INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

Voice-Enabled Interactive E-commerce

by

Ramsey Hage, B.A.Sc.

A Master's thesis submitted to the
School of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of

Master's of Applied Science
in
Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

School of Information Technology and Engineering
University of Ottawa

Abstract

This thesis studies the application of speech recognition and synthesis technologies, agents and virtual environments in the design of a digital assistant, helping a user of web-based e-commerce applications. A useful component of the user interface in an electronic commerce system is a speech user interface. This interface will contain a user's agent and serve an active role in recognizing a user's speech input as well as synthesizing the required speech sentences needed to be conveyed. The Speech GUI implemented in this thesis complements the web browser during virtual world e-commerce. The virtual world being researched in the MCRLab is a shopping mall, where users can enter a store and find sales agents ready to assist them. Each person within this virtual environment is represented with the use of an agent and avatar. The virtual environment is a multi-agent system. These agents (also represented by avatars) can navigate through the virtual environment such as a shopping mall, and interact with each other. Once dialog has occurred between a user's agent and the store's sales-agent and an item of interest is found, this item will be displayed in the virtual mall. The virtual environments technology, with the help of avatars will allow us to have good voice QoS coupled with synchronized avatar lip movements.

The ultimate purpose of the Speech GUI is to assist a user in finding items of interest within a virtual shopping store. Also, when the items of interest are found, this Speech GUI will ask the web browser to display the objects within the virtual store. Communication for this is achieved using the HLA/RTI available standard. The second type of communication associated within the Speech GUI uses the KQML protocol to communicate with other peoples' agents.

Acknowledgements

The evolution of this thesis was not a solitary effort. Many people assisted me in both the development of the code and in the written portion of the thesis.

First and foremost, I would like to thank Dr. Georganas who has been both my teacher and mentor throughout the project. He has taught me that through hard work and determination one can achieve their goals.

I would also like to acknowledge Sergei Mankovski and Babak Esfandiari from Mitel Corporation who have assisted me in the completion of this thesis. This thesis implementation is based on Sergei's Tuple Space program, developed for message exchanges between two systems. Sergei was gracious enough to take the time and explain the features of this program. Babak's expertise was also helpful in allowing the Speech GUI implemented to be speech aware.

Past and present members of the MCRLab team were also helpful, especially Xiaojun Shen who assisted me in creating mechanisms necessary for communication exchanges between my Speech GUI and the web browser. This communication was the HLA/RTI standard, which allowed me to broaden my research knowledge.

Another necessary part of this thesis component are the editors involved. I am grateful for all the help and knowledge provided by Vasilios Darlagiannis and Shannon Hurley. Their help on this thesis was invaluable and resulted in a successful outcome.

Finally, my family has also been a constant inspiration to me and has given me the necessary support to accomplish my objective.

Table Of Contents

List Of Figures

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
ARPA	Advanced Research Projects Agency
BNF	Backus Naur Form
CLIPS	C Language Integrated Production System
COM	Component Object Model
DARPA	Defense Advanced Research Projects Agency
DB	Database
DCOM	Distributed Component Object Model
DMSO	Defence Modelling and Simulation Office
DVE	Distributed Virtual Environment
EAI	External Authoring Interface
EBNF	Extended BNF
FOM	Federation Object Model
GUI	Graphical User Interface
HLA	High Level Architecture
JESS	Java Expert System Shell
JSAPI	Java Speech Authoring Programming Interface
JSGF	Java Speech Grammar Format
JSML	Java Speech Markup Language
KIF	Knowledge Interchange Format
KQML	Knowledge Query Manipulation Language
KSE	Knowledge-Sharing Effort

QoS	Quality of Service
RTI	Run-Time Infrastructure
SAPI	Speech API
SDK	Software Development Kit
VKB	Virtual Knowledge Base
VM	Virtual Machine
VRML	Virtual Reality Modelling Language
WFC	Windows Foundation Classes

Chapter 1

1 Introduction

Shopping experiences can sometimes be an unpleasant task when one has to deal with all the hustle of crowded malls. Shopping from one's home has been successful in the television medium and there is growing popularity in purchases over the Internet. A shopping experience should be very user friendly. We have progressively become a fast paced society and the creation of a medium where customers can shop at any moment is convenient for our lifestyle. Technology has been progressive with many touch-tone services now available for the telephone and form filling over the Internet. These systems create much aggravation in having to wait through a series of menus. One solution to this problem is the integration of speech aware applications. This will provide the ability for the user to interact directly with the purchasing medium through a natural language interface. With the help of the Internet, new worlds can be built using virtual reality. This provides the ability to process many kinds of virtual applications for commercial or private use. This thesis attempts to assist commercial uses of virtual reality by providing a speech-aware application. By doing so, this application can be put alongside a web browser to aid the on-going shopping experience by a user.

The evolution of the Internet was gradual and led to the possibility of having virtual reality applications. Initially, Internet traces its origins to the ARPANET, an experimental network begun in 1969 as a project of the Defense Advanced Research Projects Agency (DARPA) [45]. Now, Internet is one of the fastest growing industries in the world. In fact, in the past few months industry has recorded the biggest "boom" in the amount of new users signing on and predictions made state that by the end of the year, at least 90% of the world will be connected to it [46]. This industry helps every large company, television broadcasters,

education institutions, and others to remain competitive in the marketplace by promoting products through it. For this, each will have personal web addresses that can be viewed by customers and guests. Evolution of this technology is in a constant state of change. The latest advancements in this field give users the ability to navigate within 3-dimensional worlds. This is possible through software helpers, called plug-ins. This technology would not have succeeded the way it did if the computer industry did not as well evolve into the fast machines that are now available.

These different technologies were put in use in this thesis research. The main focus of this thesis is to use the web-browser, running a 3-dimensional virtual mall, and add voice controlled commands to it - directing the actions to be made within the virtual mall, for example, going to a specific store within the mall. At this request, the world within the web browser should change the background scenery to reflect the user's demands. Another need for speech aware applications is for a user to interact with a sales agent within a store to purchase items of interest. With use of the Internet, item payments can be made through secure forms supplied by the store that a user can fill-out.

Continual research in the field of virtual reality is being addressed by educational, industrial, and government institutions. Applications in medical, industrial, and educational training are many. Virtual reality provides many advantages over the traditional ways of behaviour. For example, to a shopper, there is no need to physically travel to a store to purchase items of interest.

For this research, the virtual application that needed to be made speech-aware is a shopping mall experience. This project is meant as a Distributive Virtual Environment (DVE) and involves many different parts from different people involved in this project. Some of the parts come from within the lab, such as the creation of the virtual mall, and others come from different sites throughout the country. To accomplish this, different standards are being researched, and one specific standard was used for the communication mechanism when developing this thesis' Speech GUI, namely the High Level Architecture – Runtime Infrastructure (HLA/RTI).

In order to accomplish speech awareness, a speech engine needed to be employed. For this, a free Speech API was found from Microsoft, whose engine provides speech synthesis as well as speech recognition. This engine provides developers with many different features such as the documentation of different Application Programming Interface (API) platforms, continuous speech recognition capability, ActiveX support for visual development, and many more.

The use of associating an agent to each user is the key behind inter-agent speech conversations. The client agents take the information provided by the shoppers and relay the speech inputs to the server-agent, representing the sales clerk. The intelligence behind the sales-clerk is based on an artificial intelligent program, written to portray the objects within the store. Each object will have certain properties associated with it. By means of an expert system, the server-agent will ask the user, through a series of questions, what information is required from the store. Following this set of questions, an object will be deemed to be desired by the user and displayed to the user before its purchase is completed.

The motivation behind this work is based on the fact that the virtual environment technology has been a focus of study with no substantial research work provided for voice-activated demands. Though, credit must be given to the fact that the currently available virtual worlds provide very good 3-D object rendering with the use of very powerful graphic accelerator cards, combined with inexpensive CPUs. As a result, virtual environments have evolved into a widespread technology that is used for training and collaboration. The addition of voice commands within the virtual world gives the user a more natural means for treating tasks needed to be accomplished within the virtual environment. Communication between multiple agents could be realized to enhance the virtual world. With each agent entering the virtual world, there will an associated profile of the user's needs, preferences, and special needs required. This will aid the agent to conduct speech dialogs with other agents based on these criteria.

Following this introduction, different chapters discuss the parts within the thesis. Chapter 2 will present all background information necessary to accomplish this thesis implementation including the Java technology, the Virtual Reality Modelling Language technology, the HLA/RTI standard, as well the KQML protocol used for inter-agent communication within the shopping mall.

Having discussed the background material, Chapter 3 discusses the project this thesis belongs to, named Enabling Technologies for E-Commerce project. This is a major project supported in part for the Canadian Institute For TeleCommunications Research (CITR). Chapter 3 also mentions similar projects researched elsewhere.

A system's architecture is presented in Chapter 4. This provides the different architectures of the client, as well as the servers used and provided from Mitel Corporation.

Having discussed this, a treatment of agent-communication is discussed in Chapter 5. The different aspects of agent-communication such as ontologies associated with the messages formed using the KQML protocol. Within this chapter, the different connections to be made by the clients as well as the server will be presented.

Before concluding this thesis, a chapter discussing the speech engine is written and provides several example grammar file formats that should be created by a sales agent in order to understand the user's speech input to queries asked of them.

Finally, a conclusion is presented with a discussion of the overall thesis implementation and summary.

This thesis achieved all its objectives and can be extended in the future for bigger projects concerning distributed virtual environments.

The direct contributions of this thesis are:

- The integration of speech recognition technology with a virtual environment e-commerce application; and
- The implementation of inter-agent communications in support of user-independent speech recognition.

Chapter 2

2 Background Material

Key elements need to be discussed in this chapter. In order to fully understand the scope of this thesis, a discussion of the technologies encountered is briefly presented. Further details about these technologies can be found in the references that will be mentioned in the next few sections.

2.1 Java

With the growing popularity of the Internet, there is an emergence of programming languages such as Java, that enables software developers to program platform-independent code. The Java programming language is developed by *SUN Microsystems Inc.*[1] and provides programmers with conventional programming concepts as well as specialized support of *Applets*. A definition given by JavaSoft [28] says that "An applet is a program written in the Java™ programming language that can be included in an HTML page, much in the same way an image is included". Since applets are programs that are downloaded and executed on a user's computer system, security issues must be defined by *SUN Microsystems Inc* within the Java specifications manual. The execution of these applets are done within a virtual machine (VM). The Java VM is an abstract computing machine. Like a real computing machine, it has an instruction set and manipulates various memory areas at run time [29]. The Java language is used whenever secure, modular, and portable codes are needed. The Java language is ideal for those cases.

[1] The software can be found at http://www.java.sun.com/ or http://www.javasoft.com/

A Java program is both compiled and interpreted. With a compiler, you translate a Java program into an intermediate language called Java bytecodes - the platform-independent codes are interpreted by the Java interpreter. With an interpreter, each Java bytecode instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed [19]. Figure 2.1 demonstrates how this works.

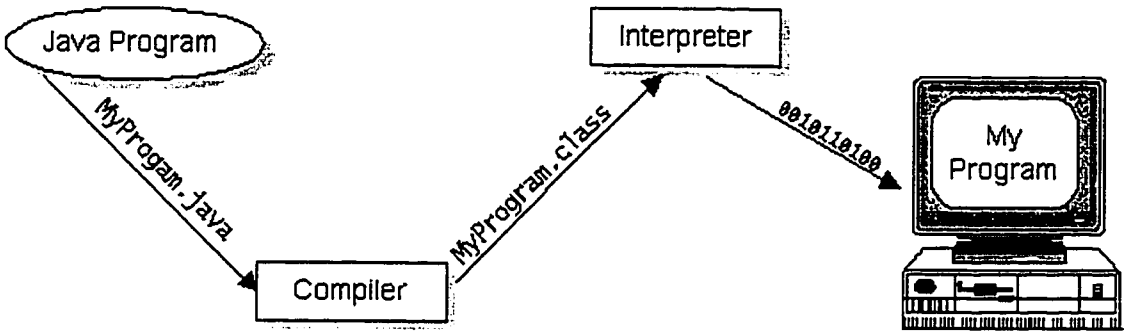


Figure 2.1: Steps To Run A Java Program

The interpretation of the code is done within the VM. Therefore, the virtual machine must implement the instruction set and direct computation to proper functional units. This is accomplished with software implementations, although hardware implementations are also conceivable. A diagram showing the process of attaining execution is taken from [20] and reproduced in the following figure.

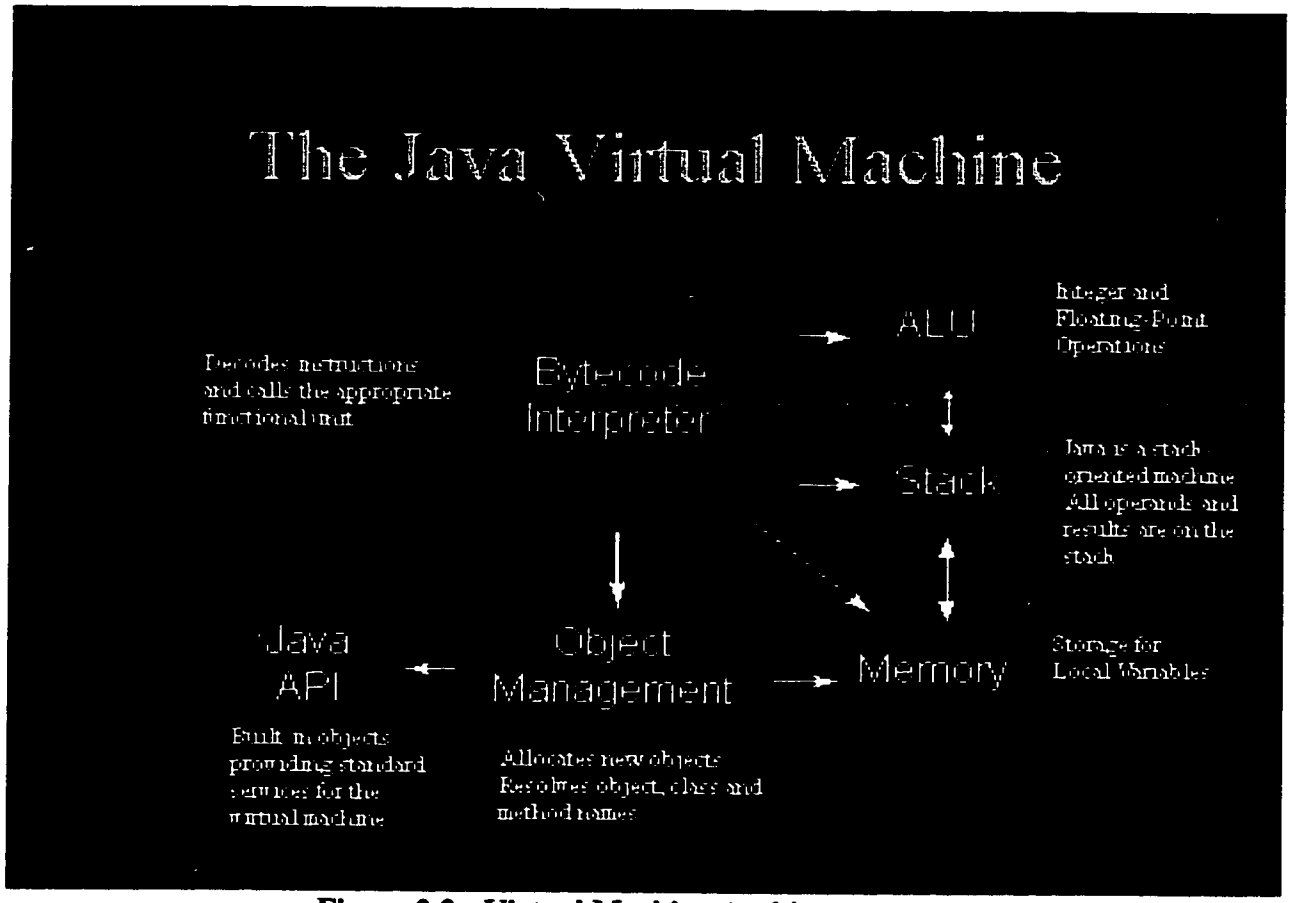


Figure 2.2: Virtual Machine Architecture

Java programs have security restrictions imposed on Java Applets. These applets are downloaded and executed inside a web browser. So, the VM has to comply with the Java standards to disallow programs of doing malicious actions on the user's system. More on applet security issues can be found on the Java web site [11]. On the other hand, applications have more freedom than applets. They cannot, however, be executed inside a web browser. Consequently, applications require the user to have installed the Java software, as well as the program to be executed.

Certain companies, like Microsoft, have developed Java-like software that substitutes for Java. For instance, they developed *Visual J++*[2] that enables programmers to use Windows Foundation Classes (WFC) and visually insert components into their Java program. This tool was used to develop the Speech Graphical User Interface (GUI) in this thesis, since it

[2] Visual J++ Homepage at URL http://msdn.microsoft.com/visualj/default.asp

allowed for insertion of Microsoft ActiveX components. A discussion of ActiveX components used is presented in section 2.3.

2.2 Virtual Reality Modelling Language (VRML)

Originally, VRML was based on Silicon Graphics' Open Inventor modelling library [30]. This library is an object-oriented toolkit for developing interactive, 3D graphics applications [31]. The development work continued at Silicon Graphics until a proposed Request-For-Proposals was adopted, with some minor modifications, as the VRML 2.0 specification. This specification, currently available, for VRML was approved in August, 1996 [32, 33].

The VRML world is described within a text file format and allows 3D interactive objects and worlds to be experienced over the world wide web. Thus, object rendering, manipulation, and interaction between users can be specified using VRML [22].

For the purpose of creating a virtual world that can be rendered inside a web browser, a VRML plug-in prevailed as the solution and was created to accompany web browsers. Plug-ins are modules that are separate from the main application. They are dynamically loaded upon startup and when needed, depending on the circumstance [21]. VRML plug-ins can be downloaded for the web browser from the internet[3]. With the help of the plug-in, the web browser will understand the VRML file and display the objects within it properly.

The VRML language is structured within a hierarchy that groups similar entities together to form collections of different sets. One category of these, useful for doing 3-D graphics, is called "Nodes". Under this category, one has many different kinds of nodes: the transformation nodes; the sensor nodes; as well as many others. VRML also provides Java programmers control of these nodes in two different ways. Both methods are discussed in [23]. The first is the Java External Authoring Interface (EAI) and documentation of this

[3] One possible download of a VRML plug-in is located at
http://www.cosmosoftware.com/download/player.html

interface is found at the VRML organization web site[4]. The second method is to associate object behaviours via scripts.

The file format of a VRML file may contain many things [24] including:

- A file header
- Comments: notes
- Nodes: nuggets of scene information
- Fields: node attributes one can change
- Values: attribute values
- Node Names: names for reusable nodes

These are written in an ASCII file, describing the virtual world being designed. An example of this file format, along with how it looks like, is presented in Figure 2.3.

```
#VRML V2.0 utf8
# A Cylinder
Shape {
    appearance Appearance {
        material Material { }
    }
geometry Cylinder {
    }
}
```

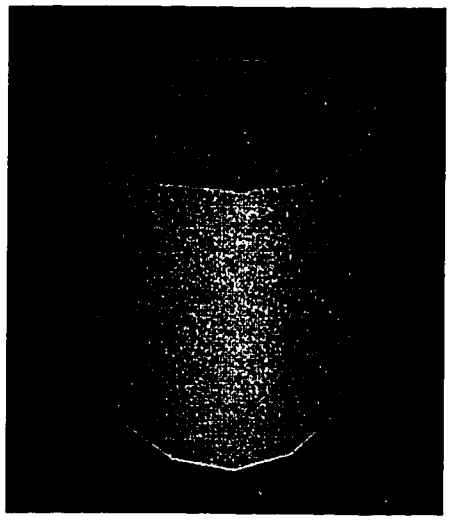


Figure 2.3: VRML File Format & Representation

[4] The EAI Reference Manual can be found at http://cosmosoftware.com/developer/moving-worlds/spec/ExternalInterface.html

In the context of the Speech GUI implemented for this thesis, the need to display VRML objects might be required in the virtual mall on a shopper's demand. The program will recognize speech sentences spoken by a user and react accordingly to the spoken words.

2.3 Microsoft ActiveX Component Use In Speech GUI

With the Component Object Model (COM) and the Distributed Component Object Model (DCOM) technologies developed by Microsoft, the COM-based ActiveX Control technology evolved. ActiveX object controls are encapsulated inside a shell to enable re-use of the objects by developers. This provided feature is very useful for developers since it does not require them to re-code certain objects and their attribute characteristics.

It is important to discuss ActiveX components accompanying speech engines since they are provided to developers with the downloaded or purchased speech engine. For instance, the Microsoft Speech Engine provides components representing a mouth that can synchronize the lips along with the text that is being synthesized. The ActiveX technology is very useful in order to have a good and quick representation of the application in the developer's mind. For this thesis, the Speech GUI was the application to create. It uses two ActiveX components, the *TextToSpeech Mouth* and the *DirectSpeechRecognition Ear.*

According to the a document entitled *ActiveX and The Web: Architecture & Technical Overview*, written in Microsoft's web site[5], ActiveX technology does not compete with Java. Rather, it tries to integrate technologies that can be used in any language, platform, or operating system. Therefore, since one of the components included in the Microsoft speech engine provides capability for lip synchronization with textual synthesis, the use of Microsoft Visual J++ was the development platform chosen along with the use of the ActiveX components. As a consequence of this choice, some problems for communications with the other implemented JDK SUN Java programs developed in the lab may arise due to

[5] Charlie, Kindel. ActiveX and The Web Architecture & Technical Overview, URL at http://www.microsoft.com/com/slides/kindel2.zip

the fact that the thesis implementation executes within a separate process than the other applications.

Figure 2.4 shows the ActiveX controls that were used for implementing this thesis. When the speech synthesis is in progress, the lip movements are synchronized along with the text that is heard. As well, the mode of the speech engine always alternates between the ActiveX controls. When the engine is in the state of recognition, the user will see the "ear" show-up in the GUI, and the lips will not be shown, and will await for a recognized word that will be found in the grammar used.

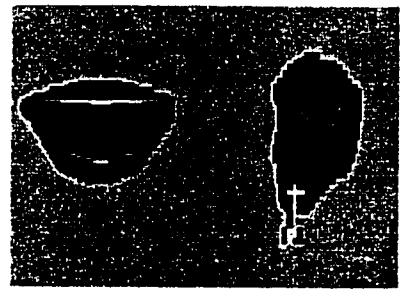


Figure 2.4: ActiveX Controls Used in Speech GUI

2.4 The HLA/RTI Standard

The ActiveX control, as described in the previous section, recognizes the words spoken from the grammar. It is at this point that a communication mechanism is needed to send the message to the web browser to do appropriate actions based on the words recognized. For example, it can show a 3D object model of an item a shopper would like to view. MCRlab is conducting research within different available Distributed Virtual Environment (DVE) standards. Therefore, the solution of the communication is based on one of the available standards. For this, a consideration is taken to see what is being done in the other DVE projects. Since this team is using the HLA/RTI standard, this will be the mechanism that will be used in the thesis for DVE communications.

The High Level Architecture (HLA) is a general architecture for simulation reuse and interoperability [8] developed by the US Department of Defense. The conceptualization of

this High Level Architecture led to the development of the Runtime Infrastructure (RTI). This software implements an interface specification that represents one of the tangible products of the HLA. The aim of it is to develop, under the leadership of the Defense Modeling and Simulation Office (DMSO), reuse and interoperability across large numbers of different types of simulations developed and maintained[6] [8]. Real-time interactions between entities in a collaborative virtual environment are implemented over RTI of the HLA. This HLA architecture is an upcoming IEEE standard for distributed simulations.

With this standard, communication from the Speech GUI with the e-commerce applet running inside a web browser is achieved. A complete section for this communication is dedicated and discussed farther down in this thesis.

2.5 Speech Recognition

Speech recognition technology is currently a popular field of study. With improved research, speech algorithms will mature and allow more efficient and accurate speech engines. There is much public interest and many users can be found for it, in both public institutions and private industries. Examples of businesses implementing speech recognizers are banks, telephone companies, movie theatres, computer companies, as well as many others. Speech synthesis is beneficial for many kinds of applications such as interactive games, web hosts, videoconferences, chat rooms and theme parks [16].

Speech recognition and speech synthesis provide a more natural interface for users. It frees the users hands for other tasks and allows the users to take advantage of their natural voice communication skills [14].

Synthesized speech can be categorized as realistic or fantasy. Realistic synthesized speech attempts to be as human-like as possible; while fantasy synthesized speech often sounds non-human, but is understandable [15]. A common method used to synthesize speech while

[6] URL at http://hla.dmso.mil/hla/

giving computer prosody[7] is to use phonemes. In order to synthesize a voice, samples of that voice need to be known and analyzed for its phonemes. As a consequence, engines need to be trained whenever new voices are needed for synthesis. The phonemes of a given voice are then put together to form sentences, simulating the human vocal tract. This technique allows the speech engine to speak any word; even invented words.

Speech recognition is somewhat harder to implement. There are several characteristics to consider with any speech engine [34, 35]:

- Continuous versus Discrete: This will define whether words can be spoken in a natural manner or whether pauses must be inserted between words to distinctly make out a word.
- Vocabulary size: The variability in size of the engine's vocabulary determines how much processor power is involved in the recognition process of words spoken out. A vocabulary must be specified since the engine must be able to match words heard with the ones available in the vocabulary.
- Speaker Dependency: Consideration of whether the engine must be trained to a user's voice must be done. An engine can be tuned to hear a certain voice. This will increase recognition accuracy when the engine is trained prior to its use.
- Acoustic ambiguity, confusability: The surrounding environment affects speech engine accuracy. Spoken language can be somewhat ambiguous. The meaning is not always clear for machines while recognizing speech. For example a computer may not be able to distinguish between "youth in Asia" and "euthanasia" without prior knowledge of context.
- Environmental Noise: Surrounding noise will affect the accuracy. The use of a headset may also improve recognition since the microphone is close to a speaker's mouth.

[7] Webster's dictionary defines prosody as the rhythmic and intonational aspect of language. This definition is obtained from http://www.m-w.com/netdict.htm

More sophisticated speech engines will take the above variables into account and provide user control over them. For instance, one might be allowed to modify engine parameters like confidence levels. This will define the confidence that the engine needs to have before concluding the word spoken. Additionally, probability models are incorporated within some speech engines. The role of probability for speech recognition is to specify the chances of word occurrences. The default settings of speech engines gives all words equal probability. Modifying this parameter tells the engine how often word occurrences may happen. For example, in the context of *Children Throughout The World*, "youth in Asia" is more likely to occur than "euthanasia". Although speech engines may give control over parameters to improve speech recognition, the drawback of this is that computation time is usually increased.

The process of speech recognition is still in the developmental stages. The capabilities of these engines are endless. Work in this field converges within different domains of study, including signal processing, pattern recognition, computer algorithms, phonetics, and more. It is important to note that speech recognition and synthesis technologies will be developed much more and will be used on a broad scale of applications. They might not take the place of the keyboard or the mouse but will enhance the already existing features.

2.5.1 Java Speech API

In order to implement a Speech GUI, a decision must be made regarding what speech engines to use. Since the Speech GUI is written in Java, the most logical choice is the Java Speech Application Programming Interface (JSAPI). Upon searching the Java web pages, we can find a specification of the JSAPI language[8] [13]. Appearing with the JSAPI specifications are its companions; the Java Speech Markup Language (JSML) and the Java Speech Grammar Format (JSGF). These give synthesizers cross-platform control of how the information stored should be synthesized, as well as give recognizers the ability to define, for all platforms, a general syntax of all the possible words that will be allowed to be heard.

---

[8] Java Speech 1.0 Specifications, URL at
http://www.javasoft.com/products/java-media/speech/forDevelopers/jsapi-doc/index.html

Even though SUN Microsystems Inc. has developed many specifications for speech recognition and speech synthesis, developers are obliged to rely on available implementations to enable their applications with speech. The Java web site lists two possible vendors who have implemented the specifications published by Java on their software[9]. They are IBM's "Speech for Java" for operating systems running the Windows platform, and "Lernout & Hauspie's TTS for Java Speech API" for operating systems running SUN Solaris.

The implementation that will be developed for the Speech GUI of the electronic commerce project in this thesis will be based on the Windows operating system since speech engines are platform dependant, and the PC platform is the most widely used. Therefore, if one of the implementations of the JSAPI is to be used, the one provided by IBM would be the obvious choice. However, this engine was not used since it requires the user's machine to have installed one of the IBM ViaVoice products. Therefore, another commercial product needed to be found for the implementation of the Speech GUI.

2.5.2 Commercial Products

In search of a speech engine, a commercial product needed to be found. It is clear that the JSAPI was not a possibility for the development of the speech application. It was seen that IBM was one of the big players involved in implementing the JSAPI, though it could not be used. This posed no problem if a free (through download) IBM ViaVoice product could be found. However, the search for the free download fell short.

The search then continued to locate other vendors. Another leading force in speech technology is the Dragon Software. Similar to IBM, this company sells a product suite that is speech aware. Though, since this is a complete application, without Software

---

[9] Vendor implementations of the JSAPI specification can be found at the URL
http://www.javasoft.com/products/java-media/speech/forDevelopers/jsapifaq.html#implementation

Development Kit (SDK) support, and since no free download was found, this was not an acceptable option either.

Finally, the proper speech engine was found. It has a free download and is not based on any software, which allows developers to program custom applications. This is the Microsoft Speech API (SAPI). This download provides developers with three different APIs. They include an API for COM programmers, C++ programmers, and Visual Basic programmers. With the use of the Microsoft J++ development environment, the ActiveX components that are included in this free download are available for development use. It is also a good choice of speech engine, since many companies base their development on this one, or at least have support for programs written in this language.

2.6 Agents and Avatars

Agents can be defined in many different ways. Most commonly, agents are described as proactive, personalised, and adapted software that can act on behalf of people. Some applications will have graphical representations linked with the agent. This representation is known as an avatar. By knowing a user, their interests, habits, and goals [36] an agent can perform individualized tasks. Therefore, agents play active roles in helping users attain their goals.

Agents, as depicted by their avatars, run within applications. Internet based agents will typically run within a web browser using 2D or 3D user representation. With the use of modelling languages such as VRML, web-based agents will need to support three areas of agent operation. A depiction of this [37] is shown in Figure 2.5.

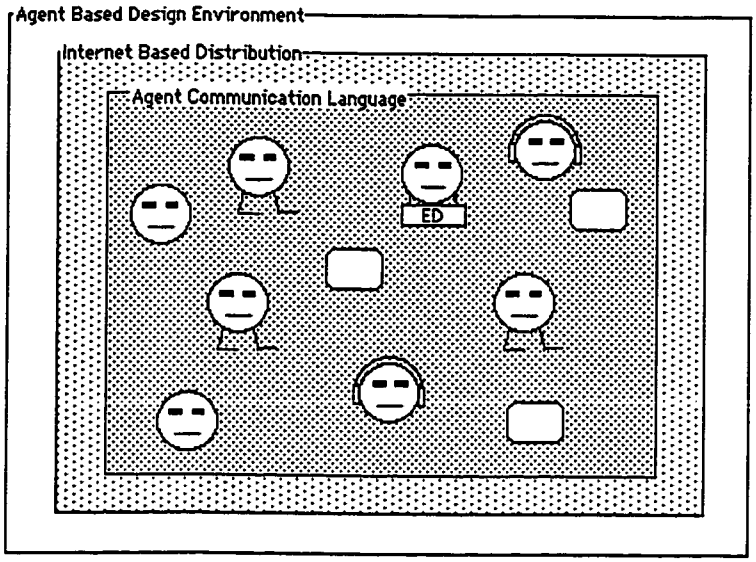


Figure 2.5: Implementation Layers for Agent Operation

In the outer level, *Agent Based Design Environment*, agents will be contained inside an application running within an adequate environment. For the DVE projects, developed at the MCRLab, the agent based environment is the web browser running a virtual shopping mall. This environment allows agents to display certain items of interest to a user, and based on user input, make a purchase if desired.

In the next level, named *Internet Based Distribution*, mechanisms must be implemented to share information between the agent and its user. Mechanisms for collaborative aspects of DVE would distribute information obtained to other user agents, where the display of the item can be shown. The means of distribution for attaining communication between multiple agents would use such standards as the Runtime Infrastructure of the High Level Architecture.

In the third and final level, *Agent Communication Language*, protocols need to be agreed upon in order to exchange information between agents. This protocol can be user defined or pre-existing. Communication for this thesis uses the KQML protocol discussed in the next section.

2.7 The KQML Language

The communication mechanism used to enable discourse between agents will be using the Knowledge Query Manipulation Language (KQML) protocol. This protocol is used since it offers sequential communication between agents and is ideal for agents connected through a database, such like the server agent. The sequential communication is provided since there are fields within KQML messages that have identifiers for ":in-reply-to" and ":reply-with". Therefore, agents can know answers to queries posed in a previous sent message. As well, shoppers may want to discuss with multiple agents at the same time. These servers may not be in the same location. Therefore, communication through KQML is allowed within the central Tuple Space medium shared between agents.

Although KQML is the underlying protocol used between agents, users will use natural language. The spoken words within the language are then encoded into a KQML message; where, it can be sent to other agents. This encoding is done within the client, where several issues need to be considered in order to inter-operate with other similar agent systems. They are:

- Language: what the individual messages mean;
- Transport: how agents send and receive messages;
- Policy: how agents structure conversations;
- Architecture: how to connect systems in accordance with constituent protocols.

2.7.1 KQML Language

KQML is a language protocol that is used to exchange information and knowledge between two or more systems. Typically, these systems will share knowledge with the goal of attaining a cooperative solution to the problem at hand. The specification of this language will be discussed in this section. The specification can be found at the Computer Science &

Electrical Engineering departments at the University of Maryland Baltimore City (UMBC) in the United States[10][5].

UMBC defines KQML in an extended Backus Naur Form (E-BNF) as shown in Figure 2.6. The convention used assumes the definitions for <ascii>, <alphabetic>, <numeric>, <double-quote>, <backslash>, and <whitespace>. The "*" means any number of occurrences, and "-" indicates set difference [1].

[10] http://www.cs.umbc.edu/kqml/kqmlspec/spec.html

<performative>	::=	(<word> {<whitespace> :<word> <whitespace> <expression>}*)
<expression>	::=	<word> | <quotation> | <string> | (<word> {<whitespace> <expression>}*)
<word>	::=	<character><character>*
<character>	::=	<alphabetic> | <numeric> | <special>
<special>	::=	< | > | = | + | - | * | / | & | ^ | ~ | _ | @ | $ | % | : | . | ! | ?
<quotation>	::=	'<expression> | ''<comma-expression>
<comma-expression>	::=	<word> | <quotation> | <string> | ,<comma-expression> (<word> {<whitespace> <comma-expression>}*)
<string>	::=	"<stringchar>" | #<digit><digit>*"<ascii>*
<stringchar>	::=	\<ascii> | <ascii>-<backslash>-<double-quote>

Figure 2.6: KQML string syntax in E-BNF[11]

The syntax in Figure 2.6 describes how KQML messages are formed. Messages can be created with variations. Within a message, some key parameters are reserved by the language specification. These are listed in Appendix A.

[11] This syntax is obtained from a draft specification of KQML Agent-Communication Language, June 1993. http://www.cs.umbc.edu/kqml/papers/#spec

Appendix B also defines the message performatives (the primitive types in the KQML language) and gives their descriptions. These messages describe an attitude about the information they contain: such as querying, stating, believing, requiring, achieving, subscribing, and offering. This attitude is conveyed in the content portion of the message. For example, if it is a subscribe message then the user requires a subscription to the agents' Virtual Knowledge Base (VKB).

New performatives may be defined by agents and are by no means limited to the ones listed. However, this does mean that the receiving agent must be able to decipher the message and handle it. The definition of the new message should contain [1]:

- The performative name;
- All parameters keywords that the performative may contain;
- Syntactic categories and semantics for all values of parameters with non-reserved keywords;
- Any additional syntactic and semantic constraints for values of parameters with reserved keywords;
- The default values of all absent parameters;
- The semantics, in terms of a statement the sender is making of itself, of the performative name applied to the parameters.

An example discourse between two machines (A and B) is taken as a scenario. Suppose machine A wishes to query Machine B's VKB, it can attain this with an ask-all message shown next.

```
(ask-all
:sender         A
:receiver       B
:in-reply-to    id0
:reply-with     id1
:language       Prolog
:ontology       foo
:content        "bar(X,Y)"
)
```

That is, machine A is asking machine B to send all the facts in the database about bar(X,Y). Machine A will expect a set of results in a list format such as [bar(a,b), bar(c,d)] as is defined by the language parameter. As well the message machine A awaits for the response of this query needs to have the ":in-reply-to" tag set as id1. The reply to such a message would be:

```
(tell
:sender         B
:receiver       A
:in-reply-to    id1
:reply-with     id2
:language       Prolog
:ontology       foo
:content        "[bar(a,b),bar(c,d)]"
)
```

These are the only two possible messages among the many that are encoded within the client-agent and issued by agents once discourse has begun. The reason for this is that the tell and ask-if messages suffice for two machines undertaking a discourse. The use of discourse is the choice that was decided upon since the scenario envisioned is one where a dialog takes place between a user and a sales agent.

From the general E-BNF syntax of Figure 2.6, a syntax of the encoded messages within client-agents is produced and shown in Figure 2.7. This grammar allows for three possible performatives: the subscribe performative, the ask-if performative, and the tell performative. As the previous BNF format, it is assumed that <string> is understood.

KQML-SpeechMessage	::=	(<performative> (:<key> <value>)*)
Performative	::=	subscribe | ask-if | tell
Key	::=	sender | receiver | in-reply-to | reply-with |
		language | ontology | content
Value[N.B.]	::=	<string> | <content-key>
Content-key	::=	[(:receiver <string>) | (:perform <string>)]
		:session <string>
		:text <string>

NOTE: The * represents the compound structure repeated zero or more times

N.B. The <content-key> is only valid after the ":content" key. For all other keys, <value>=<string>

Figure 2.7: Implemented Grammar Of Clients' Messages

KQML is used as a communication language for the exchange of knowledge between a shopping stores' database (server-agent) and a user. Considering that the client and server agents communicate with one another in a discourse manner, the *subscribe, tell, and ask-if* performatives were chosen and implemented as shown in the figure above.

The following examples illustrate how each of the implemented messages is assembled.

The Subscribe Message:

```
(subscribe
:sender         A
:receiver       B
:in-reply-to    id1
:reply-with     id2
:language       KQML
:ontology       foo
:content        (
                :receiver       B
                :session        sessionID
                :perform        "newCall"
                )
)
```

For this message, the client will ask permission to join to the server's VKB. In this case ":perform" must be the value "newCall" in order to obtain a value for the sessionID. This sessionID will be used in all subsequent calls to the server. This message originates from the client agent to the server agent.

The Tell Message:

```
(tell
:sender         A
:receiver       B
:in-reply-to    id1
:reply-with     id2
:language       KQML
:ontology       foo
:content        (
                :perform        tell
                :receiver       B
                :session        sessionID
                :text           textToSpeak/recognizedAnswer
                )
)
```

All messages of this type are statements that need to be conveyed to the receiving agent. The receiver of the statement is identified in the content portion of the message. The text to be synthesized is the value in the ":text" part of the content. This message can originate from either software agent. If it is the client-agent, then the :text key will contain the recognizedAnswer to a previous query. Otherwise, the text key contains the textToSpeak.

The Ask-if Message:

```
(ask-if
:sender         A
:receiver       B
:in-reply-to    id1
:reply-with     id2
:language       KQML
:ontology       foo
:content        (
                :perform        ask-if
                :receiver       B
                :session        sessionID
                :text           theRecognizedAnswer
                )
)
```

All messages of this type are questions that need to be asked. This message originates from the server-agent.

Figure 2.8 describes the possible flow of messages that can be sent. It is noticed that there is no communication involved/allowed between the different clients. Although, in the real world, customers may like to communicate with each other. This is not possible here.

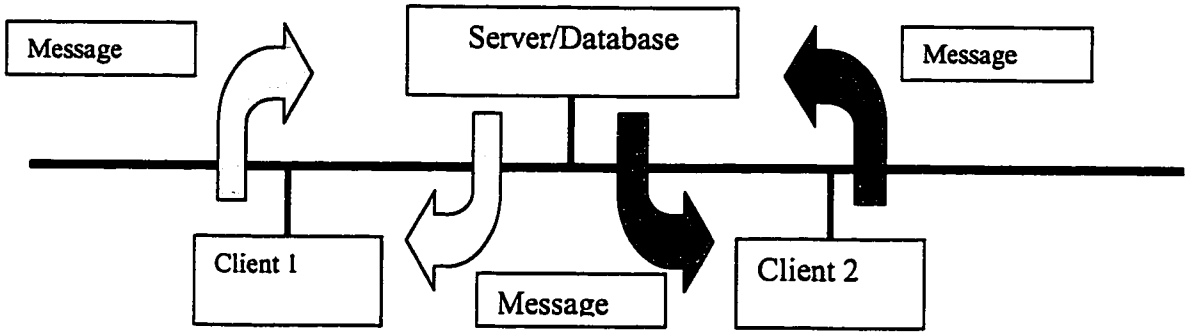


Figure 2.8: Communication Links

2.7.2 KQML Transport

This section is meant to prescribe how messages should be sent out to other KQML-compliant agents. In order to have this, the transport level provides the following abstractions[1]:

- Agents are connected by unidirectional communication links that carry discrete messages;
- These links may have a non-zero message transport delay associated with them;
- When an agent receives a message, it knows from which incoming link the message arrived;
- When an agent sends a message, it may direct to which outgoing link the message goes;
- Messages to a single destination arrive in the order they were sent;
- Message delivery is reliable

This abstraction must be considered and/or implemented in any way the agent would like. Some techniques that are often used are TCP/IP connections, inter-process calls within an ether-networked LAN, or such distributed models as Java-Remote-Method-Invocation or CORBA via some Object Request Broker software.

2.7.3 KQML Policy

It is important to note that with each agent implementation, some policies are adopted with it. For the case that was implemented (subscribe, tell, and ask-if messages), the following are assumed:

- Responsiveness: Agents will respond to messages requiring a response[12]
- Significance: Agents only send relevant messages to other agents

In different implementations of agent messages, other policies may be adopted as well. For example, networked agents will have to adopt a policy on identity in order not to have identical names of agents on the same network.

2.7.4 KQML Architecture

The architecture defines how agents are to be connected amongst each other in order to communicate. The architecture is dependent on how agents are built and perhaps the operating system that is running between the different agents (i.e. whether the OS is the same or different). The client implemented for this thesis was implemented with use of the communication medium provided by Mitel (The Tuple Space) written in Java. This medium is discussed in greater detail in section 4.2. As far as the architecture is concerned, the server and clients exchange KQML messages in this medium, and connection to it is done in any manner the agent desires.

To summarize KQML, a paper written by UMBC states that KQML can be thought of as consisting of three layers: the content layer, the message layer, and the communication layer. The value of the :content keyword is the content level, the values of the :reply-with, :sender, :receiver keywords form the communication layer and the performative name, with the :language and :ontology form the message layer. [3]

---

[12] The following messages require a response: ask-if, ask-all, ask-one, stream-all, subscribe, recommend-one, recommend-all.

Chapter 3

3 E-Commerce Project Description and Requirements

The development of virtual reality interfaces for web browsers is becoming widely popular for user applications. Virtual reality allows a user to feel immersed within the virtual scene [17] while fully experiencing the feelings involved within the environment. The application of virtual reality are very diverse. It can be used for such things as immersive video games, or collaboration work (such as university teaching, medical organ reviews, etc.), or simulations of applications that would be otherwise expensive or risky to implement.

MCRLab[13] (Multimedia Communications Research Laboratory) is developing an electronic commerce (e-commerce) application[14] where users can immerse themselves inside a virtual shopping mall and converse and trade with virtual sales agents. This virtual environment is a multi-agent (represented by avatars) system. Users would be allowed to collaborate with each other in order to trade within the virtual mall. Additionally, the agents can invoke an intelligent search of an item based, for example, on parameters such as price, colour, texture, etc. Collaboration among different users in the e-commerce world can be achieved using the RTI protocol as a means for Distributed Virtual Environment communications.

The aim of this thesis is to use virtual environment technology along with avatars, and provide users with synchronized avatar lip movements with coupled voice while ensuring

---

[13] The MCRLab is a laboratory in the University of Ottawa located on the web site http://www.mcrlab.uottawa.ca

[14] Project 98-6-4: Distributed Virtual Environment User Interface and Intelligent Agents. URL at http://www.mcrlab.uottawa.ca/research/CITR_98.htm

adequate Quality of Service (QoS). Therefore, voice recognition capability is added in the e-commerce shop application.

3.1 Main Features of This E-Commerce Project

The objectives of this research is the design of a user interface for electronic commerce application through a Distributed Virtual Environment permitting a user to navigate a virtual shopping mall, seek, retrieve and purchase items of interest, by launching intelligent software agents, and communicate/collaborate in real-time with a sales person, should the need arise [25]. The development of this project is end-user oriented. That is, users will need minimal software installations. As well, taking advantage that most people have installed a web browser on their machines, the e-commerce team tries to confine user software requirements to simply having a web browser, a VRML plug-in, and a communication mechanism that will enable the distributed aspect of the project. This communication mechanism is based on the HLA/RTI standard.

The HTML file is consisted of an applet for the electronic commerce shop, as well as an embedded VRML world that will be loaded by a web browser. The format of the HTML file will be similar to the one presented in Figure 3.1.

```
<HTML>
<EMBED SRC="EcommerceWorld.wrl" border=0 height="220" width="500">
<APPLET CODE="EcommerceApplet.class" mayscript width=500 height=280>
</APPLET>
</HTML>
```

Figure 3.1: Example HTML File Format Of The Client

As can be seen by Figure 3.1, the applet that is contained in the HTML file as well as the VRML file specified, will be loaded in the web browser and the shopper (represented by an avatar/client-agent) will then be inside a virtual mall navigating and strolling along inside the mall. The user can find many stores and each store has a sales clerk representative that can interact with the user. This sales clerk, also called a server-agent, is linked to a database

that stores a compilation of products available for purchase. The interaction of the agents will be discussed in Chapter 5.

Additional to the web browser, each client has another GUI, called the Speech GUI hereon in, where voice communication with the user is achieved. Once a command is recognized to display an object within the store, a message from the Speech GUI to the web browser is sent. At this point, the web browser must be able to handle the message and display the correct object in the virtual mall. This is accomplished through the Java applet that contains the virtual world. Then, all VRML aspects discussed previously are valid. The user can manipulate and inspect the object to decide whether or not to purchase the object.

The electronic commerce shop was developed by a member of the e-commerce team, while this thesis implemented the Speech GUI. The Speech GUI is connected through two communication links. One for local communication between the Speech GUI and the web browser, and one for client-agent to server-agent communication. Local communication is accomplished with the use of the HLA/RTI standard. The implementation needed for this is discussed in a subsequent chapter. The medium used to connect two agents together is Mitel Corporation's[15] "*Tuple Space*" written in Java, that allows agents to do database operations. Messages in the Tuple Space are assembled together using the KQML protocol.

The visualization of the agent communication only requires three KQML performatives. This is due to the fact that agents will have discourse language exchange with each other. How does one justify the selection between these performatives? To begin with, a necessary performative to be implemented is the "subscribe" performative. Any user entering the shopping store must register with the clerk to attain services. It is the user/client who initiates the dialog. At this point, the user is allowed to interact with the store's software agent. As well, the "ask-if" message is required for query purposes. This performative allows the database to ask a question that a client must respond to. Such a scenario would be to have the database ask a question, such as: "Would you like to see the dolls we have on

[15] Mitel Corporation is a Kanata-based company.

sale today?", and the user would answer "yes", "no" or other similar words accepted by the grammar. Finally, the "tell" performative must be included since the database might want to give a directive to the user such as: "Welcome to the Barbie Store".

3.2 Other E-Commerce Projects

Work on "*Usability Assessment of Collaborative Shared-Space Telepresence Shopping Services*" [18] was accomplished by implementing a virtual shopping mall ("*Teleshoppe*") where users can collaborate within the setting. The idea behind this is that users can put anything they wish in a basket and all users would be able to see the changes in the basket. The users had budget limitations imposed on them. Therefore, communication was required among the users to know what to purchase. In addition, users had the capability to talk with an assistant, the Customer Service Assistant, through a video display channel. Each of the *Teleshoppe* clients need to have the Vnet software[16] to implement the shared-space aspect of the service and the iVisit software[17] to implement both the audio communication element and the video link for the Customer Service Assistant [18].

Companies have developed systems to conduct business via the internet. For example, IBM developed the *IBM Net.Commerce* [26] and *IBM Net.Payment* [27]. These commercial products are designed for specific needs, with a focus of providing customers with purchasing power. Of course, these commercial products will be more complex than their educational counterparts.

It is clear that for the educational sector, the better solution is the one implemented by MCRLab since it requires minimal installations, available from public software sites with no necessary purchase, such as Microsoft's speech engine. It also requires users to have a web browser to be able to execute the applet that controls the objects in the virtual environment, a VRML plug-in to interpret the virtual environment, and a communication medium to pass messages in the collaborative world.

---

[16] http://ariadne.iz.net/~jeffs/vnet/

[17] http://www.ivisit.com

3.3 System Requirements

In order to properly communicate and navigate in a virtual environment, clients must have some basic elements installed on their machines. Each user must have the Microsoft Speech Engine and a soundcard with speakers and microphone installed on their system. In addition one machine must be capable of executing Mitel's Tuple Space Java application. This program will allow clients to connect to sales agents through it. Therefore, this machine needs to have an installation of the Java program. Without it, the speech clients would not be able to post and exchange KQML messages to converse with the sales agents.

Finally, a link between the virtual mall and the Speech GUI must be added. That is, make a connection between the Speech GUI and the web browser. At this point, the client is ready to have a dialog conversation with a sales agent and reflect the changes to the user within the virtual mall.

3.4 Integration Requirements of the Overall Project

The link at the client side uses the HLA/RTI as a means of communication between the Speech GUI and the virtual mall. This will allow the Speech GUI to communicate with the virtual world that should be running inside a web browser. Research on RTI as a means of communication is continually growing in the MCRLab.

Therefore, in addition to the requirements presented in section 3.3, the following components are required to be installed by a client:

- An RTI installation program; and
- A Web browser; and
- A VRML plug-in for the web browser

With all components installed, the client is ready to start the communication with a sales clerk (server-agent) and display the required objects to the user. For this, the sales clerk must communicate with the Tuple space where all message exchanges are posted. The sales

agent can connect to this Tuple space via any means, and the communication dialog can begin between the two parties.

Chapter 4

4 System Architecture

This chapter's main focus is to introduce an abstract model of the implemented Speech GUI's architecture. The diagrams will describe, in a high level overview, the different components running within the e-commerce application. Since inter-agent communication is based on a discourse language using KQML messages, sample message formations are also presented in this chapter. The client Speech GUI was tested with these types of messages and expects this kind of format. As long as the other agents form messages in the same way, inter-agent communication will not be a problem.

4.1 Designing The Server

The DVE e-commerce project is a joint project co-ordinated between many different people. A server will exist which is linked to a database and written using the KQML protocol. This server will be an expert system with knowledge of its user; by having a stored profile on them; being familiar with the user's tendencies within the electronic shopping mall; as well as their habits and other traits. An expert system is a program that tries to emulate human expertise in well defined problem domains [38]. Hence, it is providing a user with the closest interaction possible to a real human being. This, in turn, allows the users of expert systems the possibility of not having real persons waiting for user queries at the server end of the communication link.

Expert systems differ in that they are heuristic, meaning rule based. If the pre-conditions (also called facts) of the rules exist, then the rule is executed. This could possibly result in post-conditions being validated in the database, and becoming a pre-condition for another

rule. In this sense, the expert system is exhaustive, executing all rules that are valid while there still are facts asserted in the database. When there no longer exists any facts, the original problem solving objective is attained. This is also known as planning. A plan describes a way to arrive at a goal state from a given initial state [39]. Artificial Intelligent (AI) languages are ideal for these kinds of problems.

The server that will be implemented will use the Java Expert System Shell (JESS) written entirely in Sun's Java language by Ernest Friedman-Hill at Sandia National Laboratories in Livermore, CA [40]. While this server is running, it will "reason" using knowledge supplied from a CLIPS (C Language Integrated Production System) file to convey the information about the products available in the store. CLIPS is an expert system shell developed by NASA. Based on this, National Research Council (NRC) Canada has developed a fuzzy version of CLIPS, called FuzzyCLIPS [41]. Each store will have declarative rules stored within a CLIPS file that is loaded from the JESS programming environment.

The intelligence stored within the agent is the set of declarative rules that are contained within the artificial intelligent program. This intelligence is conveyed to the shopper within the store by assembling the information contained within the rules into a KQML message. It is then posted in the shared agent medium, the Tuple Space – described next.

4.2 Tuple Space Use

As discussed in the background chapter, inter-agent communication is achieved using the KQML protocol. The architecture for this is developed by Mitel, called the *Tuple Space*, and is used to allow agents to have the ability to exchange messages. Agents look in this architecture for retrieving as well as posting any messages that arises during discourse.

Messages Asserted in DB	Messages Retracted from DB	Messages Read from DB

Figure 4.1: Diagram Of The Mitel Tuple Space

The design of the Tuple Space has three sections, shown in Figure 4.1 as three columns. Each section has a purpose. The left-most column, *Messages Asserted in DB*, is designated for messages that need to be posted/asserted within the database. Originating from a sales agent, these are typically queries that are being asked to a shopper. On the other hand, messages posted from the shopper in this column are answers to queries asked from a sales agent. The middle column, *Messages Retracted from DB*, contains the messages to retract/remove facts from the database. For instance, the sales agent will want to retract an answer from the database once a question is posed to a shopper. The sales agent is then blocked, waiting for an answer. In this sense, any agent posting a retract message is blocked until a matching Tuple can be asserted in the database. Finally, the right-most column, *Messages Read from DB*, is a column where agents can post messages to enquire what facts are within the database, completing all possible database operations.

Within each messages posted in this medium, the KQML message contains a key that will specify an ontology. The ontology is described as a set of specifications on how to interpret the message to be sent or received. This will put the content into perspective for the agent.

The sample scenario on the next page, showing the beginning of the dialog, when a sales agent is waiting for a shopper to enter and subscribe to the store's information. Initially, the database is empty. The first thing that the sales agent does is to put a retract message, blocking the sales agent until a customer arrives.

```
(subscribe
:sender         "any"
:receiver       IP_OF_SALES_AGENT
:in-reply-to    "any"
:reply-with     "any"
:language       "KQML"
:ontology       ONTOLOGY_NAME
:content        "any"
)
```

The sales agent is saying: "I am waiting for any sender (anybody) who wants to talk in KQML with an ontology specified, as mine, to have a discussion with".

The shopper will subscribe to this sales agent in a similar way, shown next.

```
(subscribe
:sender         IP_OF_SHOPPER
:receiver       IP_OF_SALES_AGENT
:in-reply-to    IDENTIFIER1
:reply-with     IDENTIFIER2
:language       "KQML"
:ontology       ONTOLOGY_NAME
:content        (
                :perform        "newCall"
                :receiver       IP_OF_SHOPPER
                :session        "any"
                )
)
```

This message, in turn, says: "I would like to talk to a sales agent, whose address is IP_OF_SALES_AGENT with the KQML language and ontology specified. I am specifying that this will be a newSession with the newCall and would like to get any available session number so that I can identify myself in the future".

Once the two messages are matched within the Tuple Space, they will cancel each other out, and un-block the sales agent. The sales agent will be able to continue the message exchange with the shopper using the Tuple Space as the centre for all message exchanges.

It is important to note that whenever a database operation is performed, such as asserting or retracting a message, the expert system discussed previously (JESS) is the mechanism that handles these and allows the agents to actually do the database operations.

4.3 Client Architecture

This section gives abstract models of how the client architecture is built. It shows the reader the components within a client as well as shows how connections are made through the network to other agent systems. Figure 4.2 contains the different parts within the client. We see that the client is consisting of three main components.

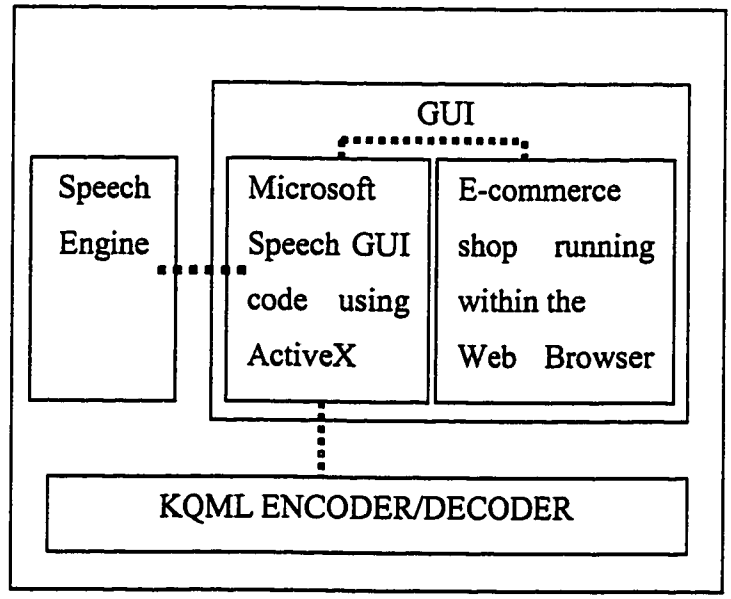


Figure 4.2: Client Architecture

The components within the client are the Microsoft Speech Engine that is downloaded freely from the Microsoft web site[18], as well as the implemented Speech GUI and e-commerce shop running within the web browser. The client must also have a KQML encoder/decoder in order to have conversation with other agents using this language.

[18] The SAPI 4.0a (Microsoft Speech API) SDK has been released February 23, 1999. Download found at http://microsoft.com/iit/

It is evident that the client above is very light-weight, requiring no software purchase for communication with other agents. The connection between the two GUI's uses the freely downloadable Runtime Infrastructure software. Nothing more is required. Connection through the internet can be made to different sales agents. This is shown in Figure 4.3. Connection can be made using the TCP/IP protocol or similar reliable protocols.

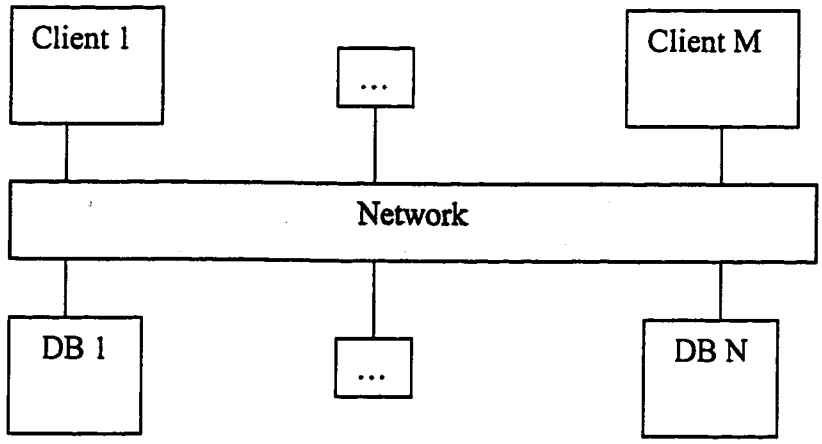


Figure 4.3: The Client-Server Connection

A client will specify which server it wants to connect to by means of specifying the name of the server as well as the ontology it wishes to conduct discourse. At this point, a session ID is given to the client from the server and communication between the agents begins.

Figure 4.4 gives a different view of this connection, where the sales agent is considered as three parts: a "Head", an "Interface", and a "Body". A similar view can be thought about the client, too. Evidently, when inside the virtual mall, users will enter stores and see representation of sales clerks. This representation is what is shown in the diagram as the "Body". The "Body" has no intelligence, it is the "Head" that is the intelligent part since it is connected to the store's database, where all information of goods is stored. The interface is just a means of communication that will be used to send information from the head to the body. An example when this is used is when such things as showing some objects in 3D are needed. A VRML representation can be sent to the scene, where shoppers can see them as well. This diagram is just another view of how the agents are.

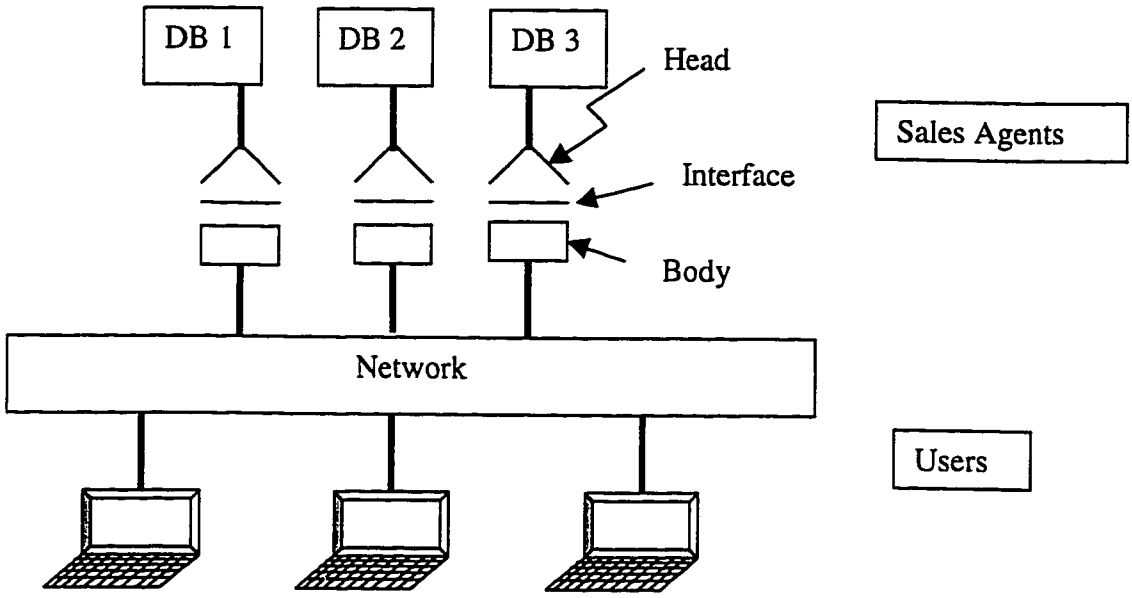


Figure 4.4: The Client-Server Connection (Another View)

On both ends of the network, avatars are useful in order to have a graphical representation of other people within the virtual mall. This feature gives users the feeling of really being inside a shopping mall. However the intelligence on both ends possesses differences and similarities. The intelligence on the client side is the human user represented within the shopping mall. The intelligence on the server side mimics human intelligence by means of a written artificial intelligent code that abides by the rules within it. This creates an atmosphere of a real shopping experience.

4.4 KQML Messages

It is important to note that dialog can occur only if the database messages match-up. Therefore, this section will describe all possible message formations that are allowed by the clients and servers. Only the messages being asserted by either the server or the client will be presented. To form a matching Tuple, one simply needs to construct the mirror images of the Tuples shown, changing only the "sender" and "receiver" fields.

4.4.1 Server Compliant Messages

There are two kinds of KQML messages that can be asserted by the server-agent. The third kind of message; "subscribe", is not shown here since it is not asserted in the database by a server, rather, it is retracted. There will be two possible server messages having the two following compositions:

```
(ask-if
:sender         IP_OF_SALES_AGENT
:receiver       IP_OF_SHOPPER
:in-reply-to    IN-REPLY-TO-ID
:reply-with     REPLY-WITH-ID
:language       "kqml"
:ontology       ONTOLOGY-NAME
:content        (
                :perform        "ask-if"
                :receiver       IP_OF_SHOPPER
                :session        SESSION-ID
                :text           QUESTION-TO-ASK
                )
)
```

Server-agents assert this kind of message when an answer is required. Typically, these messages are sent with queries, where a shopper can answer and send a response.

```
(tell
:sender         IP_OF_SALES_AGENT
:receiver       IP_OF_SHOPPER
:in-reply-to    IN-REPLY-TO-ID
:reply-with     REPLY-WITH-ID
:language       "kqml"
:ontology       ONTOLOGY-NAME
:content        (
                :perform    "tell"
                :receiver   IP_OF_SHOPPER
                :session    SESSION-ID
                :text       CONVEY-STATEMENT
                )
)
```

Server-agents assert this kind of message when a statement needs to be conveyed. Such a statement would be to say "Welcome to this Disney Toy Store". These kinds of messages can occur at any time.

4.4.2 Client Compliant Messages

There are also two kinds of KQML messages that can be asserted by the client-agent. The third kind of message, the ask-if is not an operation that is possible within the client since all queries originate from the server-agent. Client-agents can only have subscribe messages and reply to queries with tell messages. The two possible client messages will have the following compositions, shown on the next page.

```
(subscribe
:sender         IP_OF_SHOPPER
:receiver       IP_OF_SALES_AGENT
:in-reply-to    IDENTIFIER1
:reply-with     IDENTIFIER2
:language       "KQML"
:ontology       ONTOLOGY_NAME
:content        (
                :perform        "newCall"
                :receiver       MY_ADDRESS
                :session        "any"
                )
)
```

Client-agents begin discourse with a similar message assertion.

```
(tell
:sender         IP_OF_SHOPPER
:receiver       IP_OF_SALES_AGENT
:in-reply-to    IDENTIFIER1
:reply-with     IDENTIFIER2
:language       "kqml"
:ontology       ONTOLOGY-NAME
:content        (
                :receiver       IP_OF_SALES_AGENT
                :session        SESSION-ID
                :text           ANSWER-TO-QUERY
                )
)
```

The tell performative is sent from the client to the server-agent with the recognized user response to a query previously posed from the server-agent.

4.4.3 Message Sequence

Having described the different kinds of messages that can be asserted in the database, now a description of the sequencing of messages is introduced.
Figure 4.5 gives a state machine of the client-agent.

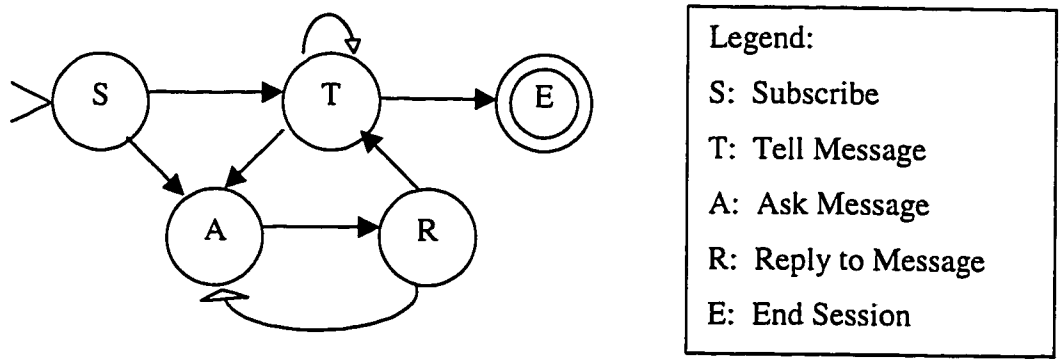


Figure 4.5: State Machine of Client-Agent

The beginning of the state machine starts off by asserting a "subscribe" message. The client can then obtain an "ask-if" message from the server or a "tell" message. Upon receiving a message, the client will generate a proper response if needed, and wait for another message. Responses for the "ask-if" and "tell" messages are either to end the current session with the server currently undertaking discourse with, or to reply to the query asked from the server agent.

The state diagram clearly shows that after receiving a tell message, while maintaining session connectivity, either another "tell" message will occur or an "ask-if" message will occur. Tell messages do not require user responses. On the other hand, the "ask-if" message forces user input. These messages are questions that need to be answered in order for the server-agent to continue the discourse. These messages can typically be such queries as "Would you like to see the items on discount today?". After a response is generated and asserted in the database, either an "ask-if" or a "tell" message will be generated from the server-agent.

It is important to note that this is how the client is implemented, user interaction must pursue throughout the session. The interaction occurs when queries need to be answered. As a result the sales agent should be written accordingly, taking into account that time is invaluable. If the sales agent asks too many questions, then a client may not have the time for it and decide to leave. On the other hand, without sufficient knowledge, a sales-agent will not know what the user is looking for. Therefore, there is a trade-off between knowledge about a user's purchasing objectives and shopper's time constraints.

Chapter 5

5 Agent Software Communication

Now that the background material has been covered, and the system architecture has been laid out, how can the KQML protocol be used by agents to have a normal discourse? After all, agents need to be able to understand messages sent to them and react accordingly to the performative and generate a response if one is required. This chapter briefly describes work of the Knowledge Sharing Effort at the University of Maryland and how they achieve software agent interoperability for agents who communicate KQML messages using different languages.

Ontologies are necessary components when using a discourse language between agents. The ontology needs to be designated with the agent ahead of time in order to give specific meanings to terminology used during discourse.

Also, the KQML architecture used (Mitel's Tuple Space) will allow us to discuss how each agent (client/server) communicates through it. Once the basis is developed we integrate the e-commerce project, using a web browser, with the Client Speech GUI and use RTI as a means for communication between them.

5.1 Knowledge-Sharing Effort Group

The Knowledge-Sharing Effort (KSE) group at the University of Maryland is an Advanced Research Projects Agency (ARPA) sponsored team whose efforts are put in finding ways to

have knowledge reuse among agents[19][7]. Since it is possible to have different agents communicate in different languages, this group tries to find ways to keep software agents interoperable with other similar agents by introducing a Knowledge Interchange Format (KIF).

The KSE has developed a specification document [2] of the KIF in order to have agents be able to interchange messages in an unknown language to one that is known. In order to do this the agent must know the KIF standard language. Knowing KIF allows languages to be changed into a native language. For example, we have an agent understanding language A, and the message received is in language B: In order to understand the message, we take it and convert it to KIF, and then convert the KIF message into language A – this message can now be understood by the agent.[20] For the purposes of this thesis, since the software agent needs to communicate with one agent (the server/database) exclusively, it is not necessary to include this feature in the implementation. It suffices that both agents will communicate using the same language and the same conventions.

Generally speaking, agent communication is not limited to a one-to-one mode of dialog. Agents can speak to any number of other agents that are visible. A picture of this communication, using KQML exchange messages, is taken from [3] and is reproduced in Figure 5.1.

[19] The KSE web site address is http://www.cs.umbc.edu/kse/

[20] In this manner, we need not know how to convert from all languages to our own. The only additional language to our own will be KIF in order to convert messages received in different languages.

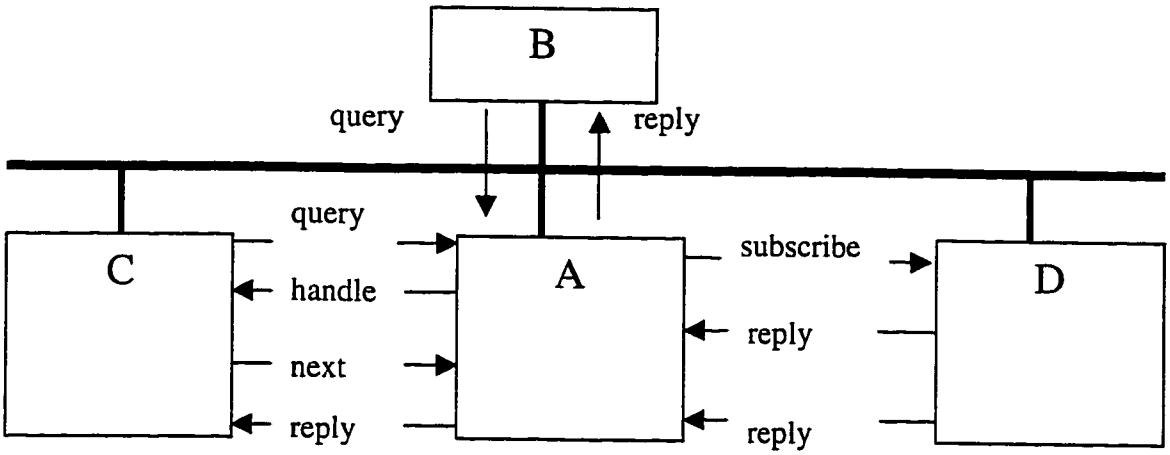


Figure 5.1: Basic Communication Protocol For Agents

5.2 Ontologies

With each message sent and received, an associated ontology is given with the message. In a short definition, ontology is described as a set of specifications on how to interpret the message sent/received.

Since agents have some conceptualization of the world (the objects and their qualities), ontology needs to be specified in order to put the received message into perspective. This will give the content of the message a meaning. For example, if the content of the message contained the word *Tire*, the meaning of this word is different depending on the ontology used. Properties are associated with each object. The ontology will give the agent the ability to associate those properties with the object. The word *Tire* in the ontology of *Toy Store* would have properties such as "made out of plastic", "diameter small", and many others too. *Tires* in the ontology of *Car Factory* would logically have different properties than the *Toy Store*. They would be "made out of rubber", "diameter big", "expensive", and so on. Although there may be many ontologies, nothing is to say that objects in two different ontologies cannot have similar properties. For example, the word *Tire* would infer a round shape, which is contained within all ontologies.

In the implemented version of the agents, a common ontology needs to be specified since only two agents will exist and cooperate to allow one shopper to do the e-commerce shopping. The ontology specified will allow the receiver of the message to understand the context of the message being spoken out. Hence, if the message was *"Show Tire"* and the ontology was *Toy Store*, the Tire with properties *"made out of plastic"* and *"diameter small"* will be shown in the virtual mall.

5.3 Server Agent Communication

Agents falling in this category form KQML messages based on what is contained within the database stored with the software-agent.

Typically, these agents base their queries to the client-agent according to pre-conditions stored in the database (DB). The answer provided by the client-agent will cause certain post-conditions to be asserted in the database. Hence, other rules will be satisfied and asked of the client-agent. Therefore, server-agents are usually based on some Artificial Intelligent (AI) language to perform database lookup. For this purpose, any AI program can be developed, and does not need to be written according to first order predicate calculus rules.

Let's take an example, presented in Figure 5.2, of a rule implemented in the CLIPS AI language. This example comes from a knowledge database to diagnose car problems. This message/rule is sent when the two preconditions (no repairs asserted in database and engine is unsatisfactory) are met.

```
(defrule determine-knocking ""
(working-state engine unsatisfactory)
(not (repair ?))
=>
(if(=(str-compare(yes-or-no-p "Does engine
knock?")?*yes*)0)
      then (assert (repair "Timing adjustment."))))
```

Figure 5.2: Example Rule In Agent Database

It is important to note that any language used to develop the database may have drawbacks. For example, Prolog has drawbacks described in *Foundations of Intelligent Knowledge-Based Systems* [4]. These include incompleteness, unfairness, unsoundness, and negations. This is because Prolog is a language that uses a depth-first rule approach, as opposed to the breadth-first or mixed version approach to find answers to queries. More information about the algorithm approach can be found in *Foundations of Intelligent Knowledge-Based Systems* [4].

Figure 5.3 demonstrates how this server-agent is connected within the architecture used in conjunction with Mitel. We must keep in mind that there is another side (the client) hooking up to this Tuple space.

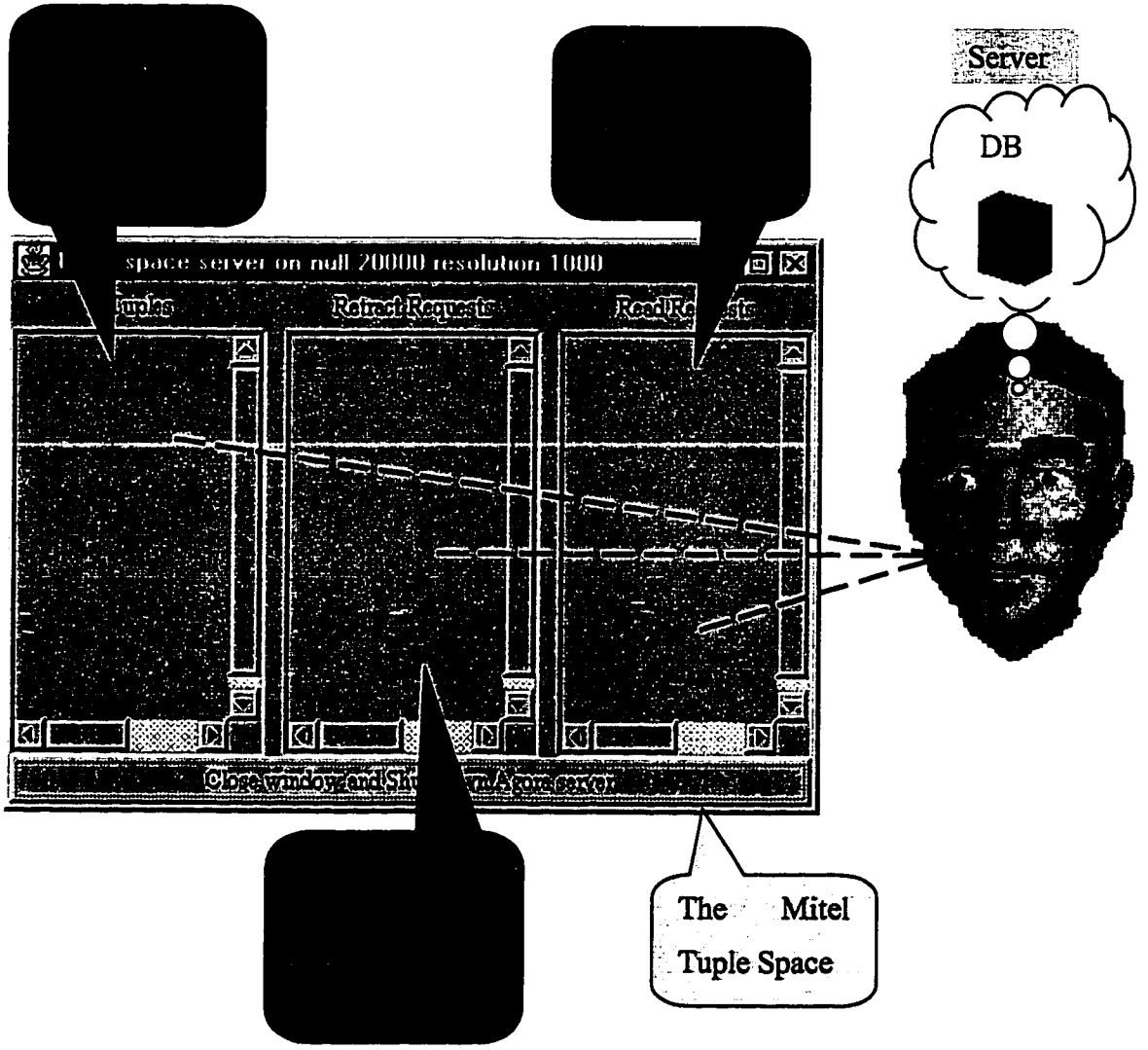


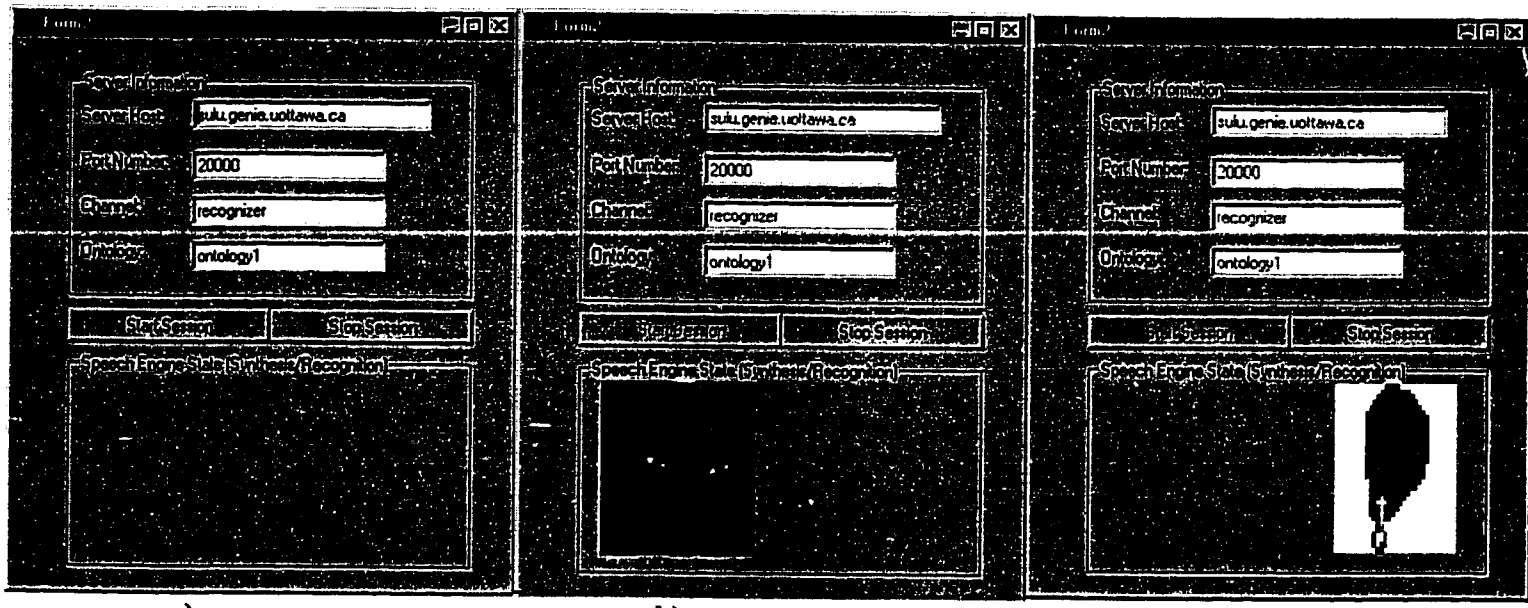
Figure 5.3: Connection of Server To Tuple Space

Clearly, Figure 5.3 demonstrates how the server-agent achieves its intelligence. The agent's brain is based on the written AI database.

5.4 Client Agent Communication

Agents falling in this category contain more sophisticated intelligence. That is, intelligence based on human thought. The agent performs operation based on the user's request. It is a totally passive agent and has no capabilities on its own. The agent is hidden behind the human shopper and does not possess any information nor does it know the environment it is in. The environment for this project is a virtual mall. However, different scenarios can be built, and these agents can be re-used in conjunction with the Speech GUI.

The user will have to complete the fields within the GUI shown in Figure 5.4. Once completed, the user will have an agent that will go and connect to a server-agent located based on the input fields entered within the GUI.



a) b) c)

Figure 5.4: a) General Dialog Box Of User For Client-Agent; b) Client-Agent during speech synthesis mode; c) Client-Agent in speech recognition mode.

We can see from these dialog windows that the client will join one single server host with some a given port number. The speech agent will discuss with the server with the ontology specified by the ontology field. Once the fields are filled out and the session started, the dialog between the server and the client begins and the user will be allowed commerce in the electronic commerce shop that the avatar is in.

Similar to the scenario presented with the server agent in Figure 5.3, the client-agent will be connected to the server through the Mitel Tuple space medium. Messages between the two agents can be exchanged in this medium. The client connection is shown in Figure 5.5.

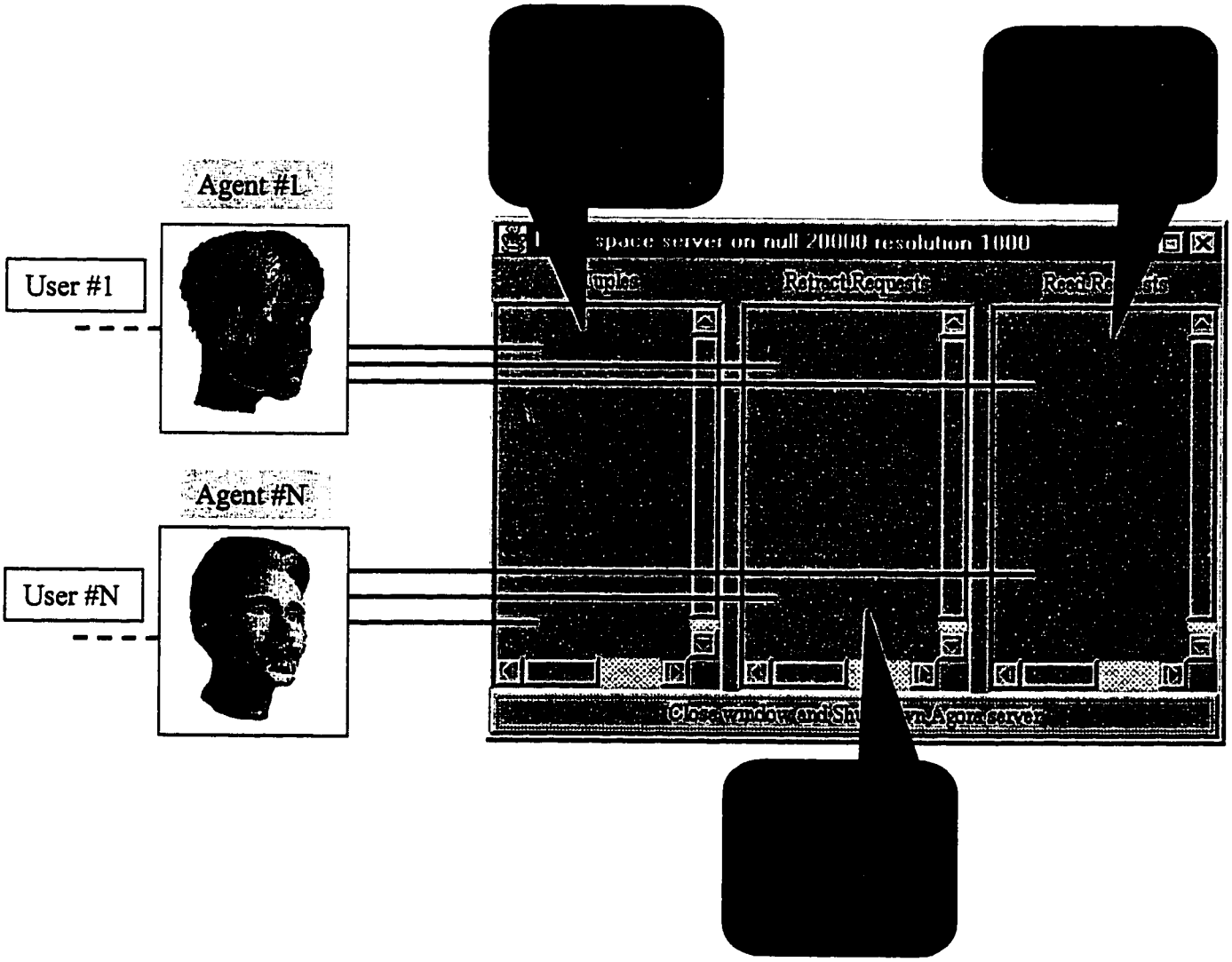


Figure 5.5: Connection of Clients To Tuple Space

The communication happening between the agents is not seen by the user. All interactions occurring between agents run independently from the users. Messages requiring a response will be synthesized at the local workstation, where the user is, and the reply will be sent through the agent and posted in the Tuple Space medium. This dialog continues until there

is no longer a query asserted in the database. At this point, the user will have found the item of interest in the shop and will be inspecting it in the virtual mall.

5.5 Connectivity Scenario Between Server & Client

Finally, when putting the whole scenery together, we can see in Figure 5.6 that the client is connected with a server and is running a virtual mall inside the web browser. It is connected to the server-agent through the Tuple space, where dialog can be processed via messages sent and received. As is shown below, no messages are found inside the Tuple Space, indicating that no dialog is in progress.

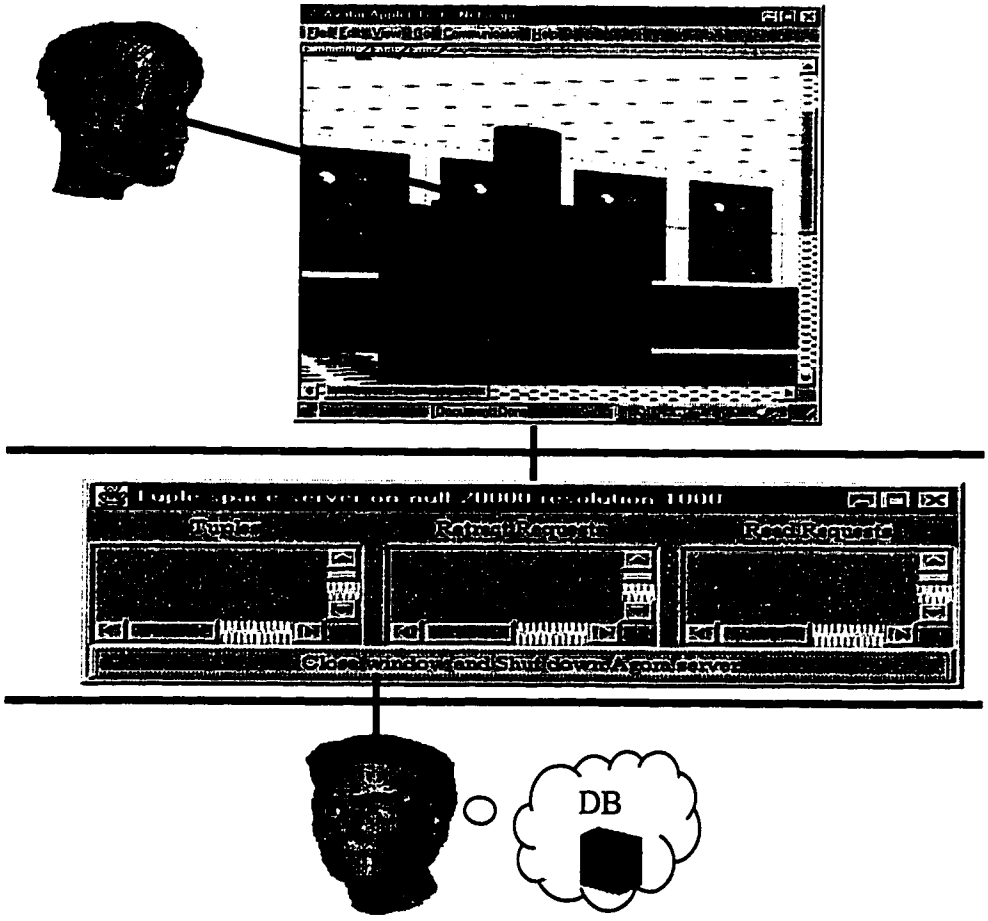


Figure 5.6: The System Interaction

5.6 Speech Agent To Browser Communication Using HLA/RTI

So far, the discussion of message exchange has been focused on how the speech client/agent communicates with the server-agent/database. This section describes how messages are sent locally between the virtual mall and the speech GUI. That is, between the two components shown in Figure 5.7, seen by a user at the client computer.

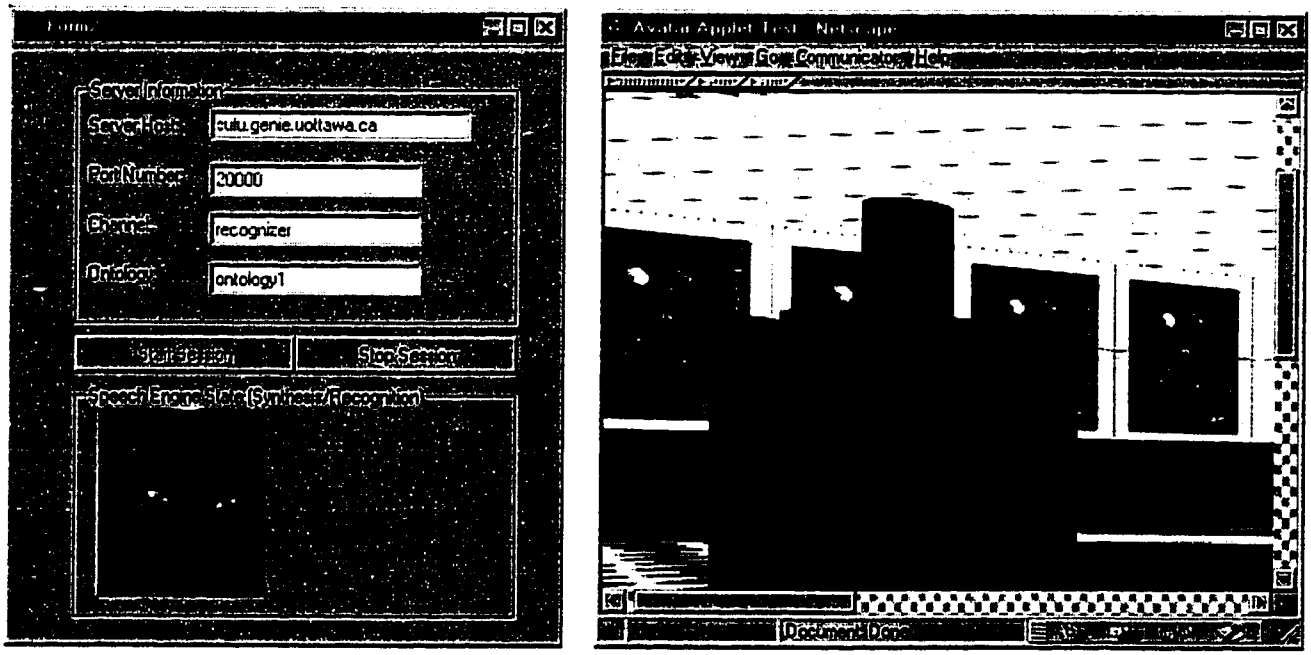


Figure 5.7: User View of Running Application

It is important to note that not all messages are sent to the web browser. For example, a kind of message that does not need to be sent to the browser is when the speech GUI synthesizes a question to the user that requires a response such as "*yes*", "*no*", or other similar words contained within the grammar. In those cases, they are locally handled without involvement of the web browser. However, messages requiring graphical display of an object (received either from agent-server or a user demand) such as "*Show the Chicago Bulls Barbie*" will have to be conveyed to the web browser. The web browser at this point will need to display the required object for the user. Since these two applications run in different Java Virtual

Machines[21], each running in a separate process, message-passing protocols do not work well. Hence, the use of the Runtime Infrastructure (RTI) of the High Level Architecture (HLA) is used.

Since RTI is used as the communication mechanism at the client side, between the Speech GUI and the web browser, a discussion of it is merited. In order to fully understand it, some definitions within the Runtime Infrastructure are needed and listed below.

Federation: A group of federates forming a community.
Federate: An entity that exchanges messages in the form of objects and interactions.
FOM: Federation Object Model.

Some of the key ingredients to implement when using RTI as a communication medium are presented and discussed below.

- Creating a new class definition
- Creating a new object of that class
- Obtaining and registering the id of the object and its attributes
- Making an RTI Event Loop to handle message during the session
- Object Subscribing/Publishing with the service
- Discussion of messages sent/received during a session
- Resignation from the Federation

First of all, since RTI is used for the communication between the speech client and the web browser applet, the information contained within the Speech GUI interface must be abstracted to a datatype. It is written and declared in the Federation Object Model. The class definition of it resembles the one produced in Figure 5.8:

[21] The Speech Graphical User Interface uses Microsoft ActiveX components and as a result the Microsoft Visual J++/Microsoft VM was used. The applet inside the web browser runs pure JDK/SUN Java code.

```
Class VoiceAgent {
protected String  mMessage;

public AgentVoice() throws RTIexception {…}
public static void PublishAndSubscribe() throws RTIexception {…}
public void UpdateMessage(String s){…}
void ResignFederationExecution(String federationName) {…}
…
}
```

Figure 5.8: Abstraction of VoiceAgent Data Type

There is one FOM per federation. It introduces all shared information (e.g. objects, interactions) and also contemplates inter-federation communication issues (e.g. data encoding schemes)[22]. The interaction between the federates inside the federation must be pre-defined in the Federation Object Model (FOM).

The next step is to create a new object via a call to the Java constructor. This will create an object with attributes (one attribute being mMessage, a String). This attribute must be linked to a federation attribute contained within a file, called Ecommerce.fed. A sample creation of this federation is shown in Figure 5.9.

[22] URL at http://www.dmso.mil/RTISUP/hla_soft/rti/rti-13r6/prog.pdf

```
(FED
      (Federation ECommerce)
      (FEDversion v1_3)
      (class Agent
            (attribute Message reliable)
      )
)
```

Figure 5.9: Federation Declaration of Objects and Attributes

Having registered the VoiceAgent attribute to the federation attribute, the RTI Event Loop will allow the VoiceAgent object to know when the attribute has changed. Any changes occurring within the attribute will be reflected and caught inside this Event Loop.

The message loop created will be similar to the one in Figure 5.10.

```
While (VoiceAgent_Joined_RTI) {
      If the object attribute (Message) has changed {
            UpdateChanges by sending to all federation members
      }
      Time Advance the Simulation
}
```

Figure 5.10: Pseudo-Code of RTI Message Loop

This message loop will allow the communication between all federates in the federation. Therefore, in order for the client-agent and the server-agent to communicate, they must reside in the same federation, called Ecommerce.

The final thing that needs discussion is how federates exchange messages with one another. Some methods are provided in the data type for this. Firstly, the PublishAndSubscribe() method defines the communication mode (i.e. the ability to receive, send, or receive and

send messages) between the agents. In the case of the Speech GUI, only a mode of sending is needed since no information is needed to come from the web browser. The method to call for sending information through the object is the UpdateMessage(String recognized) with a parameter indicated the string that was recognized.

Finally, the method ResignFederationExecution(String FederationName) is called when the RTI session is over. Typically, this is done when a user has found the item of interest and presses the "*Stop Session*" button in the Speech GUI.

As can be seen in Figure 5.11[23], when a user interaction requiring the browser to display an item occurs, the update is done locally and sent up through the layers to the federation executive, who in turn sends the event changes to other members in the federation. As a result of this, all federates update their local web browsers and collaboration can be attained.

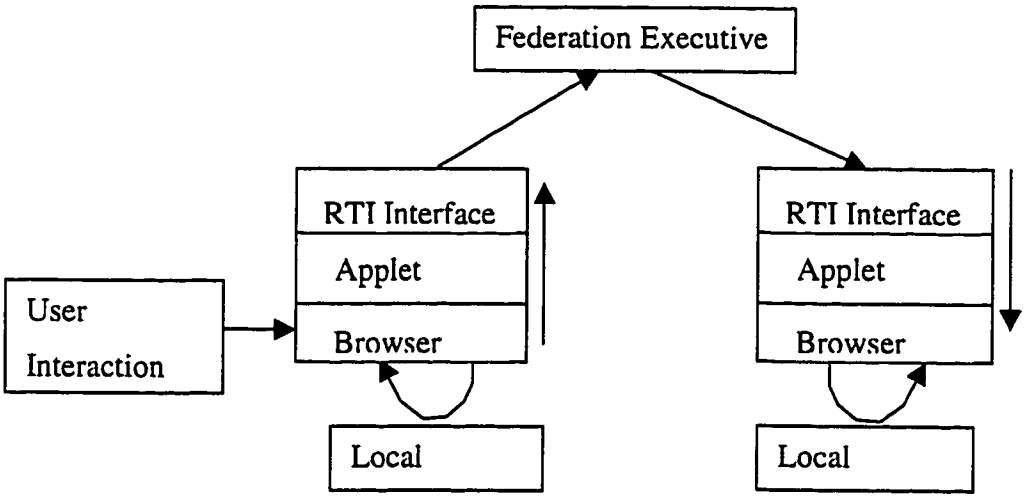


Figure 5.11: Sharing View Using RTI

There are two federations involved when collaboration is involved, the federation that links the different clients together as well as the federation which does the local communication between the Speech GUI and the web browser. The ability to do collaboration is no longer discussed since this is not the main focus of this thesis; though this discussion was warranted when discussing RTI.

[23] Taken from Xiaojun Shen's paper entitled "Collaborative Virtual Environment Over RTI in E-Commerce"

Chapter 6

6 The Microsoft Speech Engine

The Microsoft speech engine was an important tool in the development of the Speech GUI. The speech API (SAPI) provides an abstraction layer to programmers between the user's applications and the speech synthesis/recognition engines. This will allow anyone to understand what happens, at the abstraction layer, when the speech engine is synthesizing or recognizing. As well, future users of this thesis implementation can understand how to modify the grammar that is loaded once the Speech GUI starts.

6.1 Information Regarding The Application

Some background information is necessary before entering into the specifics of the developed Speech GUI. The SAPI version used to develop the Speech GUI was the SAPI 4.0. There are three SAPI documents available to describe the abstraction layer. They are: Direct COM interfaces for programmers experienced in COM programming, C++ interface for C++ developers, and ActiveX controls for developers used to programming environments where ActiveX components can be inserted.

The development environment used was a visual one, therefore the ActiveX components were used and inserted into the project. Applications using visual environments document details about object properties, methods, and events generated while the ActiveX component is active, as well as the possible error values for that control. Figure 6.1 displays the ActiveX Control API. The reader can see the *Voice Text Control* and the *Direct Speech Recognition Control* have been expanded. These are the two controls used in the implementation of the Speech GUI.

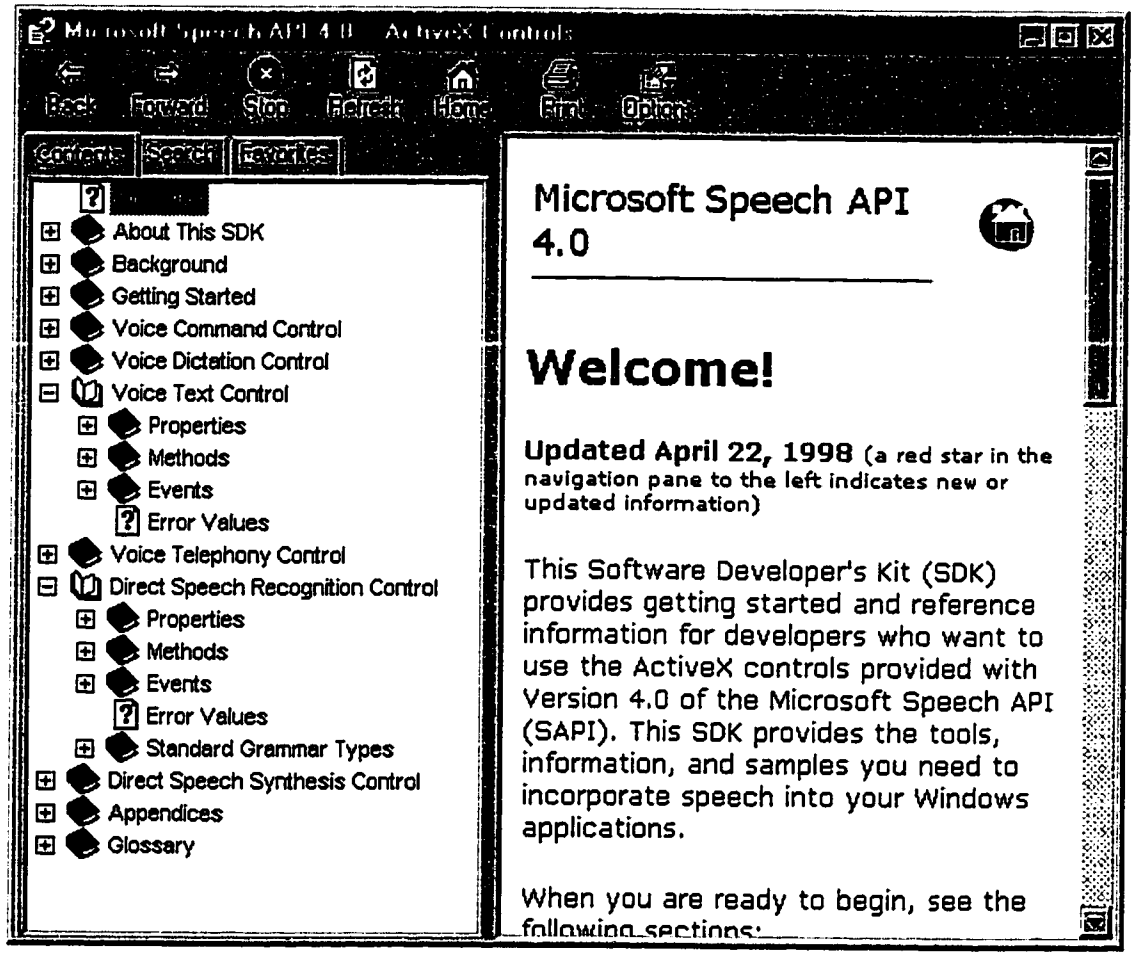


Figure 6.1: API Of ActiveX Controls

The API controls can be divided into several sections. It is apparent when viewing the Microsoft Speech API that there are two distinct sections, Voice and Direct Speech Controls. Since the Speech GUI is meant for computer applications, there is no use for the Voice Telephony Control.

The *Voice Command Control* is implemented for applications using speech recognition mainly for recognizing voice commands from a set of words found within a menu. There is no grammar loaded by this control when initialized – rather, the words to be recognized must be added during the application development within the source code. This control is ideal for programming applications that will accept commands such as "Close Window" and

the appropriate action would be issued – in this case, the application would close. This is not the requirements for the implementation of this thesis.

The *Voice Dictation Control* is used for applications that will recognize the words spoken and will display the result obtained in a sentence format (i.e. first letter of sentence would be in capital letters). The grammar associated with this control uses the engine default grammar, specific to one given language. This control provides an additional feature that other controls do not possess, due to the fact that this one would typically display the recognized speech input. The additional attribute is the *Inverse Text Normalization Grammars* feature. This provides additional grammar rules to the pre-defined default grammar to specify how sentences are formed within the language used. Typically, this control will be used when dictation of long sentences need to be generated. It is not ideal for the Speech GUI since it will recognize short answers to queries from users within the virtual shopping mall. Therefore, the only remaining possibility is the *Direct Speech Recognition Control* object.

The *Direct Speech Recognition Control* is a lower-level implementation object. Developers can use this object when the others are not suitable. To use this control, a grammar must be loaded within the object for the proper word recognition. Then, the object can be activated using the Activate method, and deactivated using the Deactivate method.

For speech synthesis, the options are between the *Voice Text Control* and the *Direct Speech Synthesis Control.* The *Direct Speech Control* is a lower-level implementation of the Voice Text Control and provides the added feature of imputing control tags within the speech being synthesized. Such control tags include the capability of adding comments within the synthesized voice, changing the speed of the synthesis, as well as many others. For the Speech GUI, it suffices to have synthesis occurring using the same engine attributes throughout the synthesis; therefore, the *Voice Text Control* is used.

6.2 Microsoft Speech Synthesis

The Microsoft Text-to-Speech engine is not a phoneme based speech engine. Rather, it is a concatenative synthesis engine. This means the audio output generated by the engine is generated from pre-recorded files, which contain sample recordings of real people. As well, the fact that synthesis is concatenative and that the range of frequencies that a telephone will reproduce (8 kHz) is much more limited than sound cards can generate (~22 kHz). The "voices" used by the text-to-speech engine are stored in different files based on both the individual speaker, and the acoustic resolution used [42]. The "style" property within the *Voice Text Control* synthesis object will therefore be associated with these different file recording frequencies. The different "style" values for the synthesis object are:

- Business
- Casual
- Computer
- Excited
- SingSong

Other object properties can be modified for synthesis. Some of the properties that can be changed are related to how the ActiveX control appears during synthesis. The modifiable properties within the *Voice Text Control* object include gender, age, dialect, lip tension, tongue position, as well as many others.

The synthesis object is created when inserting the ActiveX component within the project. Once the object "textToSpeech1" is created, synthesis can begin. This is achieved with a call to the Speak method such as textToSpeech1.Speak("Hello World"). Additionally, different events occur when the speech engine is running; for example, an event is generated when speech synthesis begins and ends. This serves as an indicator to knowing when to show and hide the mouth within the GUI. It should also be noted that during synthesis, there is lip synchronization provided, giving a nice visual effect to users, while the text is being synthesized.

6.3 Microsoft Speech Recognition

The Microsoft Speech Recognition engine is a phoneme based continuous speech recognizer [43]. As a result, the engine is speaker independent. The *Direct Speech Recognition Control* must load a grammar file in order to recognize words spoken by a user. Example grammars are listed within the Grammar File Format sub-section.

The speech recognition object, like the synthesis object, has properties, methods, and events associated with the ActiveX control. Properties associated with the speech recognition object include those of time constraints such as MaximumIncompleteTimeOut, MaximumCompleteTimeOut, as well as many others too. These properties respectively indicate to the engine how long to wait before disregarding an incomplete phrase and how long to wait before regarding a phrase as complete. As well, the recognition object has different methods associated with the ActiveX control. These include methods for loading the grammar file, activating the speech recognition engine, deactivating the speech recognition engine, as well as other useful methods.

This "recognizer1" object, like the speech synthesis object, is created when the ActiveX object is inserted inside the developed Speech GUI. Then, the object can be initialized, activated, and deactivated using the following calls:

- Recognizer1.GrammarFromFile(fileLocation)
- Recognizer1.Activate()
- Recognizer1.Deactivate()

It is important to note that for the Speech GUI implemented in this thesis, the fileLocation is the same location as the Speech GUI execution (.exe) file. The name of the file should be the name of the ontology to load with the file extension as ".txt". For instance, if the ontology being spoken is *ToyStore*, then the grammar file should be contained within a file named "*ToyStore.txt*". The grammar file should obey the file convention rules imposed by the Microsoft Speech Recognition engine.

6.3.1 Grammar File Format

A grammar file component needs to be associated with the speech recognition while the engine is active. This serves as a guide to the speech engine to know what kinds of words are expected from a user. This section will give some sample grammars that were tested, and can be used with the Microsoft SAPI 4.0. What is presented here is not an official documentation of the use of the grammar; rather, for details about the speech engine grammar file format, one can find it in the downloaded documents of the speech engine or the internet [44].

The grammar file format is divided into two different sections. The first is the grammar header portion and the second are the rules that a speech recognizer will try to understand. The header section of the grammar file can possibly contain several different parts. It would be declared as follows:

```
[Grammar]
LangID = language ID
Dialect = dialect string (optional)
Type = cfg, limited-domain, or dictation (optional)
CompileTo = cfg, limited-domain, or dictation (optional)
```

As can be seen, the grammar header involves between one to four different fields. The only field absolutely necessary is to specify a language identifier. The only one provided with the downloaded version of the SDK is the English language (the code for English is 1033). Other languages can be obtained and specified for the speech engine; however, Microsoft does not provide any additional languages with the download. The optional fields are discussed next. There is no need to discuss dialect since it is known that some languages can have different dialects. For example, English from England and English from the United States are different. The next field optionally specified is the type of grammar. The type of grammar will define what kinds of rules are possible. This will be discussed shortly. Lastly, the fourth optional component that can be specified is the "CompileTo" field. This

allows people to write rules in a preferred type and have it send the application a different format. This occurs when applications expect some kind of format from the speech engine.

Each engine type allowed has a different strategy for narrowing the set of sentences that will be recognized. The types are:

- Context-Free Grammar (CFG): This type tries to predict the next words that might possibly follow the word just spoken. This is the default type associated with the engine used, although it is always a good idea to specify a type.
- Dictation Grammar: This rule defines a context for the speaker by identifying the subject of the dictation, the expected style of language, and what dictation has already been done.
- Limited-Domain Grammar: This grammar does not provide strict syntax structures, but does provide a set of words to recognize.

The features of the different types as well as their possible compiling platform types is listed in the table on the next page for each grammar type. These are the grammar specification types discussed in the SAPI.

Type	Feature	CompileTo
CFG	Parse string Rules Command Rules Lists Default Lists (Only if CompileTo=CFG)	CFG, Limited-Domain, or Dictation.
Dictation	Topics Words Word Groups	Dictation
Limited-Domain	Topics Words Word Groups	Limited-Domain or Dictation
Frames	Frames Parse string Rules Command Rules Topics Words Word Groups	Limited Domain or Dictation

From extensive tests performed, with the downloaded speech engine, it was concluded that only CFG types were supported. An example of a CFG type grammar is shown in Figure 6.2.

```
[Grammar]
LangID=1033
Type=CFG

[<start>]
<start>=(MyRule1)

[(MyRule1)]
1=showRoom "Show " <ROOMS> "room"
2=<COLOR>

[Lists]
=ROOMS
=COLOR

[ROOMS]
=Switch
=Video

[COLOR]
=red
=green
=blue
```

Figure 6.2: Grammar File Format – Using Commands

As can be seen in Figure 6.2, the grammar uses a CFG with the English language. Some default rules provided with CFGs are listed in Appendix C. The grammar also uses most of the possible CFG features available (command rules, lists, and default lists).

In order to show all CFG features possible for the creation of rules, an example using parse string rules with the CFG type is described in Figure 6.3. Through examples, all different CFG features will have been discussed. The grammar file needed for the virtual electronic shops of the virtual mall can then be defined by the user and understood by the sales agent. For more information on creation of CFG, more details should be read and provided in Appendix D. This appendix gives techniques such as word spotting, allowing users to partially speak some of the commands that are expected.

```
[Grammar]
LangID=1033
Type=CFG

[<start>]
<start>=<Collaborative>
<start>=<NonCollaborative>

[<Collaborative>]
<Collaborative>=showRoom "Show " <ROOMS> "room"
<ROOMS>=Switch "switch "
<ROOMS>=Video "video "

[<NonCollaborative>]
<NonCollaborative>=<COLOR> [opt] <NonCollaborative>
<COLOR>=red "red "
<COLOR>=blue "blue "
<COLOR>=green "green "
```

Figure 6.3: Grammar File Format – Using Recursion

Although the engine used for this thesis only supports CFGs, it is possible to have other grammar types available with different speech engines. These SAPI compliant speech engines can have a grammar as shown in Figure 6.4. This grammar uses the a limited-domain type. Therefore, since the engine downloaded does not support limited-domain, it is not possible to use this grammar with the engine used throughout this thesis implementation.

```
[Grammar]
LangID=1033
Type=Limited-Domain

[<start>]
<start>=<Animals>
<start>=<Cities>

[WordGroups]
=Animals
=Cities

[Animals]
=cat
=dog

[Cities]
=Ottawa
=Toronto
```

Figure 6.4: Grammar File Format Using Limited-Domain Type

With the controls provided in the Speech GUI, the following engine properties are valid:

- Continuous: User can dictate words continuously
- Speaker Independent Training of the speech engine is unnecessary
- Good Environmental Noise Detection

However, SAPI documentation seems to suggest that the dictation control has the ability to have probability models associated with word groups, also called batches. This can be accomplished by using a similar construct presented on the next page.

```
typedef struct {
      QWORD       dwTotalCounts;
      DWORD       dwNumWordsClasses;
      DWORD       dwWordClassNameOffset;
      DWORD       dwNumContextGroups;
      DWORD       dwContextGroupOffset;
      DWORD       dwNumClasses;
      DWORD       dwClassOffset;
      BYTE        bBitsPerWord;
      BYTE        abFiller[3];
      DWORD       adwProbability[256];
} NGRAMHDR, * PNGRAMHDR;
```

If this feature is deemed necessary in the future, then the *Direct Speech Recognition Control* would need to be modified to reflect the needs of the application.

The potential uses of a grammar file inside of a virtual mall would be to describe a store's inventory. Including a probability model into a grammar file, a speech engine can differentiate between similar sounding words. For instance, the differences between the words "much" and "such". This feature, along with specifying an ontology will aid the speech engine. In forming the speech sentence input heard from a user, this feature along with the ontology will avoid the engine mis-recognition of words, giving proper language sentence structures.

Chapter 7

7 Conclusions and Future Enhancements

It is important to state how the evolution of computer technology brought us to the different stages of development research. Technology is evolving very quickly. Technology evolution has developed into powerful and fast computers which are connected to a tremendous and immense library, that we all know as the Internet. Today, computers are necessary tools that most people take for granted. Web browsers allow users to view vast amounts of information while being in the comfort of any environment. With the wide acceptance of the Internet, many companies have developed software programs that can be built for web browsers running different platforms. These companies created web browser plug-ins to aid the browser display items properly. For example, it would not be possible to view 3-dimensional objects without a VRML plug-in. Another example of software programs developed for use over the Internet is the Java program conceived by SUN MicroSystems Inc. Programs developed using the Java language provide advantages to developers while providing many useful features such as object-oriented programming, robustness, reliability, secureness, portability, and many more. By doing so, the Java language unified all the different heterogeneous platforms available through the web browser. Then, the web browser would not only provide vast amounts of information, but would also bring together new ideas that were previously unimaginable. Private and public companies took advantage of these benefits in order to gain customers within the marketplace while promoting the use of their software solutions. Among the many, such an example is Corel's Office For Java Suite.

Before the Internet technology appeared, agent technology was not a widely researched field. These agents did work on behalf of its user with limited resources compared to

today's agents who have more substantial means. Agents were written in a specific platform, running with limited resources. They were not completely self-driven like today's mobile agents. With introduction to the Internet, agents can now have the freedom to act on behalf of its user in distant places by way of mobility through the Internet. As well, agents written in Java may have the advantage of being able to execute informational retrieval program pieces at mobile locations; thereby, locating items of interest for its user. However, by using agent programs, one needed to implement different layers for agent operations. These were discussed in Chapter 2. They are the Agent Based Design Environment, the Internet Based Distribution mechanism, and the Agent communication Language. For the latter, use of the KQML protocol was used.

In the research of creating a distributed virtual environment for collaborative virtual shopping, the Speech GUI was developed for this thesis. This GUI provides users with agents through the graphical interface. Additionally, users will have a representation of themselves within the virtual shopping mall. This representation is called an avatar. Therefore, within the virtual shopping mall, users will have two representations; the avatar and the agent that takes care of the speech message transfers between the different agents communicating. Within the virtual environment, the Speech GUI envisioned a dialog between a sales agent and the user's agent. For this, the KQML protocol was used as the means of communication. There were three performatives chosen for this task. They were the "subscribe" message, the "ask-if" message, and the "tell" message. These performatives were chosen since a dialog discourse language was to take place between the two users. Hence, it was decided to choose one basic query performative (the ask-if performative), and one generic informational performative (the tell performative).

The agent sharing medium for messages was developed by the Kanata based Mitel Corporation, called the Tuple Space. This medium allowed all agents to connect to it, and exchange messages in this "forum". All database operations would be possible, if implemented, using this Tuple Space medium as the common medium for message sharing. The need to discuss expert systems was essential to explain how the intelligence of the server-agent is based. JESS was described as a means to create an agent expert system.

This expert system will have a program written in an artificial intelligent language connected to it, such as CLIPS. The DVE project that this thesis implementation belongs to is meant for a bigger research project involving many different people. One aspect of the project is to create a database, where agents can communicate to the user, through the Speech GUI implemented. Therefore, knowledge of a store's inventory is necessary to a server-agent trying to mimic a real person inside of the virtual mall. Artifial intelligent programs aids developers implement a server-agent describing a store's inventory.

The intelligence was based on the actual shopper. The user has a more skeletal agent than its server-agent counterpart. With support of certain KQML messages, a connection to the server-agent can be established and discourse can begin.

In order to give agents speech abilities, a decision on which speech engine needed to be decided upon. It was decided that the Microsoft Speech API would be the most logical engine to use. It provided programmers with SDKs for COM, C++, and ActiveX controls. The ActiveX controls provided lip synchronization during speech synthesis; therefore, it was decided to implement this thesis using Microsoft Visual J++ environment using two ActiveX controls. As a result of this, a separate application running alongside the web browser resulted during the virtual shopping mall environment session. Communication between the Speech GUI and the web browser was achieved using the HLA/RTI protocol.

The HLA/RTI protocol was chosen among the many standards available since the MCRLab is developing research based on these available standards. As well, the DVE team project was using this protocol as a means for message passing during the collaborative session. It is important to note that not all the speech recognition done should be distributed to the collaborative session. Rather, collaboration only allows a subset of possible recognized words to be distributed. The reason for this is that collaboration would only be necessary when changes in the virtual world are needed. Basic queries requiring answers such as "yes", "no", and other similar words should not be distributed to all the clients within the session.

Having decided on the implementation procedure, system requirements were established. When a client wishes to virtually shop, they need a web browser with a VRML plug-in installed on the local machine. The HLA/RTI implementation software allows for communication between the Speech GUI and the web browser, and the Java software provides the ability to execute the Tuple Space, if one is not already running elsewhere.

Future enhancements for the Speech GUI include the additional insertion of new ActiveX controls to better represent the avatar vis-à-vis the user. Such controls include eyes, allowing the avatar to blink from time to time. As well, the possible addition of head gestures indicating a positive or negative response can be added based on valid or invalid speech engine recognition. Furthermore, the addition of a body with controls for hands and feet can allow a more realistic view of the user representation within the virtual mall.

Due to the fact that Microsoft has developed their own Java-like software, perhaps one with the Speech GUI can be integrated within their web browser. With the current Speech GUI, the users have two representatives acting on their behalf. They are the agents who connect to the server-agents for speech discourse and the avatar within the mall for the visual effects needed. By having the Speech GUI inserted within the web browser, the two representations could be considered as one since the agent will act transparently to its user.

Additionally, the video tracking techniques allows the portrayal of a user's motion. This will allow for things like the shaking of another avatar's hand when introductions are made inside of the stores within the mall.

New KQML messages can be implemented as well. For demonstration purposes, the messages chosen for this thesis implementation sufficed to demonstrate that avatars within virtual worlds can be driven through voice commands.

Finally, with improved speech engine development, the speech recognition grammars written can be extended from context-free grammars to other types of grammars not yet supported by the engine that was used. Knowing this engine well, its full capability can be

used, giving a much better speech aware application – giving a user the best virtual shopping experience within the virtual mall.

References

[1] Specifications of the KQML Agent Communication Language. June 1993. URL at http://www.cs.umbc.edu/kqml/papers/kqmlspec.ps

[2] Genesereth Michael R. and Richard E. Fikes et. al. Knowledge Interchange Format, version 3.0 reference manual. URL at http://logic.stanford.edu/kif/Hypertext/kif-manual.html, December 1994.

[3] Finin Tim, Yannis Labrou, and James Mayfield. KQML As An Agent Communication Language. URL at http://www.cs.umbc.edu/kqml/papers/mitpress96.ps, September 1995.

[4] Torsun, I.S., Foundations of Intelligent Knowledge-Based Systems, London: Academic Press, 1995.

[5] KQML Agent-Communication Language. URL at http://www.cs.umbc.edu/kqml/kqmlspec/spec.html

[6] KQML Specifications Papers. URL at: http://www.cs.umbc.edu/kqml/papers/#spec

[7] Knowledge Sharing Effort Group. URL at http://www.cs.umbc.edu/kse

[8] High-Level Architecture. HLA Homepage, URL at http://hla.dmso.mil/hla/

[9] HLA Software Products, Run-Time Infrastructure, Programmer's Guide. URL at http://www.dmso.mil/cgi-bin/hla_dev/hla_soft.pl

[10] Shen, Xiaojun, "Collaborative Virtual Environment Over RTI in E-Commerce", 1999.

[11] SUN Microsystems Inc. Applet Security. URL at http://java.sun.com/sfaq/index.html

[12] Kindel, Charlie. ActiveX and The Web Architecture & Technical Overview. URL at http://www.microsoft.com/com/slides/kindel2.zip

[13] SUN Microsystems Inc. Java Speech Specifications. URL at http://www.javasoft.com/products/java-media/speech/forDevelopers/jsapi-doc/index.html

[14] Natural Language As An Interface Style. May 1994. URL at http://www.dgp.utoronto.ca/people/byron/papers/nli.html#Speech

[15] Larson, James A., Conversational Agents Let Users Treat Computers as People, Speech Technology Magazine. URL at http://www.speechtechmag.com/st17/convagts.htm

[16] Speech-Enabled Animated Agents. URL at http://www.fluent-speech.com/agents.htm

[17] Centre for Communication Interface Research. Virtual Worlds. URL at http://www.ccir.ed.ac.uk/applicat/worlds.html.

[18] Anderson James, Rachael Vincent, and Mervyn A. Jack. Usability Assessment of Collaborative Shared-Space Telepresence Shopping Services. Centre for Communication Interface Research.

[19] Javasoft Web Page. What is Java? URL at http://www.javasoft.com/docs/books/tutorial/getStarted/intro/definition.html

[20] Chawathe, Yatin. University of California, Berkeley. URL at http://http.cs.berkeley.edu/~yatin/cs252/javavm.html

[21] Cosmo Worlds. Plug-in Architecture Overview. February 1998. URL at http://www.cosmosoftware.com/products/worlds/win_plugin.html

[22] The Virtual Reality Modeling Language. May 1995. URL at http://www.virtpark.com/theme/vrml/

[23] VRML97 Specifications. Java External Authoring Interface. URL at http://www.vrml.org/technicalinfo/specifications/vrml97/part1/java.html#B.9.2

[24] University of California, The San Diego Supercomputer Center. VRML File Structure. URL at http://www.sdsc.edu/siggraph96vrml/course/key4.htm

[25] University of Ottawa, MCRLab. Canadian Institute For Telecommunications Research (CITR). Electronic Commerce Major Project. Project 98-6-4: Distributed Virtual Environment User Interface and Intelligent Agents. URL at http://www.mcrlab.uottawa.ca/research/CITR_98.htm

[26] IBM Corporation. IBM Net.Commerce. URL at http://www.software.ibm.com/commerce/net.commerce/

[27] IBM Corporation. IBM Net.Payment. URL at http://www.software.ibm.com/commerce/payment/

[28] JavaSoft Web Site. Applets. URL at http://www.javasoft.com/applets/index.html

[29] JavaSoft Web Site. The Java Virtual Machine Specifications. URL at http://www.javasoft.com/docs/books/vmspec/2nd-edition/html/Introduction.doc.html

[30] DML Cybrary. VRML. URL at http://www.dml.cs.ucf.edu/cybrary/fyi_vrml.html

[31] Open Inventor. URL at http://www.sgi.com/Technology/Inventor/

[32] Gorman, Trisha. Netscape World. SIGGRAPH Special Report: Web 3D graphics get boost as VRML 2.0 specification is approved. August 1996. URL at http://www.netscapeworld.com/netscapeworld/nw-08-1996/nw-08-siggraph.html.

[33] VRML Overview. August 1996. URL at http://pcgate.thch.uni-bonn.de/knut/vrml/spec/

[34] Microsoft. DirectSound: Speech in Applications. URL at http://www.windows.com/directx/pavilion/dsound/whyuse.htm

[35] Waibel, Alex and Kai-Fu Lee. *"Why Study Speech Recognition?"* in Reading in Speech Recognition. San Mateo: Morgan Kaufmann Publishers, Inc. 1990.

[36] Maes, Pattie. *"Software Agents: Humanizing The Global Computer"* in IEEE Internet Computing. Volume I, Number 4. July 1997.

[37] Aguilar, Glenn D. The World of Agents and the Evolution of Design Systems. URL at http://www.race.u-tokyo.ac.jp/~glenn/Research/Agents2/agents_paper.html

[38] GHG Internet Services. What Are Expert Systems? URL at http://www.ghgcorp.com/clips/ExpertSystems.html

[39] Torsun, I.S. Foundations of Intelligent Knowledge-Based Systems. London: Harcourt Brace & Company. 1995. (pg. 420).

[40] JESS: The Java Expert System Shell. URL at http://herzberg.ca.sandia.gov/jess/

[41] National Research Council Canada. CLIPS. URL at
http://ai.iit.nrc.ca/fuzzy/fuzzy.html

[42] Microsoft Interface Technologies. Microsoft Text-To-Speech Engine. URL at
http://microsoft.com/IIT/mstts.htm

[43] Microsoft Interface Technologies. Microsoft Speech Recognition Engine. URL at
http://microsoft.com/IIT/mscsr.htm

[44] Microsoft. Intelligent Interface Technologies. URL at:
http://microsoft.com/iit/onlinedocs/

[45] A Short History Of The Internet. An Introduction to the Internet. URL at
http://www.onshore.com/inet-presentation/inet.html

[46] Wan, Julian. Internet Boom. URL at
http://www.scotch.vic.edu.au/~scotchnc/informat/NETBOOM/Boom.htm

[47] Maes, Pattie, Robert H. Guttmand and Alexandros G. Moukas. *"Agents That Buy And Sell"* in Communications of the ACM. Volume 42, Number 3. March 1999.

Appendices

Appendix A: Summary of Reserved Parameters Keywords and Meanings

Keyword	Meaning
:content	The information about which the performative expresses an attitude
:force	Whether the sender will ever deny the meaning of the performative
:in-reply-to	The expected label in a reply
:language	The name of representation language of the :content parameter
:ontology	The name of the ontology (e.g., set of term definitions) used in the :content parameter
:receiver	The actual receiver of the performative
:reply-with	Whether the sender expects a reply, and if so, a label for the reply
:sender	The actual sender of the performative

Appendix B: KQML Performatives and their Descriptions

KQML Performatives	Description
Achieve	S wants R to make something true of their environment
Advertise	S is particularly-suited to processing a performative
Ask-about	S wants all relevant sentences in R's VKB
Ask-all	S wants all of R's answers to a question
Ask-if	S wants to know if the sentence is in R's VKB
Ask-one	S wants one of R's answers to a question
Break	S wants R to break an established pipe
Broadcast	S wants R to send a performative over all connections
Broker-all	S wants R to collect all responses to a performative
Broker-one	S wants R to get help in responding to a performative
Deny	The embedded performative does not apply to S (anymore)
Delete	S wants R to remove a ground sentence from its VKB
Delete-all	S wants R to remove all matching sentences from its VKB
Delete-one	S wants R to remove one matching sentence from its VKB
Discard	S will not want R's remaining responses to a previous performative
Eos	End of a stream of responses to an earlier query
Error	S considers R's earlier message to be mal-formed
Evaluate	S wants R to simplify the sentence
Forward	S wants R to route a performative
Generator	Same as a standby of a stream-all
Insert	S asks R to add content to its VKB

Monitor	S wants updates to R's response to a stream-all
Next	S wants R's next response to a previously-mentioned performative
Pipe	S wants R to route all further performatives to another agent
Ready	S is ready to respond to R's previously-mentioned performative
Recommend-all	S wants all names of agents who can respond to a performative
Recommend-one	S wants the name of an agent who can respond to a performative
Recruit-all	S wants R to get all suitable agents to respond to a performative
Recruit-one	S wants R to get another agent to respond to a performative
Register	S can deliver performatives to some named agent
Reply	Communicates an expected reply
Rest	S wants R's remaining responses to a previously-mentioned performative
Sorry	S cannot provide a more informative reply
Standby	S wants R to be ready to respond to a performative
Stream-about	Multiple-response version of ask-about
Stream-all	Multiple-response version of ask-all
Subscribe	S wants updates to R's response to a performative
Tell	The sentence in S's VKB
Transport-address	S associates symbolic names with transport address
Unregister	A deny of a register
Untell	The sentence is not in S's VKB

Appendix C: The Default English Language Rules Provided With SAPI 4.0

The table presented on the next page gives default rules that are provided with the SAPI for application-specific grammar use. These can be used without redefinition.

An example using some default rules, while using an application loaded grammar is given by:

```
[Grammar]
LangID=1033
Type=CFG

[<start>]
<start>=(MyRule1)

[(MyRule1)]
1=set the time to "Time=" <time>
2=set the date to "Date=" <date>
```

With this grammar listed above, one can say the following sentences:

- Set the time to eight oh six.
- Set the date to January eight nineteen ninety seven.

Rule	Default definition
<Start>	The <Start> rule, by default, is defined as: "<Start> = [opt] (JunkBegin) (Commands) [opt] (JunkEnd)". Since (JunkBegin) and (JunkEnd) are both defined by default, the application only needs to fill in the (Commands) rule.
(JunkBegin)	Filled in with a number of junk phrases the users typically speak at the beginning of sentences, such as "I want to" and "Please".
(JunkEnd)	Filled in with a number of junk phrases that users typically speak at the end of sentences, such as "please".
<PhoneNum>	This rules recognizes phone numbers.
<Year>	Recognizes years.
<Month>	Recognizes month names.
<date>	Recognizes dates.
<time>	Recognizes times.
<Dollars>	Recognizes dollars.
<Digits>	Recognizes a sequence of one or more digits.
<Digit>	Recognizes one digit.
<Fraction>	Recognizes fractions.
<Natural>	Recognizes natural numbers.
<Integer>	Recognizes integer numbers.
<Float>	Recognizes floating point numbers.
<PluralNumber>	Recognizes plural numbers, like "nineteen fifties."

Appendix D: Operators That Can Be Used Within A Rule Definition

The SAPI documentation provides several operators that can be used within rules. These are described in the SAPI along with the parse string rules. However, these operators should be allowed to be used everywhere. These are useful for such things as "word spotting". That is looking only for certain key words within a sentence.

An example grammar for word spotting:

[Grammar]
LangID=1033
Type=CFG

[<start>]

<start> = ... <SrcDst> ... <SrcDst> ...

[<SrcDst>]

<SrcDst> = "FromCity=" from (City) " "

<SrcDst> = "ToCity=" to (City) " "

This grammar allows recognition with several orders of words spoken, with wildcards put before, between, and after the recognized words looked out for. For example, sentences that are handled with this could be:

- I want to go to Vancouver leaving from Ottawa during a weekend
- I want to leave from Ottawa on the weekend for destination to Vancouver
- From Ottawa to Vancouver

- To Vancouver from Ottawa

The table below lists all operators available to developers. These are useful in many cases, and reduce the size of a grammar file format.

Operator	Description
Word	Word that must be recognized.
<Rule>	An parse string rule that must be satisfied.
<List>	A list entry.
(Rule)	A command rule that must be satisfied.
"String"	A string that's added to the parse results when PhraseParse() is called.
...	Wild-card. Any number of words (zero or more) or any type can be spoken. Applications can use this for word spotting.
[opt]	The next symbol (word or rule) is optional and doesn't have to be spoken.
[0+] or [1+]	The next symbol (word or rule) can be repeated either zero or more times [0+], or one or more times [1+].

Other important operators are the [opt], [0+], and [1+]. The opt gives users a choice to say or not to say an optional word following [opt]. An example of the use of the [opt] rule is presented on the next page. It is also important to notice that the grammar is a recursive one. That is, "MyRule1" references itself thereby allowing users to say "blue", "red", "blue red", or "red blue".

```
[Grammar]
LangID=1033
Type=CFG

[<start>]
<start>=<MyRule1>

[<MyRule1>]
<MyRule1>=<COLOR> [opt] <MyRule1>
<COLOR>=red "red "
<COLOR>=blue "blue "
```

This same definition can be redefined using the "+" operator, as shown below:

```
[Grammar]
LangID=1033
Type=CFG

[<start>]
<start>=<MyRule1>

[<MyRule1>]
<MyRule1>=[1+] <COLOR>
<COLOR>=red "red "
<COLOR>=blue "blue "
```

Operators can be useful tools when building grammars. They can significantly reduce the grammar file size.

Appendix E: Code Implemented For Speech GUI

The following pieces of code are what were developed for the Speech GUI. The code used for the server, provided by Mitel, is not produced here.

Form2.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;

import java.net.*;

/**
 * This class can take a variable number of parameters on the command
 * line. Program execution begins with the main() method. The class
 * constructor is not invoked unless an object of type 'Form2'
 * created in the main() method.
 */
public class Form2 extends Form implements Runnable
{
        mitel.electronic.SharedVariables sVariable;

        public Form2()
        {
                super();

                // Required for Visual J++ Form Designer support
                initForm();

                // TODO: Add any constructor code after initForm call
                try {
                        String cpu = java.net.InetAddress.getLocalHost().toString();
                        java.util.StringTokenizer s1 = new java.util.StringTokenizer(cpu, "/", false);
                        cpu = s1.nextToken();
                        edit1.setText(cpu);
                }
                catch (java.util.NoSuchElementException e1) { }
                catch (UnknownHostException e2) { }

                edit2.setText("20000");
```

```
        edit3.setText("recognizer");
        edit4.setText("ontology1");
        pictureBox1.hide();
        textToSpeech1.hide();
        directSR1.hide();
}

/**
 * Form2 overrides dispose so it can clean up the
 * component list.
 */
public void dispose()
{
        super.dispose();
        components.dispose();
}

private void phraseFinishMethod(Object source, xlisten.DirectSR.PhraseFinishEvent e)
{
        if (e.parsed.equals(""))
                return;

        String ee = e.parsed.toUpperCase();
        if (ee.startsWith("SHOW"))
                sVariable.rtiThread.UpdateMessage(ee);

        // Get an answer from the user to the queury and send the tell tuple to the server.
        mitel.agora.Tuple t4 = new mitel.agora.Tuple();
        mitel.agora.Tuple t5 = new mitel.agora.Tuple();
        t4.add("sender", sVariable.getClient());
        t4.add("receiver", sVariable.getServer());
        t4.add("in-reply-to", sVariable.idValue);
        t4.add("reply-wth", "any");
        t4.add("language", "kqml");
        t4.add("ontology", sVariable.getOntology());
```

```
                t4.add("content", t5);
                {
                        t5.add("receiver", sVariable.getServer());
                        t5.add("session", sVariable.getSessionID());
                        t5.add("text", e.parsed);    // put the user's answer to the server's queury.
                }
                mitel.agora.UnorderedFact f3 = new mitel.agora.UnorderedFact("tell", t4);
                (sVariable.getPort()).send(sVariable.getChannelName(), new
mitel.agora.channel.Assert(f3,2));
                (sVariable.getRecognitionObj()).Deactivate();
                (sVariable.getRecognitionObj()).setVisible(false);
                sVariable.askFinished = true;
                (sVariable.getSpeechObj()).Speak("Recognized: " + e.parsed);
        }

        private void speakingDoneMethod(Object source, Event e)
        {
                textToSpeech1.hide();
                pictureBox1.hide();
        }

        public void run() {
                mitel.electronic.commerce.Looker looker = new
mitel.electronic.commerce.Looker(sVariable);
                while (button2.getEnabled() == true) {
                        looker.handle();
                }
        }

        private void startSession_click(Object source, Event e)
        {
                System.out.println("Starting the discussion with server...");
                edit1.setEnabled(false);
                edit2.setEnabled(false);
                edit3.setEnabled(false);
```

```
            edit4.setEnabled(false);

            String server = edit1.getText();
            String port = edit2.getText();
            String channel = edit3.getText();
            String ontology = edit4.getText();
            mitel.electronic.commerce.SubscribeClass s = new
mitel.electronic.commerce.SubscribeClass(server+":"+port, channel, ontology, textToSpeech1, directSR1);
            sVariable = s.sVariable;
            sVariable.picture = pictureBox1;
            button1.setEnabled(false);

            Thread lookerThread = new Thread(this);
            lookerThread.start();
        }

        private void stopSession_click(Object source, Event e)
        {
            sVariable.rtiThread.setResignFlag();
            button2.setEnabled(false);
            this.dispose();
        }

        /**
         * NOTE: The following code is required by the Visual J++ form
         * designer.  It can be modified using the form editor.  Do not
         * modify it using the code editor.
         */
        Container components = new Container();
        GroupBox groupBox1 = new GroupBox();
        Edit edit1 = new Edit();
        Edit edit2 = new Edit();
        Edit edit3 = new Edit();
        Edit edit4 = new Edit();
        Label label1 = new Label();
```

```
Label label2 = new Label();
Label label3 = new Label();
Label label4 = new Label();
Button button1 = new Button();
xlisten.DirectSR.DirectSR directSR1 = new xlisten.DirectSR.DirectSR();
vtext.TextToSpeech.TextToSpeech textToSpeech1 = new vtext.TextToSpeech.TextToSpeech();
GroupBox groupBox2 = new GroupBox();
PictureBox pictureBox1 = new PictureBox();
Button button2 = new Button();

private void initForm()
{
        // NOTE:  This form is storing resource information in an
        // external file.  Do not modify the string parameter to any
        // resources.getObject() function call. For example, do not
        // modify "foo1_location" in the following line of code
        // even if the name of the Foo object changes:
        //   foo1.setLocation((Point)resources.getObject("foo1_location"));

        IResourceManager resources = new ResourceManager(this, "Form2");
        this.setText("Form2");
        this.setAutoScaleBaseSize(new Point(5, 13));
        this.setClientSize(new Point(355, 390));

        components.add(groupBox1, "groupBox1");
        groupBox1.setLocation(new Point(40, 24));
        groupBox1.setSize(new Point(288, 160));
        groupBox1.setTabIndex(0);
        groupBox1.setTabStop(false);
        groupBox1.setText("Server Information");

        components.add(edit1, "edit1");
        edit1.setLocation(new Point(88, 20));
        edit1.setSize(new Point(168, 20));
        edit1.setTabIndex(0);
```

```
edit1.setText("");

components.add(edit2, "edit2");
edit2.setLocation(new Point(88, 56));
edit2.setSize(new Point(136, 20));
edit2.setTabIndex(1);
edit2.setText("");

components.add(edit3, "edit3");
edit3.setLocation(new Point(88, 88));
edit3.setSize(new Point(136, 20));
edit3.setTabIndex(2);
edit3.setText("");

components.add(edit4, "edit4");
edit4.setLocation(new Point(88, 120));
edit4.setSize(new Point(136, 20));
edit4.setTabIndex(3);
edit4.setText("");

components.add(label1, "label1");
label1.setLocation(new Point(8, 24));
label1.setSize(new Point(72, 16));
label1.setTabIndex(4);
label1.setTabStop(false);
label1.setText("Server Host:");

components.add(label2, "label2");
label2.setLocation(new Point(8, 56));
label2.setSize(new Point(72, 16));
label2.setTabIndex(5);
label2.setTabStop(false);
label2.setText("Port Number:");

components.add(label3, "label3");
```

```
            label3.setLocation(new Point(8, 88));
            label3.setSize(new Point(72, 16));
            label3.setTabIndex(6);
            label3.setTabStop(false);
            label3.setText("Channel:");

            components.add(label4, "label4");
            label4.setLocation(new Point(8, 120));
            label4.setSize(new Point(64, 16));
            label4.setTabIndex(7);
            label4.setTabStop(false);
            label4.setText("Ontology:");

            components.add(button1, "button1");
            button1.setLocation(new Point(40, 192));
            button1.setSize(new Point(144, 24));
            button1.setTabIndex(1);
            button1.setText("Start Session");
            button1.addOnClick(new EventHandler(this.startSession_click));

            components.add(directSR1, "directSR1");
            directSR1.setLocation(new Point(176, 16));
            directSR1.setSize(new Point(88, 120));
            directSR1.setTabIndex(1);
            directSR1.setOcxState((AxHost.State)resources.getObject("directSR1_ocxState"));
            directSR1.setContainingForm(this);
            directSR1.addOnPhraseFinish(new
xlisten.DirectSR.PhraseFinishHandler(this.phraseFinishMethod));

            components.add(textToSpeech1, "textToSpeech1");
            textToSpeech1.setLocation(new Point(56, 96));
            textToSpeech1.setSize(new Point(40, 40));
            textToSpeech1.setTabIndex(0);

        textToSpeech1.setOcxState((AxHost.State)resources.getObject("textToSpeech1_ocxState"));
```

```
textToSpeech1.setContainingForm(this);
textToSpeech1.addOnSpeakingDone(new EventHandler(this.speakingDoneMethod));

components.add(groupBox2, "groupBox2");
groupBox2.setLocation(new Point(40, 224));
groupBox2.setSize(new Point(288, 152));
groupBox2.setTabIndex(2);
groupBox2.setTabStop(false);
groupBox2.setText("Speech Engine State (Synthesis/Recognition)");

components.add(pictureBox1, "pictureBox1");
pictureBox1.setLocation(new Point(16, 16));
pictureBox1.setSize(new Point(110, 125));
pictureBox1.setTabIndex(2);
pictureBox1.setTabStop(false);
pictureBox1.setText("pictureBox1");
pictureBox1.setImage((Bitmap)resources.getObject("pictureBox1_image"));

components.add(button2, "button2");
button2.setLocation(new Point(184, 192));
button2.setSize(new Point(144, 24));
button2.setTabIndex(3);
button2.setText("Stop Session");
button2.addOnClick(new EventHandler(this.stopSession_click));

this.setNewControls(new Control[] {
                                        button2,
                                        button1,
                                        groupBox1,
                                        groupBox2});
groupBox1.setNewControls(new Control[] {
                                                label4,
                                                label3,
                                                label2,
                                                label1,
```

```
                                                    edit4,
                                                    edit3,
                                                    edit2,
                                                    edit1});
            groupBox2.setNewControls(new Control[] {
                                                    textToSpeech1,
                                                    directSR1,
                                                    pictureBox1});

            directSR1.begin();
            textToSpeech1.begin();
        }

        /**
         * The main entry point for the application.
         *
         * @param args Array of parameters passed to the application
         * via the command line.
         */
        public static void main(String args[])
        {
            Application.run(new Form2());
        }
}
```

Looker.java

```
package mitel.electronic.commerce;

import java.util.*;
import mitel.agora.*;
import mitel.agora.ccentre.*;
import mitel.agora.channel.*;
import mitel.electronic.*;
/**
 * This type was created in VisualAge.
 */
public class Looker {
        private SharedVariables sVariable;
/**
 * This method was created in VisualAge.
 * @param shared mitel.electronic.SharedVariables
 */
public Looker(SharedVariables shared) {
        //super("Looker Thread");
        sVariable = shared;

        Port port = sVariable.getPort();
        // Create the UnorderedFact to listen for in the tuple space.
        Tuple t1 = new Tuple();
        t1.add("sender", sVariable.getServer());
        t1.add("receiver", sVariable.getClient());
        t1.add("in-reply-to", "any");
        t1.add("reply-with", "any");
        t1.add("language", "kqml");
        t1.add("ontology", sVariable.getOntology());
        t1.add("content", "any");
        UnorderedFact f1 = new UnorderedFact("tell", t1);
        UnorderedFact f2 = new UnorderedFact("ask-if", t1);
        port.send(sVariable.getChannelName(), new ReadAll(f1));
        port.send(sVariable.getChannelName(), new ReadAll(f2));
}
```

```
public void handle() {
                Vector v = sVariable.getPort().retreive(sVariable.getChannelName());
                v.trimToSize();
                for (int j = 0, k = v.capacity(); j < k; j++) {
                        v.trimToSize();
                        UnorderedFact f4 = new UnorderedFact((Tuple) v.elementAt(0));
                        (sVariable.listOfTuples).addElement(f4);
                        String perform = f4.getRelation();
                        if (perform.equals("tell")) {
                                LookerOfTells L1 = new LookerOfTells(sVariable);
                                L1.handle();
                        } else
                                if (perform.equals("ask-if")) {
                                        LookerOfAsks L2 = new LookerOfAsks(sVariable);
                                        L2.handle();
                                }
                }
}
/**
 * Returns a String that represents the value of this object.
 * @return a string representation of the receiver
 */
public String toString() {
        // Insert code to print the receiver here.
        // This implementation forwards the message to super. You may replace or supplement this.
        return super.toString();
}
}
```

LookerOfAsks.java

```
package mitel.electronic.commerce;

//import java.lang.*;
import java.util.*;
import mitel.agora.*;
import mitel.agora.channel.*;
import mitel.electronic.*;

/**
 * This type was created in VisualAge.
 */
public class LookerOfAsks {
        private String performative = new String("ask-if");
        private SharedVariables sVariable;
/**
 * LookerOfTells constructor comment.
 */
public LookerOfAsks(SharedVariables sVariable) {
        this.sVariable = sVariable;
}
public void handle() {
        sVariable.askFinished = false;
        Tuple t2 = (Tuple) sVariable.listOfTuples.elementAt(0);
        sVariable.idValue = new Long((String) t2.get("reply-with"));
        if (!(sVariable.getClient().equals((String) t2.get("receiver"))))
                return;   // This tuple is not for this client.
        Tuple t3 = (Tuple) t2.get("content");
        if (!((String) t3.get("session")).equals(sVariable.getSessionID()))
        {
                System.out.println("Wrong session number...");
                return;
        }
        String phraseToSay = (String) t3.get("text");
        System.out.println(performative + " " + phraseToSay);
```

```
        (sVariable.getRecognitionObj()).hide();
        vtext.TextToSpeech.TextToSpeech speaker = sVariable.getSpeechObj();
        (sVariable.picture).show();
        speaker.show();
        speaker.Speak(phraseToSay);
        try {
                Thread.sleep(500);
        }
        catch(InterruptedException e) {
        }
        while (speaker.getIsSpeaking()==1)
        {
                try {
                        Thread.sleep(200);
                }
                catch(InterruptedException e) {
                }
        }

        (sVariable.getRecognitionObj()).show();
        // Remove the element from the vector.
        //(sVariable.getPort()).send(sVariable.getChannelName(), new Delete(t2));
        sVariable.listOfTuples.removeElementAt(0);
        sVariable.listOfTuples.trimToSize();
        (sVariable.getRecognitionObj()).Activate();
        while (sVariable.askFinished != true) {
                try {
                        Thread.sleep(200);
                }
                catch(InterruptedException e) {
                }
        }
}
```

```
/**
 * Returns a String that represents the value of this object.
 * @return a string representation of the receiver
 */
public String toString() {
        // Insert code to print the receiver here.
        // This implementation forwards the message to super. You may replace or supplement this.
        return super.toString();
}
}
```

LookerOfTells.java

```
package mitel.electronic.commerce;

import java.util.*;
import mitel.agora.*;
import mitel.agora.channel.*;
import mitel.electronic.*;

/**
 * This type was created in VisualAge.
 */
public class LookerOfTells {
        private String performative = new String("tell");
        private SharedVariables sVariable;
/**
 * LookerOfTells constructor comment.
 * @param sVariable mitel.electronic.SharedVariables
 */
public LookerOfTells(SharedVariables sVariable) {
        this.sVariable = sVariable;
}
public void handle() {

        // Remove the element from the vector.
        Tuple t2 = (Tuple) sVariable.listOfTuples.elementAt(0);
        if (!(sVariable.getClient().equals((String) t2.get("receiver"))))
                return;   // This tuple is not for this client.
        Tuple t3 = (Tuple) t2.get("content");
        if (!((String) t3.get("session")).equals(sVariable.getSessionID()))
                return;
        String phraseToSay = (String) t3.get("text");

        if (phraseToSay.startsWith("Show")) {
                phraseToSay = phraseToSay.toUpperCase();
                sVariable.rtiThread.UpdateMessage(phraseToSay);
        }
```

```
        else {
                System.out.println("Synthesizing.."+phraseToSay);

                (sVariable.getRecognitionObj()).hide();
                vtext.TextToSpeech.TextToSpeech speaker = sVariable.getSpeechObj();
                (sVariable.picture).show();
                speaker.show();
                speaker.Speak(phraseToSay);
                try {
                        Thread.sleep(500);
                }
                catch(InterruptedException e) {
                }
                while (speaker.getIsSpeaking()==1)
                {
                        try {
                                Thread.sleep(200);
                        }
                        catch(InterruptedException e) {
                        }
                }
                sVariable.listOfTuples.removeElementAt(0);
        }
}
/**
 * Returns a String that represents the value of this object.
 * @return a string representation of the receiver
 */
public String toString() {
        // Insert code to print the receiver here.
        // This implementation forwards the message to super. You may replace or supplement this.
        return super.toString();
}
}
```

SubscribeClass.java

```
package mitel.electronic.commerce;

import java.util.*;
import java.net.*;
import mitel.agora.*;
import mitel.agora.ccentre.*;
import mitel.agora.channel.*;
import mitel.electronic.*;

/**
 * This type was created in VisualAge.
 */
public class SubscribeClass {
        private static Port port = new Port();
        private static IdGenerator idGen = new IdGenerator();
        public static SharedVariables sVariable = new SharedVariables();
/**
 * This method was created in VisualAge.
 * @param s1 java.lang.String
 * @param s2 java.lang.String
 * @param ontology java.lang.String
 */
public SubscribeClass(String s1, String s2, String ontology, vtext.TextToSpeech.TextToSpeech
textToSpeech1, xlisten.DirectSR.DirectSR directSR1) {
        sVariable.setSpeechObj(textToSpeech1);
        sVariable.setRecognitionObj(directSR1);
        sVariable.setChannelName(s2);
        try {
                sVariable.setClient(InetAddress.getLocalHost().getHostAddress());
        } catch (UnknownHostException e) {
                System.err.println("Local host unknown. Exiting program.");
                return;
        }
        sVariable.setIDGenerator(idGen);
        sVariable.setOntology(ontology);
```

```
sVariable.setPort(port);
sVariable.setServer(s1);
initSpeechEngine();
System.out.println("Waiting for server acceptance.");
// Proceed with the tuple space conversation/dialog.
(sVariable.getPort()).openChannel("tcp://" + s1, s2);

// Create and Assert UnorderedFact object (to subscribe) for tuple space.
// t1: The actual UnOrderedFact which will be asserted.
// t2: Is the content of the UnorderedFact
Tuple t1 = new Tuple();
Tuple t2 = new Tuple();
t1.add("sender", sVariable.getClient());
t1.add("receiver", sVariable.getServer());
t1.add("in-reply-to", sVariable.nextReplyID());
t1.add("reply-with", sVariable.idValue = new Long(sVariable.nextReplyID().longValue()));
t1.add("language", "kqml");
t1.add("ontology", sVariable.getOntology());
t1.add("content", t2);
        t2.add("perform", "newCall");
        t2.add("receiver", sVariable.getClient());
        t2.add("session", "any");
UnorderedFact f1 = new UnorderedFact("subscribe", t1);
(sVariable.getPort()).send(sVariable.getChannelName(), new Assert(f1));

// Now wait for the permission of the subscribe.
// t3: The actual UnOrderedFact which will be retracted.
// t4: Is the content of the UnorderedFact
Tuple t3 = new Tuple();
Tuple t4 = new Tuple();
t3.add("sender", sVariable.getServer());
t3.add("receiver", sVariable.getClient());
t3.add("in-reply-to", sVariable.idValue);
t3.add("reply-with", "any");
t3.add("language", "kqml");
```

```
        t3.add("ontology", sVariable.getOntology());
        t3.add("content", "any");
                /*
                t4.add("receiver", sVariable.getClient());
                t4.add("session", "any");
                t4.add("text", "openSession");
                */
        UnorderedFact f2 = new UnorderedFact("tell", t3);
        (sVariable.getPort()).send(sVariable.getChannelName(), new Retract(f2));
        Tuple t5 = (Tuple) (sVariable.getPort()).retreive(sVariable.getChannelName()).elementAt(0);
//      if (!(sVariable.idValue.equals((String) t5.get("in-reply-to"))))
//              return;
        Tuple t6 = (Tuple) t5.get("content");
        if (!((String) t6.get("receiver")).equals(sVariable.getClient()))
                return;
        if (!((String) t6.get("text")).equals("openSession"))
                return;
        sVariable.setSessionID((String) t6.get("session"));
}
/**
 * This method was created in VisualAge.
 */
public void initSpeechEngine() {
        // Initialize the speech engine with the file associated with the ontology given.
        String filePath = sVariable.getOntology() + ".txt";
        System.out.println(filePath);
        (sVariable.getRecognitionObj()).GrammarFromFile(filePath);
        System.out.println("Speech Engine initialized with file "+filePath);
}
}
```

MessageServers.java

```
package mitel.electronic;

import java.net.*;
import mitel.agora.*;
import mitel.agora.ccentre.*;
import mitel.agora.channel.*;
//import mitel.electronic.*;

import com.ms.com.*;
import com.ms.com.IUnknown;
import com.ms.com.Variant;

/**
 * This type was created in VisualAge.
 */
public class MessageServers {
        private static Port port = new Port();
        private static ServerSharedVariables sVariable;
        private static String SESSION = new String("0");
        private static IdGenerator idGen = new IdGenerator();
/**
*       Ask part of server.
*/
public static void addAsk(java.lang.String[] args) {
        Tuple t1 = new Tuple();
        Tuple t2 = new Tuple();
        t1.add("sender", args[0]);
        t1.add("receiver", sVariable.getClient());
        t1.add("in-reply-to", sVariable.nextReplyID());
        t1.add("reply-with", sVariable.idValue = new Long(sVariable.nextReplyID().longValue()));
        t1.add("language", "kqml");
        t1.add("ontology", sVariable.getOntology());
        t1.add("content", t2);
        {
                t2.add("perform", "ask-if");
```

```
            t2.add("receiver", sVariable.getClient());
            t2.add("session", SESSION);
            t2.add("text", "Running ask Server!!!");
}
UnorderedFact f1 = new UnorderedFact("ask-if", t1);
(sVariable.getPort()).send(sVariable.getChannelName(), new Assert(f1,2));

// Second part of the message...
Tuple t4 = new Tuple();
//Tuple t5 = new Tuple();
t4.add("sender", sVariable.getClient());
t4.add("receiver", sVariable.getServer());
t4.add("in-reply-to", sVariable.idValue);
t4.add("reply-wth", sVariable.nextReplyID());
t4.add("language", "kqml");
t4.add("ontology", sVariable.getOntology());
t4.add("content", "any");
/*
{
            t5.add("receiver", sVariable.getServer());
            t5.add("session", SESSION);
            t5.add("text", "yes");        // put the user's answer to the server's queury.
}
*/
UnorderedFact f3 = new UnorderedFact("tell", t4);
(sVariable.getPort()).send(sVariable.getChannelName(), new Retract(f3));
Tuple t6 = (Tuple) ((sVariable.getPort()).retreive(sVariable.getChannelName())).elementAt(0);
Tuple t7 = (Tuple) t6.get("content");
System.out.println("Ask tuple pair finished, answer is " + (String) t7.get("text"));

/*
int temp = Integer.parseInt(SESSION) + 1;
SESSION = String.valueOf(temp);
System.out.println("End of AskServer");
*/
```

```
}
/**
*       Tell part of server.
*/
public static void addTell(java.lang.String[] args) {
        Tuple t1 = new Tuple();
        Tuple t2 = new Tuple();
        t1.add("sender", args[0]);
        t1.add("receiver", sVariable.getClient());
        t1.add("in-reply-to", sVariable.nextReplyID());
        t1.add("reply-with", sVariable.idValue = new Long(sVariable.nextReplyID().longValue()));
        t1.add("language", "kqml");
        t1.add("ontology", sVariable.getOntology());
        t1.add("content", t2);
        {
                t2.add("perform", "tell");
                t2.add("receiver", sVariable.getClient());
                t2.add("session", SESSION);
                t2.add("text", "Running tell Server!!!");
        }

        UnorderedFact f1 = new UnorderedFact("tell", t1);
        (sVariable.getPort()).send(sVariable.getChannelName(), new Assert(f1,2));

        /*
        int temp = Integer.parseInt(SESSION) + 1;
        SESSION = String.valueOf(temp);
        System.out.println("End of Tell Server");
        */
}


/**
 * Starts the application.
 * @param args an array of command-line arguments
```

```
 */
public static void main(java.lang.String[] args) {
        port.openChannel("tcp://" + args[0], args[1]);
        sVariable = new ServerSharedVariables();
        sVariable.setChannelName(args[1]);
        sVariable.setOntology(args[2]);
        try {
                sVariable.setClient(InetAddress.getLocalHost().getHostAddress());
        } catch (UnknownHostException e) {
                System.err.println("Local host unknown. Exiting program.");
                return;
        }
        sVariable.setIDGenerator(idGen);
        sVariable.setPort(port);
        sVariable.setServer(args[0]);

        for (int i=0; i<5; i++)
        {
                // Place messages in the tuple space.
                addTell(args);
                try {
                        Thread.sleep(2000);
                } catch (InterruptedException e) {
                }
                addAsk(args);
        }
}
}
```

ServerSharedVariables.java

```
package mitel.electronic;

//import java.lang.*;
import java.util.*;
import mitel.agora.ccentre.*;
import mitel.agora.channel.*;

//import com.ms.com.*;
//import com.ms.com.IUnknown;
//import com.ms.com.Variant;

/**
 * This type was created in VisualAge.
 */
public class ServerSharedVariables{
        private Port port;
        private String client;
        private String server;
        private String channel;
        private IdGenerator idGen;
        private int sessionID;
        private String ontology;
        private vtext.TextToSpeech.TextToSpeech textToSpeech1;
        private xlisten.DirectSR.DirectSR directSR1;
        //public mitel.electronic.RTIThread rtiThread = new mitel.electronic.RTIThread();
        public com.ms.wfc.ui.PictureBox picture;
        public Vector listOfTuples = new Vector();
        public Long idValue;      // this contains the id which we are expecting the next message to arrive
with.
        public boolean askFinished;

/**
 * This method was created in VisualAge.
 * @return java.lang.String
 */
```

```
public String getChannelName() {
        return channel;
}
/**
 * This method was created in VisualAge.
 * @return java.lang.String
 */
public String getClient() {
        return client;
}
/**
 * This method was created in VisualAge.
 * @return mitel.agora.ccentre.IdGenerator
 */
public IdGenerator getIDGenerator() {
        return idGen;
}
/**
 * This method was created in VisualAge.
 * @return java.lang.String
 */
public String getOntology() {
        return ontology;
}
/**
 * This method was created in VisualAge.
 * @return mitel.agora.channel.Port
 */
public Port getPort() {
        return port;
}
/**
 * This method was created in VisualAge.
 * @return java.lang.String
 */
```

```
public String getServer() {
        return server;
}

/**
 * This method was created in VisualAge.
 * @return java.lang.String
 */
public String getSessionID() {
        Integer INT = new Integer(sessionID);
        return INT.toString();
}

/**
 * This method was created in VisualAge.
 * @return vtext.TextToSpeech.TextToSpeech
 */
public vtext.TextToSpeech.TextToSpeech getSpeechObj() {
        return textToSpeech1;
}

/**
 * This method was created in VisualAge.
 * @return xcommand.Vcommand.Vcommand
 */
public xlisten.DirectSR.DirectSR getRecognitionObj() {
        return directSR1;
}

/**
 * This method was created in VisualAge.
 * @return java.lang.Long
 */
public Long nextReplyID() {
        return (new Long(idGen.nextLocalID()));
```

```
}
/**
 * This method was created in VisualAge.
 * @return java.lang.String

public String nextSessionID() {
        sessionID++;
        return String.valueOf(sessionID);
}*/

/**
 * This method was created in VisualAge.
 * @param s java.lang.String
 */
public void setChannelName(String s) {
        channel = s;
}
/**
 * This method was created in VisualAge.
 * @param java.lang.String
 */
public void setClient(String s) {
        client=s;
}
/**
 * This method was created in VisualAge.
 * @param gen mitel.agora.ccentre.IdGenerator
 */
public void setIDGenerator(IdGenerator gen) {
        idGen = gen;
}
/**
 * This method was created in VisualAge.
 * @param s1 java.lang.String
 */
```

```
public void setOntology(String s1) {
        ontology = s1;
}
/**
 * This method was created in VisualAge.
 * @param mitel.agora.channel.Port
 */
public void setPort(Port p) {
        port = p;
}
/**
 * This method was created in VisualAge.
 * @param java.lang.String
 */
public void setServer(String s) {
        server=s;
}

/**
 * This method was created in VisualAge.
 * @param v vtext.TextToSpeech.TextToSpeech
 */
public void setSpeechObj(vtext.TextToSpeech.TextToSpeech v) {
        textToSpeech1 = v;
}

/**
 * This method was created in VisualAge.
 * @param v xcommand.Vcommand.Vcommand
 */
public void setRecognitionObj(xlisten.DirectSR.DirectSR v) {
        directSR1 = v;
}

/**
```

```
 * This method was created in VisualAge.
 * @param id java.lang.String
 */
public void setSessionID(String id) {
        Integer INT = new Integer(id);
        sessionID = INT.intValue();
}
}
```

SharedVariables.java

```
package mitel.electronic;

//import java.lang.*;
import java.util.*;
import mitel.agora.ccentre.*;
import mitel.agora.channel.*;

//import com.ms.com.*;
//import com.ms.com.IUnknown;
//import com.ms.com.Variant;

/**
 * This type was created in VisualAge.
 */
public class SharedVariables{
        private Port port;
        private String client;
        private String server;
        private String channel;
        private IdGenerator idGen;
        private int sessionID;
        private String ontology;
        private vtext.TextToSpeech.TextToSpeech textToSpeech1;
        private xlisten.DirectSR.DirectSR directSR1;
        public mitel.electronic.RTIThread rtiThread = new mitel.electronic.RTIThread();
        public com.ms.wfc.ui.PictureBox picture;
        public Vector listOfTuples = new Vector();
        public Long idValue;      // this contains the id which we are expecting the next message to arrive
with.
        public boolean askFinished;

/**
 * This method was created in VisualAge.
 * @return java.lang.String
 */
```

```
public String getChannelName() {
        return channel;
}
/**
 * This method was created in VisualAge.
 * @return java.lang.String
 */
public String getClient() {
        return client;
}
/**
 * This method was created in VisualAge.
 * @return mitel.agora.ccentre.IdGenerator
 */
public IdGenerator getIDGenerator() {
        return idGen;
}
/**
 * This method was created in VisualAge.
 * @return java.lang.String
 */
public String getOntology() {
        return ontology;
}
/**
 * This method was created in VisualAge.
 * @return mitel.agora.channel.Port
 */
public Port getPort() {
        return port;
}
/**
 * This method was created in VisualAge.
 * @return java.lang.String
 */
```

```
public String getServer() {
        return server;
}

/**
 * This method was created in VisualAge.
 * @return java.lang.String
 */
public String getSessionID() {
        Integer INT = new Integer(sessionID);
        return INT.toString();
}

/**
 * This method was created in VisualAge.
 * @return vtext.TextToSpeech.TextToSpeech
 */
public vtext.TextToSpeech.TextToSpeech getSpeechObj() {
        return textToSpeech1;
}

/**
 * This method was created in VisualAge.
 * @return xcommand.Vcommand.Vcommand
 */
public xlisten.DirectSR.DirectSR getRecognitionObj() {
        return directSR1;
}

/**
 * This method was created in VisualAge.
 * @return java.lang.Long
 */
public Long nextReplyID() {
        return (new Long(idGen.nextLocalID()));
```

```
}
/**
 * This method was created in VisualAge.
 * @return java.lang.String

public String nextSessionID() {
        sessionID++;
        return String.valueOf(sessionID);
}*/

/**
 * This method was created in VisualAge.
 * @param s java.lang.String
 */
public void setChannelName(String s) {
        channel = s;
}
/**
 * This method was created in VisualAge.
 * @param java.lang.String
 */
public void setClient(String s) {
        client=s;
}
/**
 * This method was created in VisualAge.
 * @param gen mitel.agora.ccentre.IdGenerator
 */
public void setIDGenerator(IdGenerator gen) {
        idGen = gen;
}
/**
 * This method was created in VisualAge.
 * @param s1 java.lang.String
 */
```

```
public void setOntology(String s1) {
        ontology = s1;
}
/**
 * This method was created in VisualAge.
 * @param mitel.agora.channel.Port
 */
public void setPort(Port p) {
        port = p;
}
/**
 * This method was created in VisualAge.
 * @param java.lang.String
 */
public void setServer(String s) {
        server=s;
}

/**
 * This method was created in VisualAge.
 * @param v vtext.TextToSpeech.TextToSpeech
 */
public void setSpeechObj(vtext.TextToSpeech.TextToSpeech v) {
        textToSpeech1 = v;
}

/**
 * This method was created in VisualAge.
 * @param v xcommand.Vcommand.Vcommand
 */
public void setRecognitionObj(xlisten.DirectSR.DirectSR v) {
        directSR1 = v;
}

/**
```

```
 * This method was created in VisualAge.
 * @param id java.lang.String
 */
public void setSessionID(String id) {
        Integer INT = new Integer(id);
        sessionID = INT.intValue();
}
}
package mitel.electronic;

import java.net.*;
import mitel.agora.*;
import mitel.agora.ccentre.*;
import mitel.agora.channel.*;
//import mitel.electronic.*;

/**
 * This type was created in VisualAge.
 */
public class SubscribeServer {
        private static String SESSION = new String("0");
        private static IdGenerator idGen = new IdGenerator();
/**
 * This method was created in VisualAge.
 * @param args java.lang.String[]
 */
public static void main(String args[]) {
        Port port = new Port();
        port.openChannel("tcp://" + args[0], args[1]);
        //SharedVariables sVariable = new SharedVariables();
        //sVariable.setChannelName(args[1]);
        String name;
        try {
                name = InetAddress.getLocalHost().getHostAddress();
        } catch (UnknownHostException e) {
```

```
            System.err.println("Local host unknown. Exiting program.");
            return;
    }
    //sVariable.setIDGenerator(idGen);
    //sVariable.setPort(port);
    //sVariable.setServer(args[0]);
    //sVariable.setOntology(args[2]);
    for (;;) {
            Tuple t1 = new Tuple();
            // Create t2, the actual UnOrderedFact which will be asserted.
            Tuple t2 = new Tuple();
            t2.add("sender", "any");
            t2.add("receiver", args[0]);
            t2.add("in-reply-to", "any");
            t2.add("reply-with", "any");
            t2.add("language", "kqml");
            t2.add("ontology", args[2]);
            t2.add("content", "any");
            /*
            {
                    t1.add("perform", "newCall");
                    t1.add("receiver", "any");
                    t1.add("session", "any");
            }*/
            UnorderedFact f1 = new UnorderedFact("subscribe", t2);
            port.send(args[1], new Retract(f1));

            // Wait til the client puts a subscribe message.
            Tuple f2 = (Tuple) (port.retreive(args[1])).elementAt(0);
            //sVariable.idValue = new Long((String) f2.get("reply-with"));

            Tuple t3 = new Tuple();
            Tuple t4 = new Tuple();
            t3.add("sender", args[0]);
            t3.add("receiver", name);
```

```
                t3.add("in-reply-to", new Long((String) f2.get("reply-with")));
                t3.add("reply-with", "any");
                t3.add("language", "kqml");
                t3.add("ontology", args[2]);
                t3.add("content", t4);
                {
                        t4.add("receiver", name);
                        t4.add("session", SESSION);
                        t4.add("text", "openSession");
                }
                UnorderedFact f3 = new UnorderedFact("tell", t3);
                port.send(args[1], new Assert(f3));
                int temp = Integer.parseInt(SESSION) + 1;
                SESSION = String.valueOf(temp);
        }
}
```