information so that other agents can benefit from

this higher-level information. Input agents do not only process final inputs but they also process intermediate results from input devices and can give feedback (control information) back to the devices. They also decide when it is time to stop gathering more input and release the control back to the interaction manager.

Some input agents are totally application independent and some are tied closely to the application and/or to the domain. Input agents can be specialized for certain kind of tasks or inputs. Some agents can specialize for speech inputs, some for the coordination of devices and some for the interpretation of different aspects of inputs. This supports reusable and extendable interaction components. Input agents can access all the system's information including dialogue history and user profile. They can also utilize natural language components of the system. Information received from devices is stored in the system's information storage both as received from devices and in the conceptualized form produced by input agents. This information is utilized mainly by dialogue management.

## 4. DIALOGUE MANAGEMENT

Dialogue management contains three kind of components, dialogue agents, dialogue evaluators and one dialogue manager. *Dialogue agents* are basic communication units between a user and a computer. A dialogue flow is a sequence of these units. Jaspis includes support for different agents that are suitable for different situations. In this way we can build reusable dialogue components for situations like error handling etc. It is also possible to build several different agents for a single purpose – this supports adaptive applications like multilingual ones.

*Dialogue manager* is a component that tries to choose the best possible dialogue agent for the current situation. It evaluates available dialogue agents using *dialogue evaluators* and chooses the most suitable one (like described earlier). The choosing of the most suitable agents is based on the overall situation, which contains current inputs, dialogue context and user preferences. In a very simple case there is only one agent suitable for a particular situation. In order to provide more flexibility and choices to the interaction we can add alternative agents for the same situations. Dialogue evaluators, like all other evaluators, are specialized for different kind of tasks – some of them are application specific and some are generic.

All evaluators can change the fundamental style of communication from one to another (for example, a mixed-initiative style to a system-initiative style). This means that if we have dialogue agents with different styles, like system-initiative and mixed-initiative ones we have an opportunity to switch between dialogue strategies. This model is motivated from the findings that single dialogue control strategies are not suitable for all situations. For example, mixed-initiative and system-initiative dialog handling strategies are found to have different kind of benefits and drawbacks [5]. Thus, a framework for adaptive applications should support multiple input and dialog handling strategies.

## 5. PRESENTATION FRAMEWORK

One of the key features of adaptive and especially multilingual speech applications is the ability to present flexible speech outputs. In order to provide dynamic speech outputs we use

*presentation agents* [1]. Presentation agents convert conceptual system messages produced by the dialogue management to highly context-sensitive, dynamic, adaptive, multilingual and prosody rich speech outputs which can then be sent to a speech synthesizer by the communication manager. This way speech outputs can be designed to be more intelligible and pleasant for the user [2].

*Presentation manager* works symmetrically to dialogue manager: it controls the process of constructing outputs by using evaluators to choose the best possible presentation agent for the current situation. *Presentation agents* can be very specialized in the type of messages they can generate. In this way we can have special agents for each type of messages in the system. Also we can have several presentation agents for each type of message, each producing natural language messages in different styles, form or even languages.

*Evaluators* try to find a presentation agent that can generate the most suitable message to be presented to a user. The agent should be able to present this message in such a way that it is easy to understand and pleasant to listen to. Evaluators can use all the information from the information storage to make the evaluation. In this way a user model, for example, can be utilized to choose an agent that speaks in a way the user prefers. Evaluators have access to a set of parameters describing the qualities of available agents and these values are compared to the current situation. Presentation agents also have access to the information storage and can therefore generate context sensitive output messages and for example avoid unnecessary repetitions.

When an agent has finished generating the natural language message, it stores the message in the information storage with speech synthesis control mark-up. Then the message is ready to be sent to a speech synthesizer by the communication manager.

## 6. APPLICATIONS

Using Jaspis we have constructed Postimies (Mailman), a multilingual telephone-based speech-only email client [4]. Postimies is currently partly bilingual: it uses Finnish and English for reading e-mail messages. This includes multilingual messages, i.e. messages that contain both languages. Communication between the system and the user is either in English or Finnish – mixed bilingual dialogue is not currently allowed. Jaspis is built with applications like Postimies in mind. In fact, one of the motivations behind the construction of Jaspis was the lack of proper tools to build multilingual applications like Postimies.

Postimies is a working prototype system and currently in everyday use both at our laboratory and also by users from outside laboratory. We are planning to carry out a large-scale user study soon. Jaspis will also be a base framework in several other speech user interface projects. Further applications are under development and they will address issues like dialogue adaptation in public information systems and needs of speech-enabled ubiquitous applications.

## 7. SUMMARY

We have presented a framework for constructing adaptive speech applications. The main focus of Jaspis is in interaction

methods that support adaptive spoken dialogue applications, especially multilingual ones. We have introduced how the components of Jaspis provide together a base for adaptive, distributed applications. Postimies (Mailman) is an example application written on top of Jaspis. It uses adaptive features of Jaspis to produce multilingual communication in the e-mail domain.

Currently the Jaspis framework is implemented as described here. The initial release should be available for public use very soon at the address <http://www.cs.uta.fi/research/hci/SUI/Jaspis/>. Jaspis is written in Java and includes a comprehensive API for application construction. Jaspis is written mostly in platform independent Java code but third-party components like speech recognizers require Windows NT specific code.

Future work includes support for other servers like speaker recognizers and other speech recognizers and synthesizers. We will also add more support for ubiquitous computing to the future releases of Jaspis. Since Jaspis will be used in several other projects as a base framework we will add other features as needs arise. Also we like to see how other researchers and application developers experience Jaspis and we hope to get feedback from them to improve the Jaspis framework.

## 8. ACKNOWLEDGEMENTS

This work was supported by the Academy of Finland (project 163356) and by the Technology Development Center.

## 9. REFERENCES

1. Hakulinen, J., and Turunen, M. "Presentation Agents for Speech User Interfaces". In CHI 2000 Extended Abstracts, Den Haag, The Netherlands, April 1-6, 2000, 115-116.
2. Hakulinen, J., Turunen, M., and Rähkä, K.-J. "The use of prosodic features to help users extract information from structured elements in spoken dialogue systems". In Proceedings of ESCA Tutorial and Research Workshop on Dialogue and Prosody, Eindhoven, The Netherlands, September 1-3, 1999, 65-70.
3. Potamianos, A., Kuo, H.-K., Lee, C.-H., Pargellis, A., Saad, A., and Zhou, Q. "Design Principles and Tools for Multimodal Dialog Systems". Proceedings of ESCA tutorial and workshop Interactive Dialogue in Multi-Modal Systems, 1999.
4. Turunen, M., and Hakulinen, J. "Mailman - a Multilingual Speech-only E-mail Client Based on an Adaptive Speech Application Framework". To appear in Proceedings of Workshop on Multi-Lingual Speech Communication, Kyoto, Japan, October 11-13, 2000.
5. Walker, M., Fromer, J., Fabbriozio, G., Mestel, C., and Hindle, D. "What can I say?: Evaluating a spoken language interface to Email". In Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems, 1998: 582-589.