EXTENSIBLE SPEECH RECOGNITION SYSTEM USING PROXY-AGENT

Teppei NAKANO, Shinya FUJIE, Tetsunori KOBAYASHI

Department of Computer Science, Waseda University

ABSTRACT

This paper presents an extension framework for a speech recognition system. This framework is designed to use "Proxy-Agent," a software component located between applications, speech recognition engines, and input devices. By taking advantage of its structural characteristics, Proxy-Agent can provide supplementary services for speech recognition systems as well as user extensions. A monitoring capability, a feedback capability, and an extension capability are implemented and presented in this paper. For the first prototype, we developed a data collection application and an application control system using Proxy-Agent. Through these developments, we verified the effectiveness of the data collection capability of Proxy-Agent, and the framework extension capability.

Index Terms— speech recognition systems, software architecture, standardization, extended capability, rapid application development

1. INTRODUCTION

A number of speech recognition systems and applications can benefit from general function enhancement capabilities, such as a monitoring function of user utterances, a model data upgrade function via the Internet, and so on. In today's speech recognition applications, however, there are no common frameworks to install these extensional functions. As a result, developers have to implement additional functions ad hoc, despite a system's general versatility.

Today's speech recognition applications developments can be classified into two approaches. One, called the "architecture based approach," is based on the architecture for spoken dialogue systems, such as DARPA Communicator Architecture [1]. The second, called the "library based approach," uses a Speech Recognition API directly, such as Windows SAPI [2] and Java SAPI [3]. The architecture based approach is usually employed to implement applications whose design focus is a spoken dialogue interface. In this approach, developers can use some frameworks provided by the architecture, such as natural language processing and dialogue control. However, not all applications can be designed around the speech interface and many of these architectures are based on distributed technologies. This approach, therefore, can be employed only in limited application developments, such as a call center and some enterprise applications. Conversely, the library based approach is employed to add a speech interface easily to existing applications, such as car navigation systems. A speech recognition engine is regarded as a library in this approach. This approach, however, requires developers to design entire speech interfaces. However, the development a user-friendly speech interface is not easy, and the frameworks required for this approach are not as well understood as those employed in the architecture approach. In this research, therefore, we consider the framework needed to develop a speech recognition application using the library-based approach.

One of the most critical issues reported in speech application developments in the library based approach is the problem of mismatches between the usage expected by the developers and actual usage by the users [4]. A solution to this problem is needed. Both the addition of a monitoring function of actual users' behavior and installation of a common speech interface are possible solutions. In order to run the monitoring function efficiently, a feedback function to developers is also needed. In addition, a delivery function to the runtime environments of the upgraded model may also be desired. In a traditional approach, however, direct extensions to applications or speech recognition engines are required, since an application can reference an engine directly. As a result, the applicable scope of these extensions is very limited. At the same time, installing a standard interface is also difficult since there are no generic frameworks that share common knowledge and implementations.

In this research, we designed and implemented speech recognition systems such that an application can reference a speech recognition engine indirectly. To support such an indirect reference, we propose a framework that uses a software component, Proxy-Agent. Proxy-Agent is designed to provide a plug-in-based extension framework. In this model, we regard the speech recognition engine not as a single library but as a single plug-in. By using Proxy-Agent as a standard platform of general function extensions in speech recognition systems, we aim to create a new market for speech recognition components, models, frameworks, and peripheral functions.

In this paper, we first describe Proxy-Agent and its design in Section 2. Section 3 describes the architecture of Proxy-Agent. In Section 4, we explain how Proxy-Agent is implemented, and explain the monitoring capability in Sec-

This work was supported by the METI Project "Development of Fundamental Speech Recognition Technology"

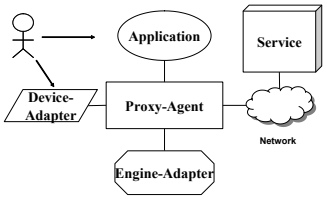

Fig. 1. Proxy-Agent Overview

tion 5. Section 6 describes applications that use Proxy-Agent. In Section 7, we discuss the advantages and disadvantages of Proxy-Agent. We present our conclusions in Section 8.

2. BASIC APPROACH

Proxy-Agent is a software component located between application programs, speech recognition engines, and input devices, and takes care of their collaborations (Fig. 1). It provides extension capabilities for cross-application and cross-engine functions as well as for application-dependent and engine-dependent functions. When using Proxy-Agent, an application controls a speech recognition engine via the interface provided by the Engine-Adapter. Engine-Adapter is a virtual speech recognition engine composed of more than one plug-in component; the speech recognition engine itself is one of the plug-in components. Since each speech recognition engine usually consists of more than one module, such as an acoustic model and a language model, these constituents can also be plug-ins. Engine-Adapter retrieves input data via the Device-Adapter managed by Proxy-Agent. Device-Adapter is a data provider that encapsulates the data acquisition logic via an actual input device, and actually performs the task.

As mentioned above, the primary goal of Proxy-Agent is to provide a standard platform of general function extensions in speech recognition systems. The remainder of this section describes the key features of the platform, and why these features are important to achieve this goal.

Extension capability is the basic and most important feature of this platform. Each extension is a plug-in to Proxy-Agent. Examples of extensions are the speech recognition engine, constituents of the speech recognition engine, frameworks using the speech recognition engine, and application implementations dependent on speech recognition.

Networking capability is embedded in the platform to communicate with server services. Since Proxy-Agent is designed independently with respect to applications and engines, comprehensive server services can be provided. Because it is combined with other features in the platform, this platform needs to support the "network effect by default" [5], which means the platform is designed to enable both users and developers to contribute to the growth of the framework.

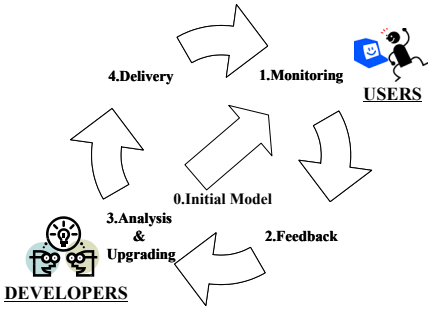

Fig. 2. A positive feedback loop. 0. Proper initial model is be provided. 1. User behavior on actual execution environment is monitored. 2. Usage information is fed back to the developer side. 3. Developers analyze usage information to detect mismatches, and upgrade models to decrease mismatches. 4. Upgraded models are delivered to the client side.

Monitoring capability is supported by the platform with feedback capability using the networking capability. As a solution to the mismatch problem mentioned above, a development paradigm based on continuous improvement is supported in this platform. In this paradigm, we presume that mismatches always exist in all speech applications. Therefore, we provide a framework by which developers can obtain the behavior of actual users at runtime. By taking advantage of its structural characteristics, Proxy-Agent can monitor the signals from applications to speech recognition engines as well as the input-output information from these engines, all of which are critical to achieve the paradigm.

Upgrade capability can be used to deliver and install upgraded plug-in components in runtime environments using the networking capability. Combined with the monitoring capability, this feature is necessary to achieve continuous improvement. This positive feedback loop between engine developers, application developers, and users is one of the most essential features of speech recognition systems (Fig. 2).

Sharing capability of linguistic resources, component implementations, and framework implementations is a side effect of the framework and is abundantly needed in the growth of speech recognition research. This feature is also important to expand common frameworks, such as a framework to generate a standard speech interface, because all frameworks can be executable in the platform.

3. PROXY-AGENT ARCHITECTURE

Proxy-Agent is designed as an infrastructure to execute plug-ins and provides an extension point between applications and

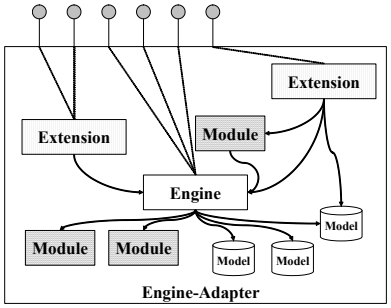

Fig. 3. Engine-Adapter Overview: The interface is defined as an aggregation of plug-in interfaces

engines. This extension point is used to install application / engine independent features or frameworks into the system. This section describes the architecture of Proxy-Agent.

3.1. Engine-Adapter

Engine-Adapter is defined as a virtual speech recognition engine. It provides an extension point for applications and engines. This extension point is used to install application / engine dependent features as well as independent features. The granularity and roles are varied according to the plug-ins, which include the following: constituent units of a speech recognition engine such as an acoustic model plug-in, a voice activity detection (VAD) functional module plug-in, as well as the speech recognition engine itself with a functional extension plug-in (Fig. 3). The application programming interface of Engine-Adapter is defined as an aggregation of the plug-ins' callable interfaces. An application can call the interface via Proxy-Agent freely.

Functional extensions include plug-ins to 1. filter input / output data, 2. intercept method calls, 3. define new methods to wrap other method calls, 4. encapsulate operational logic on the speech recognition engine. Some of these plug-ins, usually prepared for each application, are tightly coupled to the engine. These tightly coupled plug-ins can help keep an application and an engine loosely coupled. Since all of these extensions potentially can be shared with other applications, this scheme can also help share knowledge.

Engine-Adapter only provides fundamentals such as the means to handle an input device, a plug-in scheme and a component load scheme. The definitions of the component interfaces are left to the engine developers. At the same time, however, since all components follow the same plug-in rules, developers are free to use pre-defined components. According to these rules, we expect effective plug-ins to be shared as a de facto standard by preparing a common repository for all of these plug-ins.

3.2. Device-Adapter and Recognition Process Sequence

Device-Adapter is defined as a plug-in to acquire binary data via a runtime environment. It supports buffering, and it begins

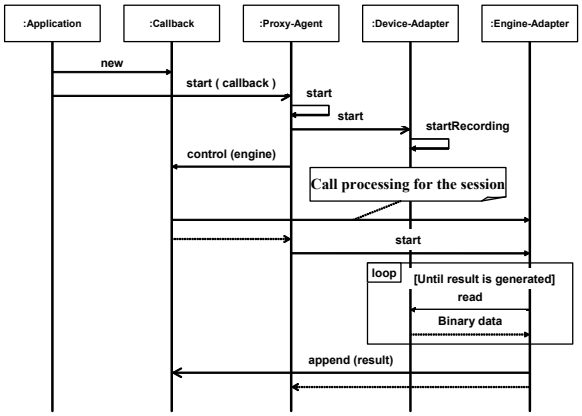

Fig. 4. Sequence Diagram for the recognition process: 1. Proxy-Agent sends a start signal to Device-Adapter when it receives a start signal from the application. 2. Proxy-Agent calls for control to the Callback Object specified by the application. 3. Callback Object controls Engine-Adapter (such as loading a grammar construction). as defined by the application. 4. Proxy-Agent sends a start signal to Engine-Adapter. 5. Engine-Adapter starts recognition with the reading data acquired via Device-Adapter

to acquire data into the buffer when it receives a start signal by Proxy-Agent. Engine-Adapter reads the buffered data via Device-Adapter when the Device-Adapter's recognition process is started.

In order to compensate for the shortcomings of the redundant layer in this architecture, the framework precisely maintains the timing of the start signal. Using Device-Adapter's buffering capability and a callback framework, all data after the start signal can be obtained, including data during the intermediate preparation time (Fig. 4).

4. IMPLEMENTATION

4.1. Implementation Platform

As a platform to implement Proxy-Agent, we use Eclipse RCP (Rich Client Platform) [6]. Eclipse RCP is the platform that underlies Eclipse IDE for building and deploying rich client applications. The Eclipse RCP platform, based on the OSGi [7] platform, provides highly extensible plug-in architecture that assures interoperability of applications and services delivered and managed via networks.

Based on the Eclipse RCP platform, all constituents such as Proxy-Agent and Engine-Adapter are implemented as Eclipse plug-ins. As the result, the Eclipse IDE environment can be used to implement Proxy-Agent's plug-ins. In addition, the plug-in management features implemented on networks based on OSGi can also be used to maintain Proxy-Agent's plug-ins.

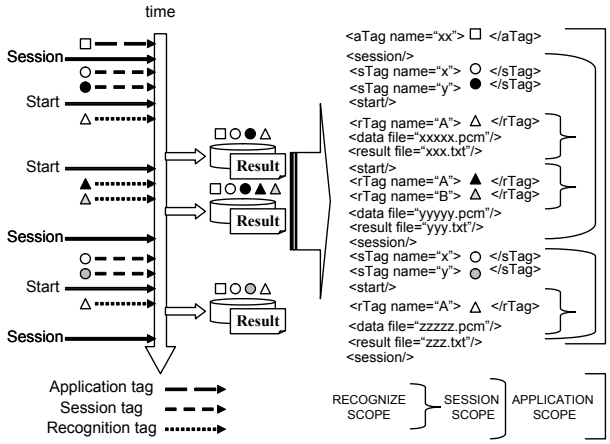

Fig. 5. Tags to Speech Segments

4.2. Engine-Adapter Implementation

As an implementation of Engine-Adapter, we developed the Sphinx-4 [8] Engine Plug-In. Sphinx-4 is a Java based open source speech recognition engine that is highly reconfigurable. We extended the Sphinx-4's resource loading framework such that classes and resources can be loaded from other plug-ins on Proxy-Agent. Sphinx-4's model loader for the HTK (Hidden Markov Model) format acoustic model was also developed as a plug-in.

4.3. Device-Adapter Implementation

As an implementation of Device-Adapter, we developed an audio input plug-in for a microphone. The buffering capability was implemented using a cyclic queue. In addition, we provided constant recording capability to support the recognition using past data before the start signal. This feature can be used when a user starts the voice input before sending a start signal, such as pushing the talk button after the utterance has started.

4.4. Proxy-Agent Application Programming Interface

We prepared the following two types of communication protocols between applications and Proxy-Agent: the client-server type and the application-plug-in type. The client-server type uses asynchronous message passing based on local TCP/IP, and the application-plug-in type uses a Java API directly. As a client application, C++ and Java are available.

5. MONITORING CAPABILITY IMPLEMENTATION

An autonomous data collection framework was implemented. The collected data are the data actually read by the speech recognition engine. The collected data are stored in the repository managed by Proxy-Agent. The endpoint of the data is determined by the engine or application stop message. When Proxy-Agent receives the results from the engine, it stores the data in the same time it is sending them to the application. It is also possible to tag the speech segments via API[9]. Based on the scope, segments are tagged as shown in Fig. 5. Developers can use these tags as keys to find needed segments for analysis.

The data upload framework is implemented in the Proxy-Agent as well as in server programs to receive data. In Proxy-Agent, one management thread is launched to compress the data stored in the repository, and to upload and to delete data. We implement Proxy-Agent and Receiver to communicate to each other with SOAP (Simple Object Access Protocol; XML based RPC). Since Proxy-Agent can be deployed in various network locations, this framework supports the RESUME function. With this feature, Proxy-Agent doesn't have to restart uploading from the beginning of the data each time.

6. APPLICATION DEVELOPMENTS

Using the Proxy-Agent prototype described in the above section, we developed the following two applications: the data entry application and the voice control application. Through these developments, we verified the monitoring capability and Engine-Adapter's extension plug-in capability.

6.1. Data Entry Application

A data entry application for travel information was developed. This application was developed to investigate how voice inputs changed when the user interfaces were changed. Four comparative interfaces were prepared. Each interface had ten input fields, and a network grammar was assigned to each input field. Speech recognition was performed to fill the field.

The application was developed as a client-server type. The Sphinx-4 Engine Plug-In and Network Grammar Loading Extension were plugged in to Engine-Adapter. All models and grammar definitions were packed into Engine-Adapter. In addition, we used the constant recording capability on the Device-Adapter and configured it to obtain voice input from 200 [ms] before the start signal.

Experiments were performed at the university cafeteria. A total of 908 speech segments was collected from 78 subjects using the monitoring function of Proxy-Agent. The collected speech segments were analyzed in terms of the occurrence frequency of fillers and the change of speaking style. As results of the analysis, the following facts were observed: 1. The developer's assumption was not correct (a well-designed user interface still cannot reduce fillers; 38data actually contained fillers). 2. Constant recording capability was not necessary in this application (no subject started speaking before push the button). 3. Some users read out the label of the field in some interfaces (this was not expected behavior). Through these tests, it was verified that effective information was easily collectable via the monitoring capability.

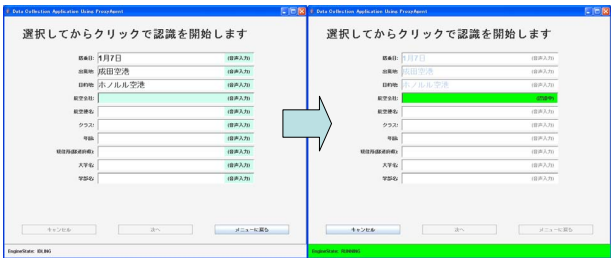

Fig. 6. Data Entry Application Overview: (left) The user selecting the field to input, (right) Voice input processing.

6.2. Voice Control Application

As a prototype of the new speech user interface, we developed a voice control application using Proxy-Agent. This application was originally developed as a standalone application to propose and evaluate the interface. This application was then redesigned to extract the framework for building the proposed speech interface. In order to solve the "what should I say" problem, the proposed interface used "functional structure," a tree structure of executable functions of the system, and "continuous keyword input" to access the command [10].

The application was designed as an application-plug-in type. Sphinx-4 Engine Plug-In was plugged in to Engine-Adapter. The extension was designed to depend on the Sphinx-4 Engine and to require "functional structure XML" from the Engine-Adapter. The grammar and dictionary were generated from the XML by this plug-in. An acoustic model was plugged in separately. Two APIs were defined: a method to specify the current context of the tree and a method to retrieve a list of path candidates from the word result. These APIs were exposed to the client. Fig. 7 shows the result of the redesign. Fig. 8 shows the dependency of the developed plug-in at its selection dialog on Eclipse.

Through this development, the pre-developed speech recognition application was successfully redeveloped based on the Proxy-Agent architecture. Since the developer had never developed with the Eclipse plug-in development environment, it took around 40 hours for the construction. As a result of the development, however, the target feature was successfully extracted and an executable framework plug-in was developed. It was also verified that this extension was feasible by adding the acoustic model plug-in and functional structure XML.

7. DISCUSSION

In this section, we discuss the advantages and disadvantages of using Proxy-Agent compared to other approaches.

First of all, we discuss the disadvantages of using Proxy-Agent compared to the direct reference to the engine. In order to use Proxy-Agent, additional time and resources are required. The needed time is for learning, and the needed resources are CPU and memory. Usually, most developers need the time for learning because Proxy-Agent is not well known.

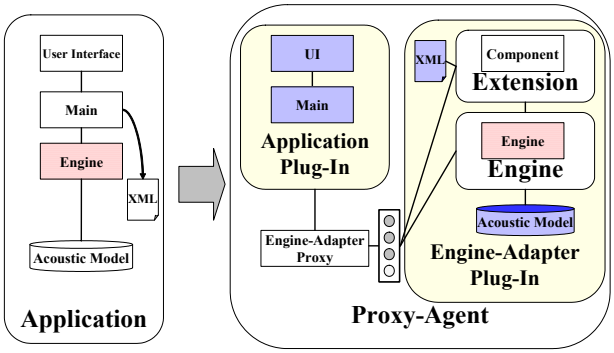

Fig. 7. System redesign as an Engine-Adapter Plug-In

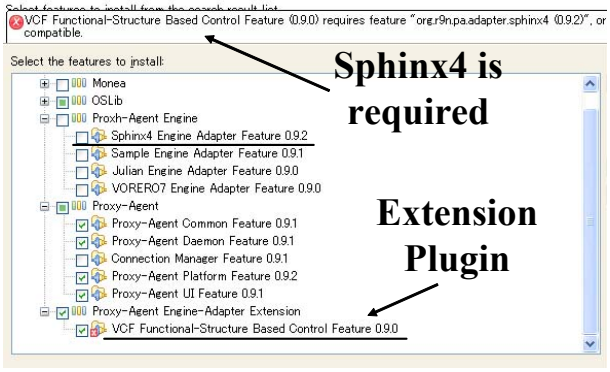

Fig. 8. Update Site for the Plug-Ins; "Sphinx4 engine is required" to install the plug-in is indicated.

This means that it may not be suitable to use Proxy-Agent as a first prototype development if the prototype is very simple. However, when the prototype needs peripheral functions, such as the monitoring and buffering functions in Device-Adapter, the learning costs would be justified. Moreover, in terms of production, network capabilities would be necessary for the applications. Of course, it is possible to develop these functions into the application [11]. However, the "network effect" is less promising. In terms of resources, Proxy-Agent requires relatively heavy resources because it uses Eclipse RCP on Java. Certainly, it is not possible to use the current Proxy-Agent in lightweight devices, such as handheld devices. However, because an embedded version of Rich Client Platform is being developed [12], it will be possible to use this platform in the near future.

In contrast, maintainability may be much superior when Proxy-Agent is used. One of the most distinct features in Proxy-Agent is its virtual speech recognition engine feature, to which developers can plug-in their application-dependent implementations. Using this feature, application developers can encapsulate all of the engine-dependent implementations. As a result, an application and an engine can be loosely coupled, which is a better approach in terms of maintainability. Without Proxy-Agent, an adapter pattern or wrapper pattern

may be used. In this approach, the application developer defines a common interface internally, and wraps the engine to support the interface. This approach is also common in other speech recognition framework developments [13]. This approach, however, requires more implementations by the developers. Some people might say using AOP (aspect-oriented programming) [14] is a better solution. Certainly, the feature to intercept an application call to the engine is a similar idea. However, the capabilities provided by Proxy-Agent are not just an interception: it controls the data flow of the device adapter, manages plug-ins, and so on. Therefore, AOP is not enough to replace the Proxy-Agent style of programming.

Finally, we consider the use of Proxy-Agent in spoken dialogue systems in distributed environments. In this case, there are some popular architectures, such as Open Agent Architecture [15] and DARPA Communicator Architecture [1]. In most of these architectures, a speech recognition engine is defined as a single constituent of the architecture, such as the agent [15] and server [1]. Applications, therefore, don't have to manage the engine directly. Moreover, some architectures, such as Jaspis [16], also have monitoring or data collection functions. However, they don't provide the capabilities supported by Proxy-Agent, such as extension capability and upgrade capability. This means that it is possible to replace a speech recognition agent or server with our Proxy-Agent based the speech recognition system, if developers need these functions.

8. CONCLUSIONS

In this paper, we presented an extension framework for a speech recognition system based on the software component called "Proxy-Agent." Proxy-Agent was designed to be located between application programs, speech recognition engines, and input devices, and Proxy-Agent takes care of their collaborations. All of extension, networking, monitoring, upgrade and sharing capabilities were introduced as key features of this platform. In particular, to focus on the problem of mismatches, we designed the monitoring capability as well as other key capabilities. An implementation based on a highly extensible plug-in architecture, Eclipse RCP, was described. Based on the application developed using the prototype, we verified that effective information was easily collectable via the monitoring capability, and plug-in development was feasible based on the Engine-Adapter's extension capability. In conclusion, Proxy-Agent based speech recognition systems provide a good solution toward the mismatch problem. As future work, we will develop larger applications and perform longer-term operation tests.

9. REFERENCES

[1] Alan Goldschen and Dan Loehr, "The role of the darpa communicator architecture as a human computer interface for distributed simulations," in *Spring Simulation Interoperability Workshop (SIW)*. Simulation Interoperability Standards Organization (SISO), 1999.

[2] "Windows sapi," `http://research.microsoft.com/research/srg/sapi.aspx`.

[3] "Java sapi," `http://java.sun.com/products/java-media/speech/index.jsp`.

[4] S.Furui and et al., "Development of practical speech recognition systems," Tech. Rep. 100007350, Achievement report of NEDO's project, 2006.

[5] Tim O'Reilly, "What is web 2.0: Design patterns and business models for the next generation of software," `http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html`.

[6] "Eclipse rcp," `http://www.eclipse.org/rcp/`.

[7] "The osgi alliance," `http://www.osgi.org/`.

[8] Paul Lamere and et al., "Design of the cmu sphinx-4 decoder," in *EUROSPEECH 2003*, 2003.

[9] Teppei NAKANO, Akira UMEMOTO, Shinya FUJIE, Tetsuji OGAWA, and Tetsunori KOBAYASHI, "Efficient monitoring capabilities of user behavior for speech recognition applications based on proxy-agent architecture," Tech. Rep. 2007-SLP-65, IPSJ SIG, 2007.

[10] Tomoyuki KUMAI, Teppei NAKANO, Tetsunori KOBAYASHI, and Yasushi ISHIKAWA, "A proposal and evaluation of new speech user interface based on functional-structure," Tech. Rep. 2007-SLP-67, IPSJ SIG, 2007.

[11] HARA Sunao, MIYAJIMA Chiyomi, ITOU Katsunobu, and TAKEDA Kazuya, "Speech data collection and evaluation by using a spoken dialogue system on general purpose pcs," Tech. Rep. 2006-SLP-64, IPSJ SIG, 2006.

[12] "Eclipse embedded rcp," `http://www.eclipse.org/ercp/`.

[13] Savitha Srinivasan, "Design patterns in object-oriented frameworks," *Computer*, vol. 32, no. 2, pp. 24–32, 1999.

[14] Gregor Kiczales and et al., "Aspect-oriented programming," in *European Conference on Object-Oriented Programming*, 1997, vol. 1241, pp. 220–242.

[15] Adam Cheyer and David Martin, "The open agent architecture," *Autonomous Agents and Multi-Agent Systems*, vol. 4, no. 1, pp. 143–148, March 2001.

[16] M. Turunen and et al., "An architecture and applications for speech-based accessibility systems," *IBM SYSTEMS JOURNAL*, vol. 44, no. 3, pp. 485–504, 2005.