5-1-2001

Intelligent Voice Email Agent: A Multimedia Solution

Xuecheng Wang
University of Nebraska at Omaha

Intelligent Voice Email Agent – A Multimedia Solution

Presented to the

Department of Computer Science

and the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

University of Nebraska at Omaha

by

Xuecheng Wang

May 2001

THESIS ACCEPTANCE

Acceptance for the faculty of the Graduate College,
University of Nebraska, in partial fulfillment of the
requirements for the degree Master of Science in Computer Science,
University of Nebraska at Omaha.

Committee

Peter Wolcott 4/16/2001

[illegible] 4/16/2001

[illegible] 4/18/01

Chairperson [illegible] 4/16

Date 4/16/2001

ABSTRACT

Intelligent agent theory is an important concept in artificial intelligent area. An intelligent agent, in a nutshell, is an intelligent program that uses agent communication protocols to exchange information for automatic problem solving, performing specific tasks on behalf of their users. Our objective is to investigate what an intelligent agent consists of and to implement several important aspects of it. In particular, we are interested in an intelligent agent that is able to take care of the incoming messages while the user is concentrating on some other duties. We develop an agent-based design framework and implement an intelligent agent system - voice email system that monitors incoming emails for us while we are surfing the Internet. A special feature of this system is that the agent reads any new messages for users. This initiative is based on the perception of real world needs and the academic research development. Based on what we have done, we can extend our agent capabilities. For example, two mail agents should be able to communicate each other to achieve more complicated task.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis we explore intelligent agent theory, an important concept in artificial intelligent area, by designing an intelligent voice e-mail agent system. Such kind of work extends intelligent agent theory scope to multimedia mechanism that is an innovative work in this area. Our work mainly covers the explorations of several aspects of intelligent agent theory and the implementations related to our intelligent agent system architecture. In this introductory chapter, we discuss our initiatives, our approaches, and give an overall idea about what our work is like. The more detailed discussions are presented in the following chapters.

1.1 The Initiatives – What We Do

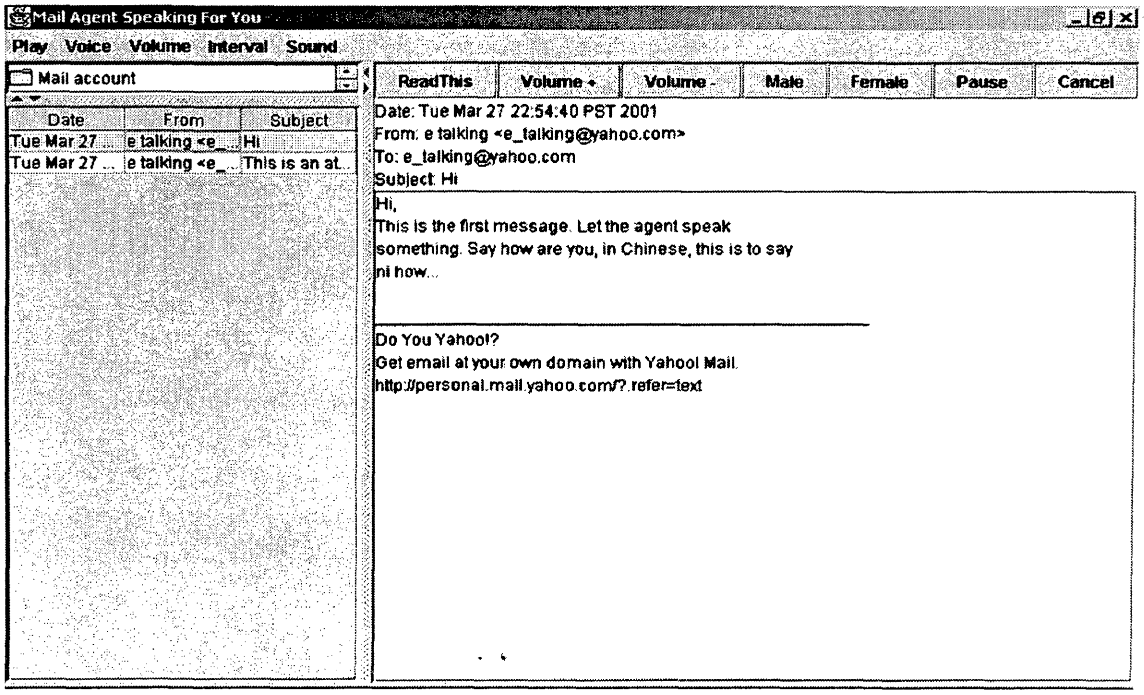
Since the invention of computer, we have dreamed computer programs automatically perform boring and complex tasks for us. With the development of artificial intelligence (AI) research, the intelligent agent (IA) concept is proposed and it seems that IA can accomplish this task for us.

What is an intelligent agent? A discussion of various definitions is provided in Chapter 2. An agent, according to the *Webster's New World Dictionary*, is "a person or thing that acts or is capable of acting or is empowered to act, for another". An intelligent agent (IA), in a nutshell, is an intelligent program that uses agent communication protocols to exchange information for automatic problem solving, performing specific tasks on behalf of their users. Specifically, the initiatives for utilizing intelligent agents are to simplify distributed computing and overcome user interface problems. Such kind of program is distinguished from other types of software program by its independent properties, therefore the program is capable of completing complex assignments without intervention.

The objective of this thesis is to investigate what an intelligent agent consists of and to implement several important aspects of it. In particular, we are interested in an intelligent agent that is able to take care of the incoming messages while the user is concentrating on some other duties (such as debugging or running a computer program, surfing the Web pages, accessing the databases, as well as others). With the help of an intelligent agent, the user is able to focus on his/her own interests while not delay the handling of any important messages, because the agent may make decisions for the user based on a set of instructions given by the user ahead of time. The users only handle emails or phone messages that deserve immediate attention. This scenario imposes tremendous challenges for the system design. For example, how to make the agent identify what are important messages, how to implement a set of instructions that agent has to follow, how to synchronize user's other duties with the agent's background activities etc. Due to the complexity of this problem, in this thesis, we only deal with a simplified scenario, namely, how to use agent-based techniques to build a voice email system that monitors incoming emails for us while we are surfing the Internet. A special feature of this system is that the agent reads any new messages for users. Therefore users can focus on their own duties while not delay the handling of any important messages. Users are informed of the content of the messages that they want to know. In order to achieve this goal, we first develop an agent-based design framework and then implement it. With the completion of the agent, it connects to mail server, fetching all the messages in the folder. All the messages in the folder are shown in a graphic user interface, therefore users can go through all the messages. If users want to listen to any message, just click that message and select speak function, then the agent reads for users. As long as the agent is active, it checks if there are any new messages in the folder. If there are any, the agent reads the message if user give instructions to agent. It identifies various types of messages such as plain/text, multipart and nested message. A visual notification is also shown on the screen. With both visual and voice notifications, our agent becomes a multimedia intelligent agent. We are able to set the agent preferences by selecting voice type (either male or female), selecting the age (young or aged), adjusting the volume, deciding how long of interval to check new messages etc. The intelligent agent system, in fact, transforms any text email messages to voice via speaker. This initiative is based on the perception of real world needs and the academic research development. The idea described here has not been explored in existing literature, and may find a wide range of applications because its functions are realized via audio.

1.2 The Approaches – How We Do

We designed and implemented our intelligent agent that monitors user's email account by notifying the new incoming messages not only with text notification but also with voice notification. The intelligent agent interface is shown in the Figure 1.1. It is a combination of intelligent agent theory and architecture, network protocol, voice technology and theory, electronic mail protocol, Object-Oriented design theory and Java API. The agent implementation provides great convenience to interface users, enabling them to surf the net while obtaining any new messages from their account.



Date	From	Subject
Tue Mar 27 ...	e talking <e_...	Hi
Tue Mar 27 ...	e talking <e_...	This is an at...

Date: Tue Mar 27 22:54:40 PST 2001
From: e talking <e_talking@yahoo.com>
To: e_talking@yahoo.com
Subject: Hi

Hi,
This is the first message. Let the agent speak
something. Say how are you, in Chinese, this is to say
ni how...

Do You Yahoo!?
Get email at your own domain with Yahoo! Mail.
http://personal.mail.yahoo.com/?.refer=text

Figure 1.1 The Main Interface

1.3 Organization of the Thesis

The rest of the thesis is organized as the following: the various definitions of agent, intelligent agent theory and frameworks are discussed in chapter2. In chapter 3 we give a detailed discussion our model technical background introduction including: speech technology (speech synthesis and speech recognition), Internet email protocol, Object-Oriented design theory and Java API. Chapter 4 and 5 are the core chapters of our work. In chapter 4, we discuss our voice email agent architecture. We focus on the development of our agent architecture, discussing the features of our architecture and how we implement our features in the agent architecture in this chapter. In chapter 5, we have a detailed discussion of our agent implementation such as how the techniques in chapter 3 are implemented. In the last chapter the significance of our work and the improvements we should consider in the future work are discussed.

Chapter 2

Intelligent Agent: Theory, Classification, Framework and Architecture

In this chapter we explore various definitions of agent, the intelligent agent theory, and certain intelligent agent frameworks. It gives us a background introduction about intelligent agent theory, application, and research developments. Doing a survey of intelligent agent concept is to justify the need of using subsumption architecture [Brooks, 1986] where our agent is built.

2.1 What Is An Agent

2.1.1 Exploring various definitions of agent

To discuss what is an agent is important for our work because we want to design an intelligent agent system with the properties of an agent. In computer science area, agent concept is used for its basic functions but not limited to it. One of the extended applications is software agent concept. A software agent is a software entity which functions continuously and autonomously in a particular environment, often inhabited by other agents and processes. One implication of software agent is its intelligent behaviors, which indicates that intelligent agent might have services capabilities, autonomous decision, and commitment features. Because artificial intelligent is a science that simulates certain processes from human, these processes have intelligent property. Therefore intelligent agent is considered as an important research area of artificial intelligent. The research of intelligent agent (or in a broad sense, software agent) gives us a good opportunity to explore artificial intelligence theory. In the application field, intelligent agents are also applied in many business areas including business and management.

Let us explore intelligent agent definition first. In the research field, there is no clear definition of what is an intelligent agent. Stan Franklin in his article, *Is it an Agent, or just a Program?* [Franklin, 1996] gave us a full introduction of several definitions:

The first one is provided by Russell and Norvig [Russell and Norvig, 1995], who were the authors of popular AI textbook "Artificial Intelligence: a Modern Approach". Russell and Norvig gave such a definition that *"an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors"*. It focuses on environment, the sensing and reactions.

As one of the pioneers of agent research, Pattie Maes of MIT's Media Lab thought [Maes, 1995] *"autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment,* *and by doing so realize a set of goals or tasks for which they are designed"*. The contribution of this definition is that it indicates that agents have to act autonomously so as to "realize a set of goals."

Intelligent agent has also been widely discussed in white papers. For example, Sankar Virdhagriswaran noticed that "*the term agent is used to represent two orthogonal concepts. The first is the agent's ability for autonomous execution. The second is the agent's ability to perform domain oriented reasoning*". It has also been noticed that *"intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires"*, which indicates that an agent acts for another [IBMWP, 2000]. One of examples could be an information gathering agent (http://www-4.ibm.com/software/speech/).

Though what is intelligent agent still need further investigation, our work which designs a intelligent agent system follows Wooldridge and Jennings' proposal that is a more clear and complete definition. Wooldridge and Jennings [Wooldridge and Jennings, 1995] proposed that an intelligent agent refers to *" a hardware or (more usually) software-based computer system that enjoys the following properties:*

- *autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;*
- *social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;*
- *reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;*
- *pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative."*

They indicate that an agent is autonomy, sensing and acting, allowing for a broad, but finite, range of environments. There is also a communications requirement for an agent. This definition could be more comprehensive compared to previous ones.

2.1.2 A more formal definition of agent

Further more, Wooldridge and Jennings [Wooldridge and Jennings, 1995] had tried to put intelligent agent in a theoretical setting by defining the intelligent agent with the components that comprise it. Therefore an intelligent agent is a structure of (L, E, A, F, C, I):

L: a set of agent states describing the internal status of the agent;

E: a set of external or environment states, representing inputs to the agent,

A: a set of actions representing those that the agent might perform,

F: next state function determined by E and L,

C: choice function from L to A,

I: an initial state.

This structure is a clear picture of what is an intelligent agent including its internal status, inputs to the agent, actions that an intelligent agents perform, next state function, choice function and initial state.

One point need to be pointed out is E (environment states): a set of external or environment states, representing inputs to the agent. Intelligent agents are situated in some environments, sometimes if we modify environment, we no longer have an intelligent agent. For example, a robot is not an intelligent agent if there is only visual sensors but without light in an environment.

Another remark is related to A (actions): a set of actions representing those that the agent might perform. An intelligent agent is not defined by its task. For example if a spell checker corrects the typo when a user is typing, it is an intelligent agent. Otherwise a spell checker appended to a word processor is not an agent. Therefore all intelligent agents are computer programs, but not all programs are intelligent agents. In some sense, intelligent agents are the subset of computer programs.

A final point we clarify is that intelligent agent has a broader sense in the real world, especially in some business applications. What we use here is an intelligent agent that could be any software agent with the property of demonstrating intelligent behavior.

With the help of intelligent agent as a structure, our intelligent agent system design is clear and complete. Figure 2.1 tries to illustrate what is an intelligent agent:

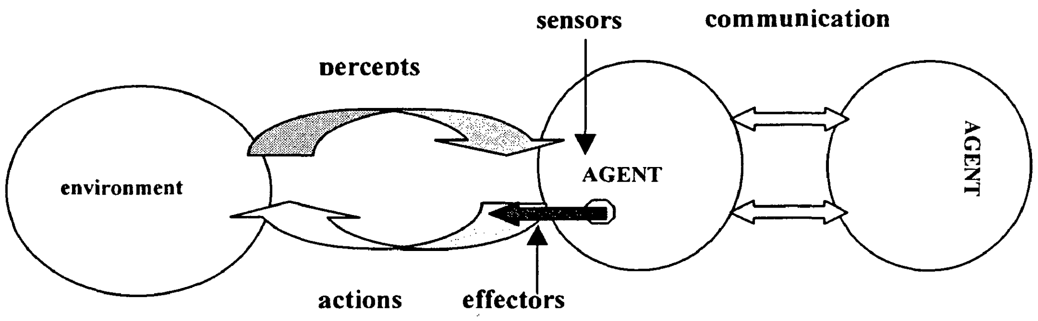


Figure 2.1 The General Model of an Intelligent Agent

From this figure we can see that an intelligent agent perceives its environment through sensors, and acts through its effectors. There are also communications between agents, and each agent sensors outside environment as well.

2.2 The Intelligent Agent Theory

A good understanding of intelligent agent theory development is helpful in designing our own system. In the following section, we discuss intelligent agent theory including its development phase, various theories, and classification.

2.2.1 Early intelligent agent research

An agent idea was first introduced in the mid-1950's by John McCarthy, he was also the first one who introduced the concept of artificial intelligence (AI). The term of intelligent agent (IA), however, was introduced formally by Oliver G. Selfridge several years later. They considered an agent as a system that could carry out the detailed computer operations when given a task and could also ask for and receive advices when it got stuck.

In the development of intelligent agent concept, deliberative agent was one of the most important concepts worthwhile mentioned. It was the first generation of agent research from 1977. This agent concept came from the symbolism research in artificial intelligence, especially the deliberative thinking paradigm. In this paradigm, agents with an internal symbolic reasoning model, in order to achieve their goals, engage in planning and negation with other agents.

2.2.2 New theories in 1990's

The intelligent agent research went into second-generation phase that started around 1990. The study focused on development of agent theories, architectures and languages, and a significant broadening of the typology of agents being investigated. The main part was about autonomous agents. During this period of theory development, researches in distributed artificial intelligence (DAI) played an important role as symbolism in the first generation of agent concept development. Their contributions were especially to distributed problem solving and multi-agent systems. The main idea is how to decompose the original problem into subproblems which are to be solved by various agents, how to synthesis partial results obtained from these agents, how to enable agents to communicate and interact, and how to ensure acts act coherently etc.

2.2.3 Three key issues

Agent theories, architectures and languages are three key issues in recent intelligent agents research [Wooldridge and Jennings, 1995]. These three key issues give a clear picture of future intelligent agent research development.

- *Agent theories*

Agent theories are about what an agent is, representation and reasoning about agent properties. Such representation and reasoning are implemented by the use of mathematical formalisms;

- *Agent architectures*

They focus on implementation of intelligent agents system -- how to construct and design a software or hardware system that satisfies the properties of intelligent agents;

- *Agent languages*

They are software systems that are for programming and experimenting with agents, these programming languages may embody the various principles proposed by researchers.

2.3 The Intelligent Agent Classification

In order to find the place where our intelligent agent stands for, we should take a look at the classification of intelligent agents. Theorists have formalized the intelligent agent classification to collaborative agent, interface agent, mobile agent, information/Internet/Intranet agent, autonomous agent, and reactive agent [Nwana and Azarmi, 1997] as shown in Figure 2.2.

Collaborative agent: indicating that in order to reach mutually acceptable agreements agents have to negotiate with each other;

Interface agent: by it name, indicating when a user wants to learn a particular application, it provides support such as proactive assistance;

Mobile agent: is the agent that is able to roam in wide area networks, with characteristics of autonomous and co-operative;

Information/Internet/Intranet agent: is closely related to Internet and Intranet, it principally manages, manipulates or collects information from many distributed sources;

Autonomous agent: has goal-directedness, proactive and self-starting behavior with characteristics of situatedness, autonomous and flexible;

Reactive agent: is the agent that responds to the present environment where it is embedded in a stimulus-response manner, it usually does not have internal, symbolic models of its environment.

We will discuss reactive agents with more details because our agent model belongs to this type. Reactive agents are special type of intelligent agents. They act or respond in a stimulus-response manner to the embedded environment. According to Maes, there are three key ideas of reactive agents: (1) *emergent functionality*: it indicates that there is no any priori plan for the behavior of the agents, their interactions are dynamic; (2) *task decomposition*: a collection of modules forms a reactive agent, each module acts autonomously for its specific tasks; (3) *reactive agents*: usually act upon such representations as those close to raw sensor data. The most important application of *reactive agent* is Brook's *subsumption architecture* which we introduce in the following section. His architecture is utilized in reactive software agents though it has been mostly implemented for physical robots.

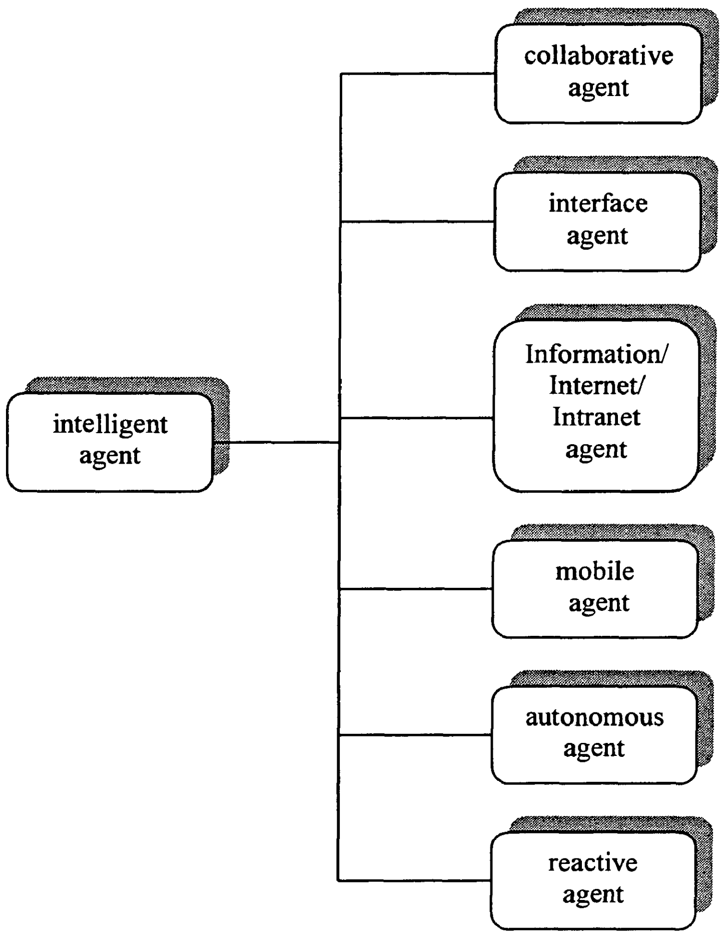


Figure 2.2 Intelligent Agent Classification

2.4 The Intelligent Agent Framework

The intelligent agent frame is the place where intelligent agent locates. The framework is fulfilled by the involving parties which are classified into three major types in our case: the agents involved, any resource that agents need to use, and the environment where agents execute. One of the most complete and contemporary models is framework of Ajanta system [Karnik and Tripathi, 1998]. This framework model is to discuss a mobile agent that represents a user in a network to migrate autonomously from node to node, and perform computations on behalf of that user. Although Ajanta system is to deal mobile agent, the three principal parties of intelligent agent are clearly represented in this model: the *agents* involved, any *resource* that agents need to use, and the *environment* where agents execute.

Each agent has its own state (internal data) and code. As we discussed in the preceding section, an intelligent agent is a structure of L: a set of agent states describing the internal status of the agent; E: a set of external or environment states, representing inputs to the agent; A: a set of actions representing those that the agent might perform; F: next state function determined by E and L; C: choice function from L to A; I: an initial state. These key elements are represented in this framework model.

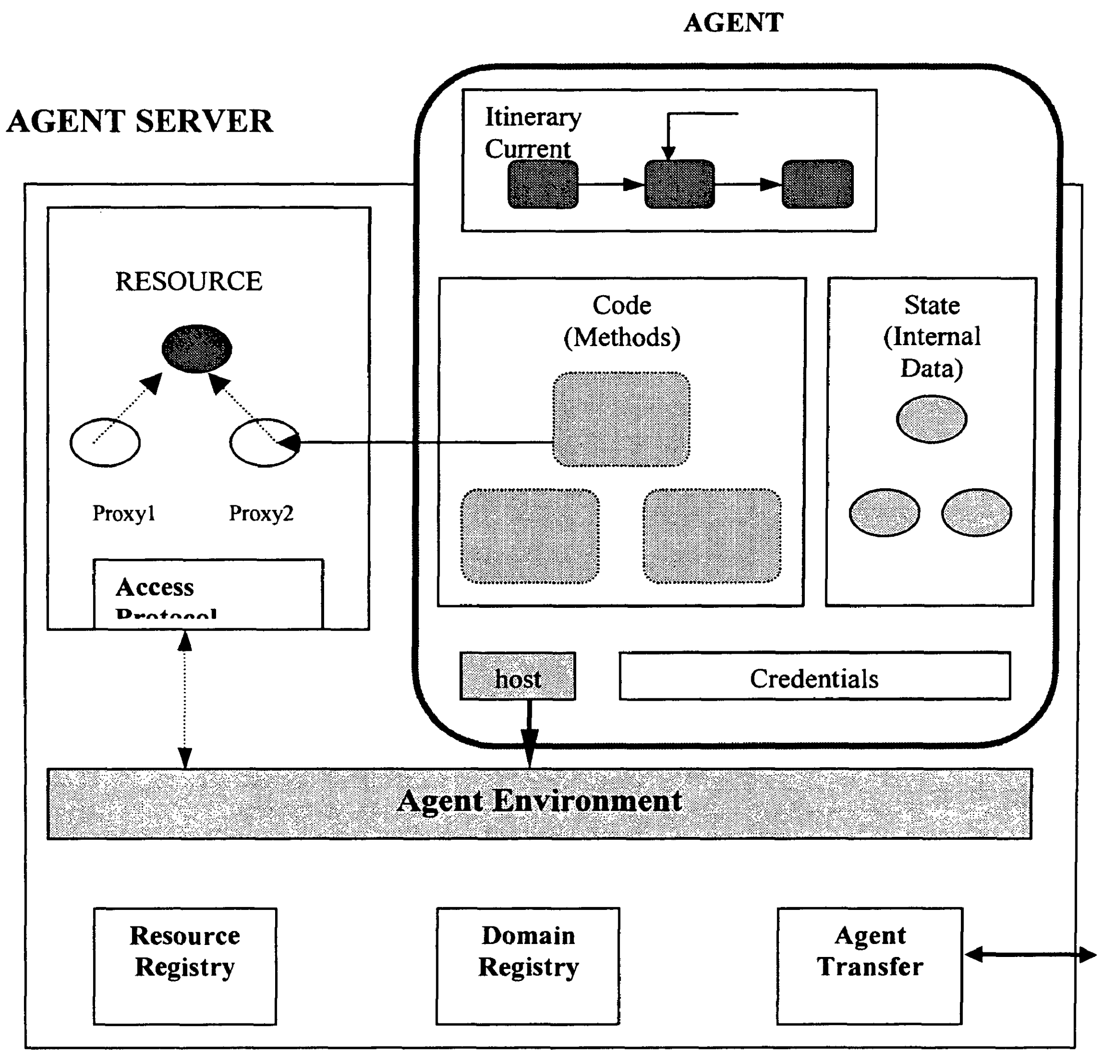


Figure 2.3 The Ajanta General Model

In Ajanta system, agent accesses the resources through proxy, which acts as an intermediary between agent and resources. The advantage of utilizing proxy is that it could have security check and meanwhile have resource availability check.

Proxy plays an important role in communication between agents as well. The environment where agents execute acts as the interface between agents and the services provided by the agent server (or host). In order to request migration, communication with other agents, or access the resources, an intelligent agent could invoke the primitives provided by the environment.

What we have discussed is an overall intelligent agent framework, our agent implementation is a *reactive agent* which responds to the present environment. To design such an agent, we need some kind of architecture to build on. The one we select is subsumption architecture.

2.5 The Subsumption Architecture

The subsumption architecture is the one we build our agent on. In our work, we not only utilize subsumption architecture but also integrate OO design methodology to it. Therefore the advantages of subsumption architecture are fully taken.

2.5.1 What is subsumption architecture

The subsumption architecture was proposed by Rodney A. Brooks [Brooks, 1986]. Generally speaking, it is a layered architecture where each layer senses and acts in order to perform its task. Each layer is also an agent that indicates that it satisfies all the requirements of an intelligent agent. Under such an architecture, an agent is built up with simple behaviors controlled at a low layer, and complex behaviors at its higher layers. The different layers are not completely independent. The advantage of subsumption architecture is to decompose a system into parallel tasks, therefore this architecture increases robustness, concurrency support, incremental construction and ease of testing. Let us propose an example to illustrate the difference between traditional functional decomposition and subsumption architecture: in traditional functional decomposition, a robot functions could be decomposed into: *sensors* → *perception* → *modeling* → *planning* → *task recognition* → *motor control.* It is a very straightforward decomposition, functions are ordered. Next function is based on the completion of the previous one. While in subsumption architecture, these functions are parallel: *avoid objects<wander <explore < build maps <monitor changes <identify objects <plan actions <reason about object behavior.* The symbol < denotes increasing levels of competence. The parallel functions let subsumption architecture to decompose a system into parallel tasks that are not completely independent. It is possible that additional functions are added to this architecture.

2.5.2 Principles of subsumption architecture

The general process of building up a subsumption architecture is to first decompose the problem into a series of task-achieving behaviors. Each is implemented in its own layer. The control system has the property of "plug and play" because of each layer has its own access to sensors and actuators. The building process is incremental, a new one can be added to an existing layer without modification of the built layers. The layers are independent, in the sense that the next layer is built with its own access to sensors and actuators. The higher layer, however, achieve its task with the help from lower layer. Higher layer could contain lower level layer as one of its subsets. All the layers work together to achieve the overall goal. A concrete example helps clarify this concept: suppose there is a level 0 agent is built, this agent contains some sensors which monitor and process procedures. In this stage, we could regard it a complete and simple intelligent agent after testing. According to subsumption architecture, we can add the next control layer to the agent. This layer can monitor the data paths in the level 0 layer and put data onto the level 0 data paths. In this sense, level 1 layer *subsume* the normal data flow of level 0. Additional layer could be added if need. The principles of subsumption architecture could be indicated in the following:

(1) There is no central model of the world;

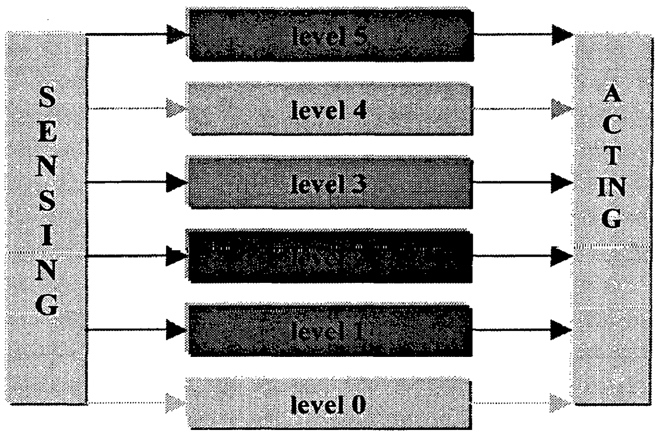
(2) There is no separation into perception, central processing, and actuation systems, in another word, they are intertwined;

(3) Adding more specific behaviors to the existing layer will increase its capabilities;

(4) Messages are available on the appropriate input port when needed;

(5) Behaviors run in parallel, requiring conflict resolution; in this scenario, different behaviors attempt to control the same actuator in different ways.

The subsumption architecture is shown in Figure 2.4, from which we could note the architecture has a set of layers, each layer has its own sensing and acting function though it is not necessarily that all reactive agents have actuators and sensors.



There are six layers in the architecture figure, from lower level to higher level. Layers work asynchronously, that is higher level can inhibit those in lower layers

Figure 2.4 Subsumption Architecture

2.6 Summary

In this chapter we discussed various definitions of agent, the intelligent agent theory, and some intelligent agent frameworks. With the help of this discussion, we can begin our intelligent agent design and implementation. Two important parts are Ajanta system and subsumption architecture. Although Ajanta system is to deal mobile agent, the three principal parties of intelligent agent are clearly represented in this model: the *agents* involved, any *resource* that agents need to use, and the *environment* where agents execute. These parties should be covered in every intelligent agent including our work. Based on such a framework with some changes, we are able to develop our own framework, design an agent system and implement it.

Chapter 3

Supporting Techniques For Reactive Agents

In this chapter, we give a detailed discussion of our model technical background including: speech technology mainly Text-to-Speech Synthesis (TTS), Internet email protocol, Object-Oriented design theory and Java API. These techniques serve as necessary tools for implementing reactive agents.

3.1 TTS and Internet Email Protocol

3.1.1 Text-to-Speech Synthesis (TTS)

Text-to-Speech Synthesis (TTS) researches have been conducted for many years. TTS is the creation of audible speech from computer readable text, which presents a rich array of challenges, and it requires a variety of areas including acoustic phonetics, prosody, computational phonology, computational morphology, and corpus-based linguistics. This development trend comes from the need of man-machine interaction. There are two main approaches to speech production in TTS field: *concatenative* and *rule-based.* Each has its own advantages and weakness.

The concatenative approach is to piece together stored speech units that are originally extracted from natural speech. There are large numbers of factors that can influence the properties of speech segments in natural speech. The forms of the speech units could be stored are either in raw waveforms, or in sets of parameters derived from the waveforms. The current trend in concatenative synthesis is toward waveform concatenation. This methodology is to produce more human-sounding voice quality than earlier TTS systems. The next process is to select units from the unit database, concatenate and modify to reflect prosodic properties of the utterance. The prosodic properties are either intonational or durational. It is a very complex and difficult work to select the best units in order to reconstruct a particular utterance, which needs more research work in the future. On the other hand, all of the perceptually relevant acoustic parameter values are produced by a set of rules in rule-based synthesis. These rules are context-sensitive based on an analysis of natural speech patterns. How to capture the perceptually relevant generalizations (rules) is the principal challenge for this rule-base approach. The task is to produce appropriate values for high-quality synthesis.

Many researchers had achieved accomplishments in this field. Hertz in 1979 and 1982, Allen, Hunnicutt and Klatt in 1987 had such an attempt to assume a segmentation of speech into adjacent phoneme-sized units [BELL, 2000]. They found that in order to appropriately capture generalizations about the acoustic patterns of speech within and across dialects and languages, units of varying sizes are required.

Research institutions are actively involved in this field. The Bell Lab's Text-to-Speech system contains several components, for example, the text analysis capabilities of the system [BELL, 2000]. It detects the ends of sentences, perform rudimentary syntactic analysis, expand digit sequences into words, and disambiguate and expand abbreviations into normally spelled words. These words can be analyzed by the dictionary-based pronunciation module that provides pronunciation for most of words. Another component handles prosodic phrasing, word accentuation, sentence intonation, and the actual speech synthesis. The AT&T Lab's group devotes their efforts to increase the naturalness of speech synthesis significantly while maintaining good intelligibility [ATT, 2000]. Their new TTS was introduced in 1998 and marked a dramatic leap in naturalness. Microsoft's Text-to-Speech engine is also a concatenative synthesis base, the audio output from the engine is generated from files, which contain information derived from recordings of real people [MSFT, 2000]. IBM ViaVoice Text-To-Speech is the speech synthesis process that goes through several high level linguistic stages to create highly intelligible speech output [IBM, 2000]. The uniqueness of their TTS is using small snippets of actual spoken recordings and gluing them together. The formant technology utilizes a Klatt type synthesizer. The attributes of their TTS are their highly intelligible and flexible aspects. It can accommodate an unlimited number of voices by modifying the gender, pitch, head size, roughness, etc. However, exploiting the strength of TTS requires the integration of TTS and other technologies, as well as a good understanding of related technical fields. This thesis explores several features of TTS and related technologies so that the advantages of TTS can be fully utilized by integrating TTS with other techniques. As mentioned early, as Bell Labs and AT&T put much efforts on TTS research, most of the applications of TTS are about telephone systems, such as: using an in-flight phone to get stock quotes, generating spoken prompts in voice response systems, serving as an interface to an order-verification system for salespeople in the field, and giving users the ability to access textual information over the phone, etc. Studying the proposed topic provides an excellent opportunity to add new functionalities to TTS. One of such directions could be adding voice function to Internet email applications. It requires good understanding of TTS, networking protocol, and software design methodology. The most important is the good understating of text stream to speech stream techniques. What we are doing is in this direction, which expands TTS application field.

There are no such works that are to integrate intelligent agent theory, TTS, Internet email, networking protocol, and OO design together so far. Especially the power of OO design makes model implementation more easily and reusable, which is the strength of Object-Oriented methodology. My thesis research is such an endeavor to accomplish this goal.

3.1.2 Internet Email protocol

We add voice functions to the agents by utilizing TTS and other techniques such as network protocol etc. The method to implement this idea is to have Internet email application with its unique voice function. Email is one of the protocols included with the Transport Control Protocol/Internet Protocol (TCP/IP) suite of protocols. A popular protocol for receiving e-mail is POP3 and a popular protocol for sending it is SMTP. There are two protocols dealing with the receiving of email: POP (Post Office Protocol 3) and IMAP (Internet Message Access Protocol). POP3 is the most recent version of a standard protocol. It is a client/server protocol for receiving e-mail. The email is received and held by the Internet server. The email receiver periodically checks the mailbox on the server and downloads any mail. An alternative protocol is IMAP. It is different than POP3 in that email is viewed at the server as though it was on the client computer with IMAP. An email message deleted locally is still on the server. E-mail can be kept on and searched at the server. POP service could be thought of as a "store-and-forward" while IMAP service can be thought of as a remote file server.

The protocol for transferring email across the Internet is SMTP (Simple Mail Transfer Protocol), which is a TCP/IP protocol. Because its limitation to queue messages at the receiving end, it's usually used with one of two other protocols, POP3 or IMAP. In such a way, the user save messages in a server mailbox and download them periodically from the server. It means that users typically use a program that uses SMTP for sending e-mail and either POP3 or IMAP for receiving messages [Wood, 1999].

3.2 Object-Oriented Design and Java API

We apply an object-oriented approach to this task, which is approved a very fast and efficient method. The Object-Oriented Design (OOD) is an approach that partitions a system into objects. Each object in the system includes the data and all necessary operations that manipulate it. The primary initiatives for object orientation are to keep objects unchanged though their functions tend to change with the evolution of a system. Therefore the system is more maintainable with OOD approach. There is an interesting connection between the concepts of agents and objects. However, objects are not conceptually compatible with agents. Though an object is defined in terms of its operations (methods), thus controlling its own state (which is a particular snapshot of its behavior), it does not have control over its overall behavior. Agent-based software engineering is often compared to object-oriented programming. The main difference lies in the language of the interface: in agent-base software engineering, agents use a common language with an agent-independent semantics [Bradshaw, 1997] [Chen, 1999].

Three most important concepts of OOD are encapsulation, messaging, and inheritance. We first introduce the concept of implementation hiding which is the same as access control. Access control puts boundaries within a data type. Therefore the client programmers know what he can and what he can't use. On the other hand, the interface is separated from the implementation. Inclusion of data and operations within objects combined with implementation hiding is called encapsulation [Eckel, 2000].

With the help of encapsulation, system designer just choose objects they need and execute them. At the same time, the maintenance becomes more convenient since changes in one object do not affect other objects. Messaging are used for manipulating objects, each message requests an object to perform actions. Therefore an object does useful work for you, such as completing a transaction, drawing something on the screen, or turning on a switch. You can send requests to different objects, and each object can complete only certain tasks.

Another important characteristic of OOD is inheritance. It provides a nice mechanism to utilize the existing objects, and make additional modifications. Therefore there is a hierarchy where objects are arranged from general to more specific objects. Each object in lower level inherits attributes and behaviors from higher level objects.

Because of these three characteristics of OOD, it is possible that objects are reusable. The objects can be moved around easily. They are self-contained and autonomous, therefore it is a ideal scenario that these objects could be used as components in many kinds of systems.

The implementation language is Java. Its idea of platform independent enables "program once and run everywhere". It is an ideal language for this research that explores integration of electronic mail and voice technology. We apply JavaMail API and JavaSpeech API (http://java.sun.com/products/javamail/index.html and http://java.sun.com/products/java-media/speech). The JavaMail API provides a set of abstract classes that model a mail system, which enables us to create an independent framework [SUN, 2000]. This framework is platform independent. The JavaMail API is implemented as a Java platform standard extension. It supports the implementation of the POP3, IMAP and SMTP.

IBM ViaVoice Text-To-Speech is the speech synthesis process that goes through several high level linguistic stages to create highly intelligible speech output [IBM, 2000]. The Java implementation of the ViaVoice SDK is to incorporate IBM's ViaVoice speech technology into user interfaces. The SDK could be used in our text-to-speech synthesis supporting multi languages such as French, German, Italian, Spanish, UK English, and US English. It is an implementation of

Java Speech API which was developed by Sun Microsystems in collaboration with IBM and other speech technology developers, in other word, it is built on top of the native speech recognition and synthesis capabilities in IBM ViaVoice. Our implementation of TTS is based on such a technology and integrates it with our agent system.

3.3 Summary

We introduced our model technical background such as Text-to-Speech Synthesis (TTS), Internet email protocol, Object-Oriented design theory and Java API in this chapter. These techniques serve as enabling techniques for our email agent – a multimedia intelligent agent: TTS supports intelligent agent in that it can give us audible notification and read for us; Internet email protocol such POP3 and IMAP enable intelligent agent to receive and send e-mail; OOD supports intelligent agent in that objects are self-contained and autonomous, therefore they can be used as components in our system. Our agent system is implemented by utilizing these technologies that work together in our work.

Chapter 4

Development of Voice Email Agent Architecture

The voice email agent architecture is core of our work, our mission is to develop such a architecture that is reusable, incremental, partial-independent and easy to integrate with a broad sense of intelligent agent applications. In this chapter we focus on the development of our agent architecture, discussing the features of our architecture and how we implement these features in the agent architecture.

4.1 Project Model Implementation Methodology and Project Design Considerations

A good methodology for implementation is crucial for our work. It provides us a guideline for how our agent model should be implemented. In this section, we discuss the process of our work, and the factors that we should consider (http://java.sun.com/products/java-media/speech). Based on such an implementation methodology, we discuss the agent architecture.

4.1.1 Project model implementation methodology and design considerations

Designing and building a good intelligent agent system is a rather complex task that requires a deep and comprehensive understanding of intelligent agent theory and other related issues. In order to accomplish this task, we should follow these guidelines:

(1) Carry out a comprehensive study of intelligent agent theory;

(2) Identify the framework of our intelligent agent system;

(3) Define the requirements for intelligent agent implementation;

(4) Implement the system according to the framework and requirements defined;

(5) Evaluate the system and completing the documentation.

Because our voice email agent is an application with special voice features, we should take the following aspects into our considerations:

(1) We should provide a robust, cross-platform voice email agent system with speech synthesis function. The agent should handle new message notification correctly and give proper process. We need an agent that runs on any platforms because the future users will use it on multiple platforms;

(2) The agent should access to state-of-the-art Internet mail and speech technology. The changing technology requires us to implement our agent with the updated one to ensure that any new features are utilized;

(3) The agent should be simple and easy to use.

4.1.2 The reasons that we add voice feature to the agent

Speech is one of the most natural mediums for human communication, email has become another popular communication medium with the Internet development. If the agent we develop were intelligent in both visual and audible aspects, it would be more powerful and convenient with such an extended feature of voice. Voice in our agent is one of the output approaches besides the text interface. Table 4.1 indicates that it is appropriate to add voice feature to our agent:

When the voice is appropriate	**When the voice is not**
(1) Users need to look at something else instead of screen or something else attracts users' attention; (2) A personality need to be embedded into an interface; (3) For the users who have a physical disability	(1) Users need to present a large amount of information; (2) Something need to be compared; (3) Something is private or confidential

Table 4.1 The Scenario for Adding Voice Feature

When there is a new message coming, users usually are doing other duties. It is desirable that if the new message can be read for them. It is even more desirable that if user gives certain instruction to let agent know if he want to hear it or not. Therefore adding voice feature is necessary. Because we can change the voice properties such as age, gender and volume, the agent posses certain personalities we need instead of interface we can only look at. Undoubtedly, if a message is private or confidential, we are able to take off the voice function as simple as it is.

4.1.3 The challenges for integrating speech into agent system

During the process of design, basically there are six challenges that we have to face:

(1) *Transience*: contrast to graphical interface which is persistent, speech is of transience; if there is a lot of information, the users might only remember part of it, they only have limited ability to remember transient information;

(2) *Invisibility*: namely, it is difficult to communicate the functions to users such as what they should do, what actions they should take;

(3) *Asymmetry*: people can not listen as easily and quickly as they speak; therefore, listening and speaking should be compromised in the process of design;

(4) *Synthesis quality*: the quality is not always satisfactory and natural. It is even more difficult when the speech output is dynamic because the pre-recorded output does not help;

(5) agent should be able to recognize simple instructions from user;

(6) agent handling any new message requires the coordination of voice functions and other GUI interfaces. In other words, should users have visual notifications first or have both notifications at the same time.

Only do we consider the above-mentioned challenges and find solutions, our agent is able to convey natural voice stream to user and take voice stream from user, therefore it becomes a really multimedia intelligent agent.

4.2 The Voice Email Agent Architecture

Before we explore our agent model architecture, we give overall discussion about our work. Whenever we connect to the Internet, we can open this agent by typing command in the command line. The agent can connect to any POP3 mail server, checking message folders, fetching all the messages in the folder, and notifying us of any new messages. It identifies various types of messages such as plain/text, multipart and nested message.

This goal is accomplished in the following steps:

(1) The agent connects to our mail server, fetching all the messages in our INBOX folder. All the messages in the agent are displayed in a GUI interface, therefore we can go through all the messages. If we want to listen to any message, just click that message and select speak function, then the agent reads for us;

(2) As long as the agent is active, it checks if there are any new messages in our folder. If there are any and user want to heart it, the agent reads the message for user including sender and subject. It identifies various types of messages such as plain/text, multipart and nested message;

(3) A visual notification is also shown on the screen. With both visual and voice notifications, our agent becomes a multimedia intelligent agent;

(4) We are able to set the agent preferences by selecting voice type (either male or female), selecting the age (young or aged), adjusting the volume, deciding how long of interval to check new messages etc.

This process is indicated in Figure 4.1:

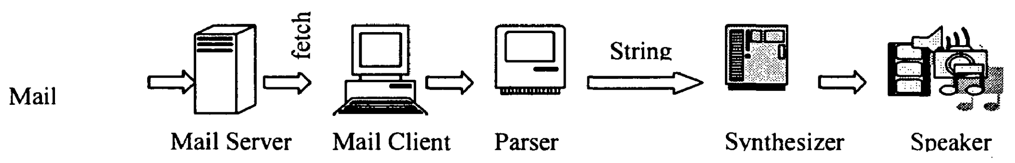


Figure 4.1 The Overall Flowchart

4.2.1 Developing a layered agent model

We introduce a layered model as the solutions for the challenges we face. This solution models the agent into three layers: account process layer, notification layer, and speech integration layer. Therefore the layered agent accomplishes the goals we set.

Account process layer

The account process layer is the first layer that does the basic works such as connecting to the mail account, fetching messages etc. This layer is below the other two layers: speech integration layer and notification process layer, whenever we open our agent system this layer is the one that work within the system. Although we introduce more implementation details in chapter 5, it is easy to understand if we put more concrete examples in this chapter. Basically account process layer creates a mail *Session* object managing both configuration options and user authentication information. Each session could connect to multiple message stores and transports. The agent establishes the default session that accesses INBOX folder. The default session is created as

Session defaultSession

= Session.getDefaultInstance(props, authenticator);

The *Properties* object contains default values and other configuration information such as *mail.store.protocol*, *mail.transport.protocol*, *mail.host*, *mail.user*, and *mail.from* etc. If there is no specific requirement, *the system properties* object that is retrieved from the *System.getProperties* method is used. Then the *Store* and *Folder* classes are created in this layer. As we know, messages are stored in folders where new messages are usually delivered. Access protocols are also defined in *Store* to access folders and retrieve messages from folders. The *Folder* can contain both subfolders and messages, and all the messages within a *Folder* are sequentially numbered. As we note, the basic functions are achieved in this layer.

Notification layer

This is the layer on top of *account process layer*. Its actions are based on the results of the first layer, in another word, our intelligent agent automatically notifies the new incoming messages as long as the account is active. It checks the account every certain amount of time. As long as there are any new ones, our agent sends a message to the next speech agent to notify the user with both text and voice notification.

Speech integration layer

This layer represents a speech agent that deals with either speech input or speech output. The speech agent basically has such processes:

(a) Identifying requirements such as what language or dictation capability;

(b) Locating and creating an engine; allocating the resources for the engine, and setting up;

(c) Beginning operation of the engine such as resume it;

(d) Using the engine;

(e) Deallocating the resources of the engine.

Every speech engine must be in one and only one of the following four allocation states: either DEALLOCATED, ALLOCATED, ALLOCATING_RESOURCES, or DEALLOCATING_RESOURCES. The ALLOCATED state has several sub-states: either the PAUSED or the RESUMED state.

The speak out function is provided by a *Synthesizer* which speaks text, manages a queue of text to be spoken. The properties of speech engine are the *SynthesizerProperties* objects that define five synthesizer properties. These properties could be modified during operation of a synthesizer in order to effect speech output properties. The first property is the *voice* property that is used to control the speaking voice of the synthesizer. The other four properties are to control *prosody* of speech including the pitch, intonation, rhythm, timing, stress and other characteristics that affect the style of the speech.

4.2.2 The agent architecture

Our intelligent agent is based on subsumption architecture we discussed in the previous section. The general process of building up a subsumption architecture is to first decompose the problem into a series of task-achieving behaviors. Each is implemented in its own layer. Our agent could be classified as reactive agent that responds to the present environment. Therefore three layers are used to represent account process, voice notification, and speech. Figure 4.2 indicates our subsumption architecture:

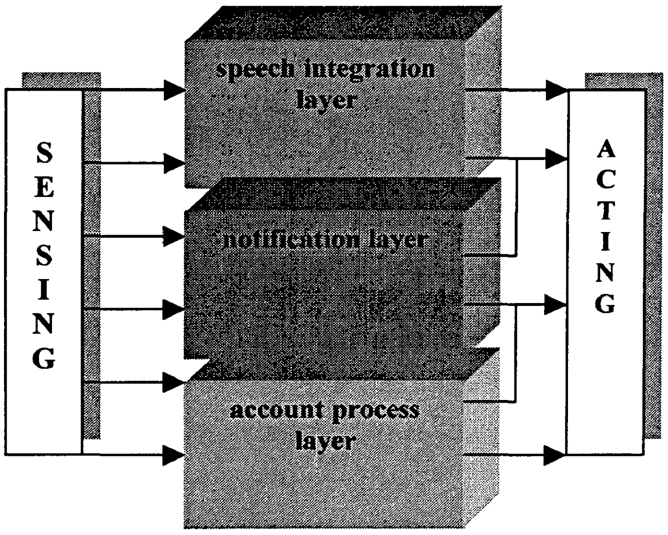


Figure 4.2 An Intelligent Voice Email Agent Architecture

Three layers are clearly represented in the figure, and each layer represented one subagent of our intelligent agent system. The main property of this architecture is "plug and play", each one has its own access to outside environment. Because the building process is incremental, a new one can be added to an existing layer without modification of the built layers. These layers are independent in the sense that each one has its own access but not absolutely independent. The *account process layer* provide data for *notification layer*, they are somehow interrelated. The *speech integration layer* acts upon the changes in two previous layers, but it could act itself. Therefore each layer of control is simply added to existing one to achieve the overall goal.

In the meantime, each layer is also reusable. For example, account process layer could be used by other agents as long as such agents need the email process functions: connecting to a session, opening folders etc. Incremental indicates that the whole agent system is built layer by layer. The first one is account process layer that provides necessary infrastructure for the later layers. The notification layer is built above account process layer and the speech integration layer is another one built on top of notification layer. It is possible that a new layer could be added on the top of the speech integration layer if needed. The function of speech integration layer is utilized by this new layer.

4.2.3 Features of the layered agent model

With the help of integration of subsumption architecture and Object-Oriented technology, we develop our agent architecture with outstanding features that are reusable, incremental, partial-independent and easy to integrate with a broad sense of intelligent agent applications.

(1) Reusable:

The reusable feature of our architecture is to allow developer to reuse part of architecture such as a component of the architecture. The components can be moved around easily because they are self-contained. The implementation of each component is invisible to developers. Therefore they know what components he can and what he can't use. There are also communications between different components by sending requests to ask someone else to complete certain tasks. Therefore each component in the architecture is reusable.

(2) Incremental:

Incremental process indicates that a new component can be added to an existing one. The incremental feature of our architecture provides a great convenience to designer, which allows them to know which component could be designed first and which one could be later. Certain component is based on the completion of another component.

(3) Partial-Independent:

The component is partly independent in the sense that though certain components are built with its own access to the outside world including other components, they achieve their tasks with the help from other layers sometimes. Higher layer could contain lower level layer as one of its subsets. The partly independent component cooperates sometimes to achieve a specific goal. Because each of them is not absolutely independent, we could add a new component to existing one, this process is incremental as we just discussed. Therefore with such features, our Voice Email agent architecture is easy to integrate with a broad sense of intelligent agent applications.

4.3 Summary

The agent we developed in this chapter is intelligent in both visual and audible aspects. When there is a new message coming, it reads new message for users. The introduction of our agent architecture demonstrates that our voice email agent architecture is such an architecture that is reusable, incremental, partial-independent, and easy to integrate with a broad sense of intelligent agent applications. The advantages of this architecture are that it is a layered model: account process layer, speech integration layer, and notification layer. We develop our agent architecture with outstanding features that are reusable, incremental, partial-independent and easy to integrate with a broad sense of intelligent agent applications. The agent with such features is able to handle the challenges we face: transience, invisibility, and speech synthesis quality, and the coordination of voice functions and other GUI interfaces.

Chapter 5

Implementation of Intelligent Voice-Email Agent

In this chapter, we have a detailed discussion of our agent implementation that is how the techniques in chapter 3 are implemented including speech technology such as Text-to-Speech Synthesis (TTS), Internet email protocol, Object-Oriented design theory etc. In the first subsection, we give a brief introduction of what our agent is and what functions it has.

5.1 The Principal Functions and Features

The Voice-Email intelligent agent system communicates text messages and their contents, by which its intelligent converter delivers abbreviations, punctuation, time and date formats, and many other potentially ambiguous texts in a way that is easily deciphered and correctly spoken. It connects to our mail server, fetching all the messages in the folder. All the messages in the folder are shown in a graphic user interface, therefore we can go through all the messages. If we want to listen to any message, just click that message and select speak function. As long as the agent is active, it checks if there are any new messages in our folder. If there are any, the agent read the message for us including sender and subject based on user instruction. It identifies various types of messages such as plain/text, multipart and nested message. A visual notification is also shown on the screen. There are also functions associated with the speech synthesizer: selecting voice type (either male or female), selecting the age (young or aged), adjusting the volume, deciding how long of interval to check new messages etc.

5.2 The Implementation

Our Voice-Email system utilizes Java programming language, which enables it to be used on any platform on which the Java Virtual Machine (JVM) is installed. It uses JDK1.3 and JavaMail and JavaSpeech API (http://java.sun.com/products/javamail/index.html and http://java.sun.com/products/java-media/speech). The speech engine is IBM's ViaVoice.

5.2.1 The implementation of TTS

As mentioned in the previous chapters, IBM ViaVoice Text-To-Speech is the speech synthesis process that goes through several high level linguistic stages to create highly intelligible speech output. The uniqueness of their TTS is using small snippets of actual spoken recordings and gluing them together. The Java implementation of the ViaVoice SDK is to incorporate IBM's ViaVoice speech technology into user interfaces. The SDK could be used in our text-to-speech synthesis supporting multi languages such as French, German, Italian, Spanish, UK English, and US English. It is an implementation of Java Speech API which was developed by Sun Microsystems with IBM, in other word, it is built on top of the native speech recognition and synthesis capabilities in IBM ViaVoice [IBM 2000]. Our implementation of TTS is based on such a technology and integrates it with our agent system. We first discuss the algorithms behind the Speech engine interface to explain how it works.

5.2.1.1 The speech engine and its properties

We implement speech engine by the *javax.speech* package that defines an abstract software representation including engine's properties. The speech engine that handles text output could be described in the following:

(1) Identifying functional requirements for an engine

(2) Locating and creating an engine for those functional requirements

(3) Allocating the resources for the engine

(4) Setting up the engine

(5) Beginning operation of the engine

(6) Using the engine

(7) Deallocating the resources of the engine

This is the basic speech engine work flow, the outcome of this process is a speech engine with basic properties, we can utilize it until deallocates the resources for the engine. Each engine has its own properties defined in the *EngineModeDesc* class, and additional properties for synthesizers are defined in

SynthesizerModeDesc classes. The basic properties include *EngineName*, *ModeName*, *Locale* and *Running*, the additional properties are *List of Voices*. *EngineName* is a string that defines the speech engine name, and *ModeName* is another string that defines a specific mode of operation of the engine. *Locale* refers to the languages/countries supported by the engine, and *Running* is a Boolean value that indicates if the engine is running on a platform. The additional property of *List of Voices* is an array of voices that the synthesizer is capable of producing. Each voice is a combination of voice name, gender, age and speaking style. The synthesizer is a sub class of speech engine, which is an engine that converts text to speech. As a type of speech engine, speech synthesizer inherits much of functionalities of a speech engine. In the following section, we discuss synthesizer's states.

5.2.1.2 The speech synthesizer state

We first discuss the states of speech engine and then some special states of synthesizer. A speech engine has four states: DEALLOCATED, ALLOCATED, ALLOCATING_RESOURCES AND DEALLOCATING_RESOURCES. The ALLOCATED state has two sub states: PAUSED, or the RESUMED states. Figure 5.1 illustrates these four states:

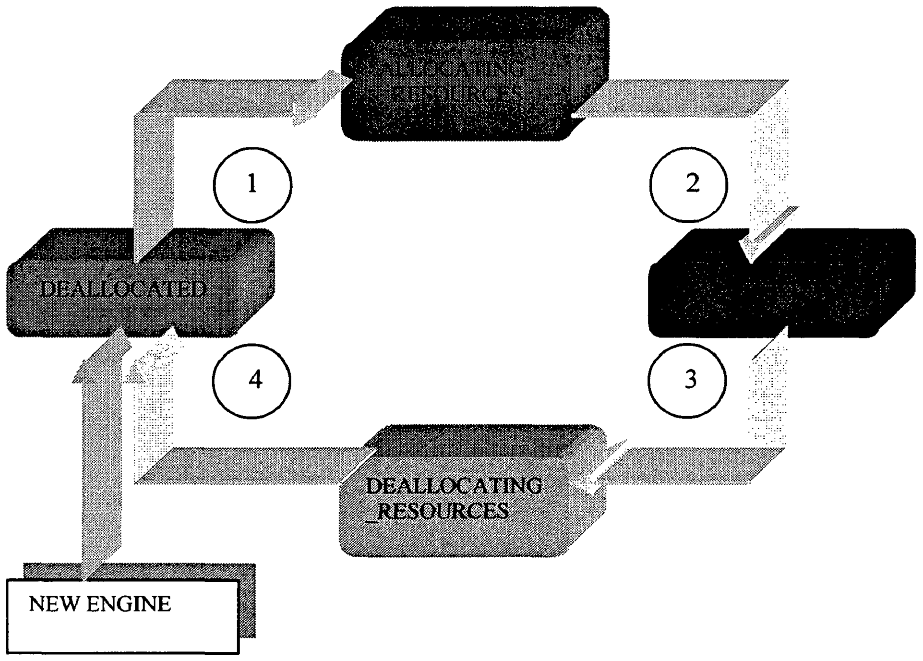


Figure 5.1 The Speech Synthesizer States

A new created engine is always in the DEALLOCATED state, then a call to allocate is needed. As indicated in the above figure, from DEALLOCATED state to ALLOCATING_RESOUCES state is (1) and from ALLOCATING_RESOURCES to ALLOCATED is (2) and so on so forth. There are also additional sub states for synthesizer in ALLOCATED state plus PAUSED and RESUMED states mentioned above: QUEUE_EMPTY and QUEUE_NOT_EMPTY. Any allocated synthesizer is either one of these two states. Notice that QUEUE_EMPTY and QUEUE_NOT_EMPTY states are parallel to PAUSED and RESUMED states and run independently. The reason for these two additional states is that speaking out text puts an object onto synthesizer's speech FIFO output queue. The first object is spoken is the one on the top of the queue if the queue is not empty or the one when a paused synthesizer is resumed. Figure 5.2 illustrates the sub states of an allocated speech synthesizer.

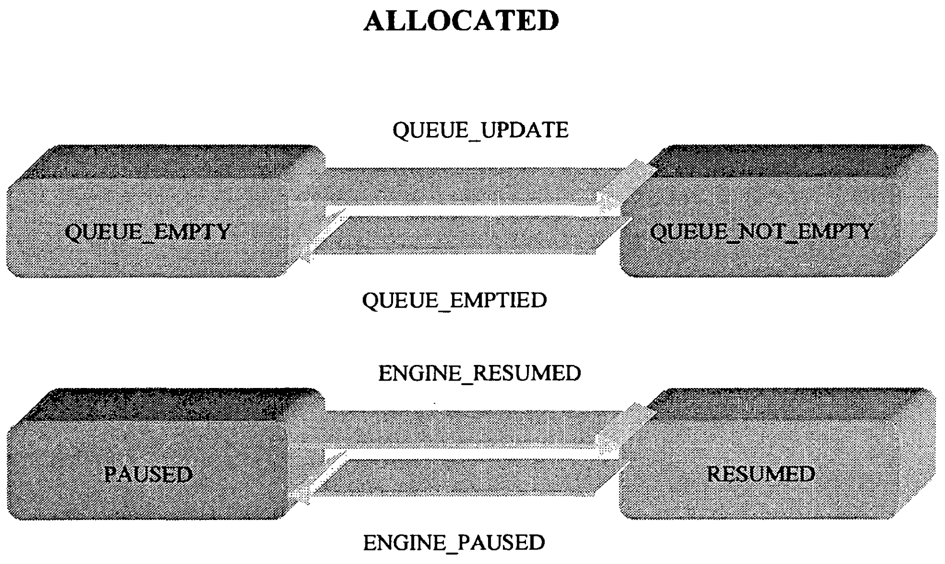


Figure 5.2 The Substates of ALLOCATED

5.2.1.3 The speech synthesizer mechanism

As indicated in section 5.2.1.1, a speech engine handling text output identifies functional requirements, locating and creating an engine, allocating the resources for the engine and setting up the engine, beginning operation of the engine, using the engine, and then deallocating the resources of the engine. For speech synthesizer, this process is more specific as illustrated in the following algorithm:

(1) Creating a synthesizer for certain kind of language

(2) Getting it ready to speak

(3) Calling the allocate function

(4) Calling the resume function

(5) Calling the SpeakPlainText function

(6) Waiting till speaking is done

(7) Cleaning up by calling deallocation function

There are four principal steps in this process: create, allocate and resume, generate, and deallocate. By calling the *createSynthesizer* function of the *Central class* of *javax.speech* package, a speech synthesizer is obtained. The *Synthesizer* object is prepared to output speech text by calling *allocate* and *resume* methods. At this point, the synthesizer is in the RESUMED state. Before the synthesizer finishes speech, the *waitEngineState* method is called, and then *deallocate* frees the resources.

5.2.1.4 The speech synthesizer properties

We have given a brief introduction of speech engine properties in section 5.2.1.1, in this section a more detailed discussion of speech synthesizer properties are introduced. There are two categories of synthesizer properties: *voice* and *prosody*.

The *voice* property controls the voice of synthesizer: each one is defined by a name, gender, age and speaking style. The gender could be GENDER_FEMALE, GENDER_MALE, GENDER_NEUTAL or GENDER_DON'T_CARE. The age could be AGE_CHILD (up to 12 years), AGE_TEENAGER (13-19), AGE_YOUNGER_ADULT (20-40), AGE_MIDDLE_ADULT (40-60), AGE_OLDER_ADULT (60+), AGE_NEUTRAL, and AGE_DONT_CARE. The speaking style could be classified as *casual*, *business*, or *happy* etc. One example of a voice could be defined as the following:

Voice("name_string", GENDER_FEMALE, AGE_MIDDLE_ADULT, "happy");

The defined *Voice* object could be used in the selection of a synthesizer as shown in the following pseudo code:

```
    SynthesizerModeDesc = new SynthesizerModeDesc( );
    SynthesizerModeDesc.addVoice(Voice);
    SynthesizerModeDesc.setLocale(Locale.US);
    Synthesizer = Central.createSynthesizer(SynthesizerModeDesc);
```

The next category is about *prosody* which includes the pitch, intonation, rhythm and timing, stress and other properties. Therefore the prosodic features are: *Volume*, *Speaking rate*, *Pitch*, and *Pitch range*. *Volume* is a float with range of 0.0 (silence) to 1.0 (loudest). *Speaking rate* is also a float value in words per minute, the higher the faster output. *Pitch* is a float value in Hertz and *Pitch range* is also a float range value for pitch variation.

5.2.1.5 JSML and speech synthesizer properties

We use JSML for controlling speech more smooth. JSML stands for *Java Speech Markup Language* that enables applications to provide a high quality and naturalness of synthesized speech by annotating text with additional information. It is a kind of markup language similar to HTML. By adding markers, we can control the output of synthesized speech such as pronunciation of words and phrases, the emphasis of words, the placements of boundaries and pauses, and the control of pitch etc. The properties mentioned in last section can be used within JSML text. One of the examples is shown in the following:

```
// Speak the sender's name slower to be clearer
StringBuffer.append("Message from"+"<PROS RATE=\"-30\">"+ sender
                    + ",</PROS>");
```

```
// Date
StringBuffer.append(" delivered " + "<SAYAS class=\"date\">" + date +
                    "</SAYAS>");
// Subject
StringBuffer.append(",with subject:"+"<PROS RATE=\"-30\">" +
                    subject + "</PROS>");
```

In the above example, we can add special tags to let speaking of string slower and clearer. The advantages of using JSML text are that the properties could be controlled more finely and naturally. It can even change the property for word level as shown in the previous example.

5.2.2 The implementation of electronic mail

By utilizing JavaMail API, we can add electronic mail ability to any applications that we want them to have process email functions. It makes implementation more easy and simple. The principal content of such API is appropriate convenience classes that include common mail functions and protocols. One of advantages of JavaMail API is that it supports many different message system implementations, either different message stores or different message formats, or different message transports. It provides base classes and interfaces for any client applications. In this section, we discussion the implementation details.

5.2.2.1 JavaMail API architecture

JavaMail API is a layered architecture that is composed of three layers: (1) abstract layer, (2) implementation layer and (3) JavaBeans Activation Framework (JAF).

Abstract layer includes classes, interfaces and abstract methods which support all the mail handling functions. The implementation layer refers to internet implementation that uses internet standards such as RFC822 and MIME etc. The next layer is JAF that handles message data. With such an architecture, applications can send, receive and store a variety of messages in many different message system implementations.

5.2.2.2 The mail handling function implementation

The typical message handling process includes the following steps: (1) creating a mail message; (2) creating a *Session* object; (3) sending the message; (4) retrieving the message; (5) carrying out some operations on the message.

In the next section, we discuss each step in details: what is the class about, how it is created and interactions between them. Figure 5.3 illustrates this process:

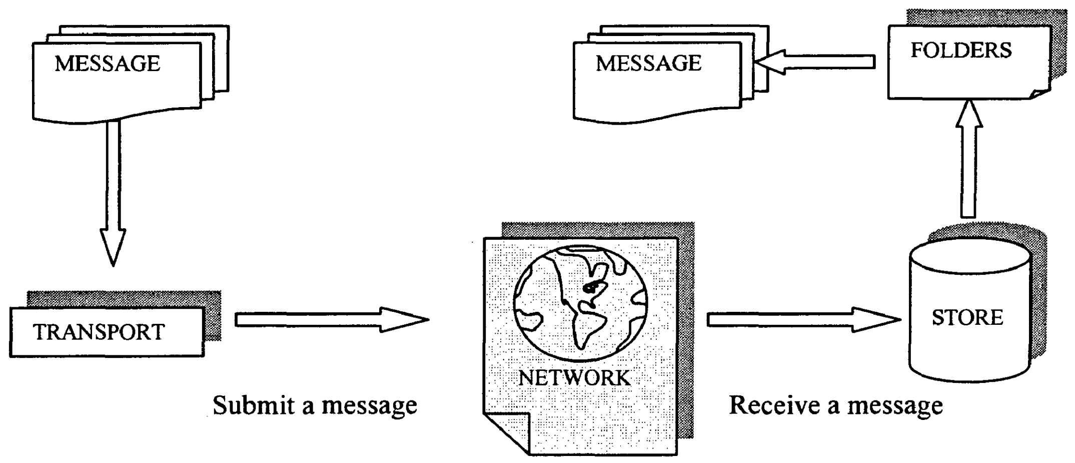


Figure 5.3 The Handling Function

We use such objects in the mail function: *Message*, *Transport*, *Store* and *Folder* as shown in the above figure.

5.2.2.3 Implementing JavaMail API

This section includes the major classes we use and how they are implemented in our agent system. There are five classes we use: *Message* class, *Folder* class, *Store* class, *Transport* class, and *Session* class.

The *Message* class

The *Message* class is an abstract class which specifies the attributes and content for a message. The attributes of a message include the address information and the structure of the content. The address information is used for message transportation: examples are *From*, *To*, *Subject*, *Reply-To* etc. The content of a message is a collection of bytes.

The *Folder* class and *Store* class

The *Folder* is the container for messages or subfolders. The methods of fetching, appending, copying and deleting messages are defined in *Folder* class. The *Store* class is the database that contains folders. The access protocol is also defined in this class such as IMAP, POP3 etc.

The *Transport* class and *Session* class

The *Message* object is sent by calling *Transport.send* method, the *Transport* class sends messages to the recipients and models the transport agent as well. The *Session* class is the container for *Store* and *Transport* objects. The way to obtain *Store* and *Transport* is to call the functions in *Session* class. The *Session* object specifies such properties that define the interface between client application and the network.

5.3 The Navigation of Principal Functions

In this section, we navigate our intelligent agent work flow in the order of user perceptions. This process is partitioned into several phases, starting from first log in account, checking messages in mail account, until the notification when there are new messages in the account.

Phase I: Start the application

In order to start this agent, user should input the command line like:

C:\java MailAgent –L pop3://username:passwd@servername

In more details, if a user opened a Yahoo mail account with user name as e_talking, then server is: pop.mail.yahoo.com and account is:e_talking@yahoo.com

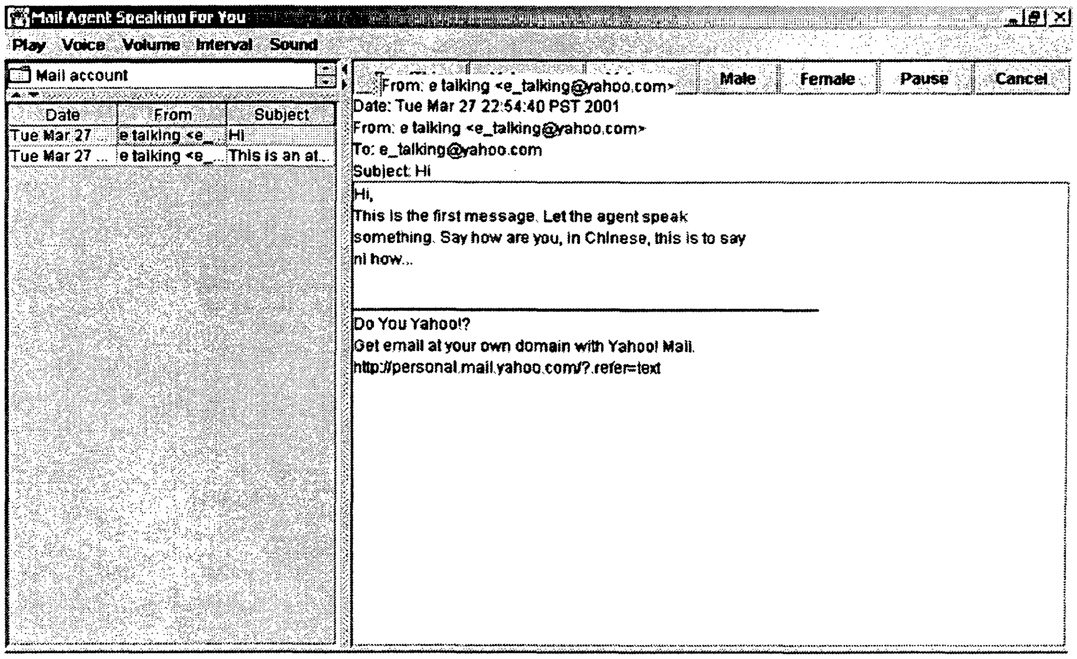
If the agent has completed the connection with server, the agent works for user. The agent speaks a short introduction via speaker including:

"Welcome!"

"Mail Agent Speaking For You"

"and now you have *n* total messages in your box"

n is message number in user INBOX folder. INBOX folder is the one that most protocol put messages in. At the same time, the main interface of our agent is shown as indicated in Figure 5.4.



Date	From	Subject
Tue Mar 27 ...	e talking <e_	Hi
Tue Mar 27 ...	e talking <e_...	This is an at...

From: e talking <e_talking@yahoo.com>
Date: Tue Mar 27 22:54:40 PST 2001
From: e talking <e_talking@yahoo.com>
To: e_talking@yahoo.com
Subject: Hi

Hi,
This is the first message. Let the agent speak
something. Say how are you, in Chinese, this is to say
ni how...

Do You Yahoo!?
Get email at your own domain with Yahoo! Mail.
http://personal.mail.yahoo.com/?.refer=text

Figure 5.4 The Voice Email Agent Screen (1)

The agent screen is divided into three parts: in the left upper corner, the account is shown; in left down corner, the messages in the user INBOX folder is shown including Date, From and Subject. In the right of the screen, the content of selected message is shown here. The agent fetches all the messages in the folder and the header of each one is displayed. Therefore user can go through all the messages. By clicking any of the messages, we can look at the content in right screen. If user wants to listen to any message, there are two alternative ways: either by a short cut button shown on upper right part *ReadThis* or by clicking *Play* button on the main menu. The functions associated with the Speech

Synthesizer such as voice type (either male or female), the age (young or aged), the volume, how long of interval to check new messages, and what kind of music notifications could be selected by clicking the corresponding buttons on the main menu. The short cut buttons are also shown on the right of the screen.

Phase II: Check new message

As long as the agent is active, it checks if there are any new messages in our folder. If there are any, a visual notification is also shown on the screen as shown in Figure 5.5. At the same time, the agent will play music as notification. The agent reads the message based on the instructions from user. If user wants to hear it then just says yes. The agent identifies various types of messages such as plain/text, multipart and nested message. Then it also speaks a notification such as:

"Hello e talking, you have *n* new message"

" Message 1"

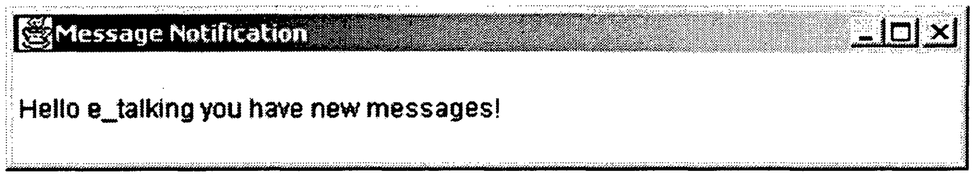
"This message is sent by *sender* and the subject is: *subject*"

. . .

" Message *n*"

"This message is sent by *sender* and the subject is: *subject*"

n is the number of new in the folder, sender and subject are retrieved from the new just arrived.



Hello e_talking you have new messages!

Figure 5.5 The Voice Email Agent Screen (2)

According to the type of incoming message, the agent speaks different notification accordingly. For example, if the message is a text/plain type, then the agent speaks:

" This is a plain text message"

"The content is: ......"

If the type is multipart type, the agent speaks

"This is a Multipart message with attachment"

"The content is: ......"

"This is attachment"

"The content is: ......"

If it is message/rfc822 type, then it speaks:

"This is a Nested Message"

"The content is: ......"

If none of above, it speaks:

"Ops, I can not speech for you, could you have a look?"

The new message is shown in left down corner by clicking upper left account name. This is to refresh the messages in the user INBOX folder. Then the new one is shown including Date, From and Subject. Therefore our intelligent agent is able to take care of the incoming messages while the user is concentrating on some other duties. With the help of an intelligent agent, the user is able to focus on his/her own interests while not delay the handling of any new messages. Because user can select how long to check new ones, our agent acts based on a set of instructions given by the user.

The background algorithm for checking new messages is shown in the following:

```
Open folder INBOX with Folder.READ_ONLY
Get total message number by calling folder.getUnreadMessageCount()
Speech "Welcome! "Mail Agent Speaking For You, and now you have n
        total messages in your box"
Close folder
Open folder with Folder.READ_ONLY
Get preMessages = folder.getMessageCount()
Close folder.close
While ( true)
{
```

```
If folder is not open
        Then open folder
Get totalMessages = folder.getMessageCount()
Get newMessages = totalMessages - preMessages;
If newMessages is greater than 0
{
        Get userName = urln.getUsername()
        //taking "_" out from user name
        Get easyName = userName.replace('_', ' ')
        Speech "Hello " + easyName + "you have " +
                newMessages + "new message"
        Get all the messages in INBOX me = folder.getMessages()
        Get number of messages: j
        for ( i = newMessages; i > 0; i--)
        {
                Get subject sub = me[j-i].getSubject()
                Get fromString
                Speech "Message " + n + "This message is sent by"
                        + fromString
                Speech " and the subject is: " + sub
                Then process the content of each: me[j-i]
                }
            }
            preMessages = totalMessages;
        }
```

A detailed representation of interactions between three layers is shown in Figure 5.6. Whenever there are any new messages or we log in our account the first time, the agent handles this function in its account process layer as indicated by (1). If this is a new one, the account process layer invokes notification layer that process notification function according the properties predefined as indicated by (2). Upon completion, notification layer calls speech integration layer to speech the messages to user as indicated by (3). If user just wants to listen to the content by clicking certain buttons, account process layer notifies speech integration layer directly without invoking notification layer as indicated by (4) in the Figure 5.6. When users start our agent, they has no idea that there are three layers cooperate background. The impression they have is a whole application with audio feature.

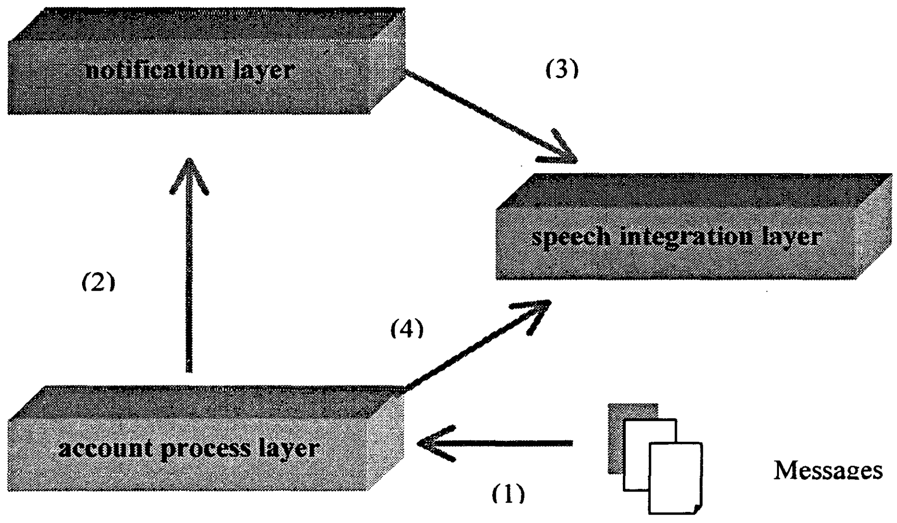


Figure 5.6 The Interactions Between Layers

5.4 A Brief Summary

In chapter 5 we discussed our agent implementation details including how the techniques in chapter 3 are implemented including Text-to-Speech Synthesis (TTS), Internet email protocol, Object-Oriented design theory etc. With the completion of implementation, our agent can check if there are any new messages in our folder. If there are any, the agent reads the message for us. It can identify various types of messages such as plain/text, multipart and nested message. A visual notification is also shown on the screen. With both visual and voice notifications, our agent becomes a multimedia intelligent agent. We can also select voice type (either male or female), the age (young or aged), the volume, how long of interval to check new messages etc. Our Voice-Email intelligent agent system delivers abbreviations, punctuation, time and date formats, and many other potentially ambiguous texts in a way that is easily deciphered and correctly spoken.

Chapter 6 Conclusion and Future Work

In this section we discuss the significance of our work and the improvements we should consider in the future work. The significances part focuses on how our work makes new efforts in the respective areas.

6.1 The Significances of Our Works

As we have indicated that intelligent agent is an intelligent program that uses intelligent agent technology to exchange information for automatic problem solving, performing specific tasks on behalf of their users. In our work we investigate what is need for an intelligent agent and use our results to design such an agent and implement it. Such an intelligent agent work is based on our design framework and works for us. As already shown, such a multimedia email system monitors the email for us while we are surfing the Internet and read the message for us. It notifies any new incoming messages not only with text notification but also voice notification. Such an intelligent agent is developed based on our design framework. The intelligent agent system combines network protocol, voice technology and theory, electronic mail protocol, Object-Oriented design theory and Java API.

6.2 Our Work In A Broader Sense

As we had indicated in chapter 2, our work is classified into reactive agent category. Such kind of agents responds to the present environment where they are embedded in a stimulus-response manner, monitoring outside by sensors and utilizing data from sensors. One of the applications is robot that is a physical agent. The subsumption architecture is suitable for reactive agent of robot. Our work extends such architecture to agents that are software applications together with utilization of OO technology. Therefore the goals of different layers in subsumption architecture are achieved more easily and we can extend our agent more conveniently. This is our new efforts for the subsumption architecture.

Based on what we have done, we can extend our agent capabilities. For example, two mail agents should be able to communicate each other to achieve more complicated task. If there are any important messages, one agent can forward them to another agent that can reach users. One agent is in office server and another one is in home computer. By Internet connection, agents are able to communicate each other over Internet. Each agent stores the IP address of another one. Because of Internet volume, user can set some criteria to allow agent fill out some messages such as large size ones, or the ones with certain sender. The agent is also able to forward message without attachment if that attachment is too large. Figure 6.1 shows this scenario. We can even go further by adding more layers to let agent navigating on the Web. The agent can find a file and send that file back to user.

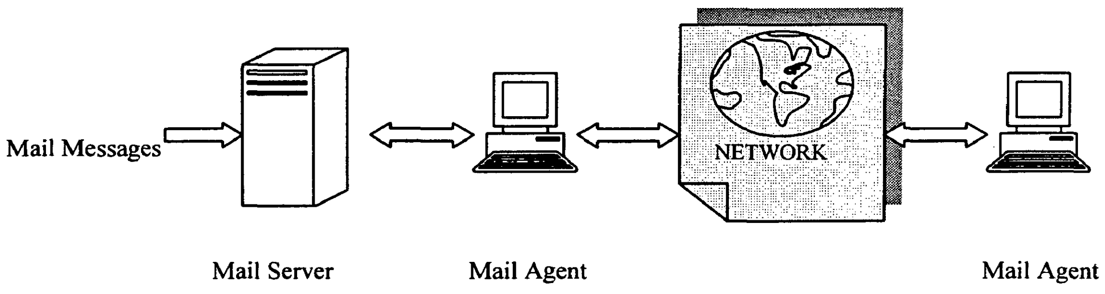


Figure 6.1 The Forward Features Between Agents

6.3 Future Work

In previous chapter, we have discussed our agent system, including our objectives, initiatives, its theoretical background, the whole model design and its implementation details. Though the agent realized our objectives through implementation, it need further improvements and there are still challenges ahead such as:

(1) How to expand the agent architecture to more flexible applications;

(2) How to improve our design to facilitate further development of intelligent agent;

(3) How to improve scalability and performance etc.

As long as we put efforts on these issues, we will have a more flexible and efficient intelligent agent system.

References

[ATT, 2000] http://www.research.att.com/areas/

[BELL, 2000] http://www.bell-labs.com

[Bradshaw, 1997] Jeffrey M. Bradshaw, *Software Agents*, MIT Press 1997

[Brooks, 1986] Rodney Brooks, *A Robust Layered Control System for a Mobile Robot*, IEEE Journal of Robotics and Automation, April, 14-23.1, 1986

[Brooks, 1990] Rodney Brooks, *Elephants Don't Play Chess*, In Pattie Maes, ed., Designing Autonomous Agents, Cambridge, MA: MIT Press 1990

[Chen, 1999] Z. Chen, Intelligent agents, *The IEBM Handbook of Information Technology in Business*, pp. 561-569, 1999

[Eckel, 2000] Bruce Eckel, *Thinking in Java*, Prentice Hall 2000

[Franklin, 1996] Stan Franklin, *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag 1996

[IBM, 2000] http://www-4.ibm.com/software/speech/

[IBMWP, 2000] http://activist.gpl.ibm.com:81/WhitePaper/ptc2.htm

[Karnik and Tripathi, 1998] Neeran Karnik and Anand Tripathi, *Agent Server Architecture for the Ajanta Mobile-Agent System*, in Proceeding of the 1998 International Conference on Parallel and Distributed Processing and Application (PDPTA'98), Las Vegas 1998

[Maes, 1995] Pattie Maes, *Artificial Life Meets Entertainment: Life like Autonomous Agents*, Communications of the ACM, 38, 11, 108-114 1995

[MSFT, 2000] http://www.microsoft.com/enable/dev/default.htm

[Nwana and Azarmi, 1997] H.S. Nwana and N.Azarmi, *Software Agents and Soft Computing,* Berlin: Springer 1997

[Russell and Norvig, 1995] Stuart J. Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Englewood Cliffs, NJ: Prentice Hall 1995

[SUN, 2000] http://java.sun.com/products

[Wood, 1999] David Wood, *Internet Email*, OReilly 1999

[Wooldridge and Jennings, 1995]M. J.Wooldridge and N. R Jennings (eds.) *Intelligent Agents*, Berlin: Springer 1995