# AFAIK: A Help System for the Intelligent Room

by

Andy Chang

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology

August 1, 2001

Author [illegible]
Department of Electrical Engineering and Computer Science
August 1, 2001

Certified by [illegible]
Dr. Howard E. Shrobe
Thesis Supervisor

Accepted by [illegible]
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

AFAIK: A Help System for the Intelligent Room
by
Andy Chang

Submitted to the Department of Electrical Engineering and Computer Science

August 1, 2001

In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

As intelligent environments become more capable, they should also provide interactive and convenient help to a user. As a step toward this vision, I have developed an agent-based help system for the MIT Artificial Intelligence Laboratory's Intelligent Room, called AFAIK. AFAIK stresses autonomy, interactivity, and the use of multiple modalities. For example, AFAIK interacts with a user through a speech interface in addition to a traditional graphical user interface. Help content is written in XML, which provides structure to the knowledge within AFAIK. As of April 2001, the Help System is an operational component of the Intelligent Room.

Thesis Supervisor: Howard E Shrobe
Title: Associate Director, MIT Artificial Intelligence Laboratory

## TABLE OF CONTENTS

1. Introduction ........ 7
  1.1. Objective: to provide an intelligent help system ........ 7
  1.2. AFAIK is driven by four motivations ........ 8
    1.2.1. Smart environments are complex ........ 8
    1.2.2. The knowledge about the Intelligent Room is distributed ........ 8
    1.2.3. An autonomous avatar is needed ........ 9
    1.2.4. The Intelligent Room is extending beyond a research platform ........ 9
2. Background ........ 11
  2.1. The Intelligent Room is a smart environment ........ 11
  2.2. Metaglue is a software system for programming agents ........ 13
  2.3. Previous work relating to AFAIK ........ 13
    2.3.1. A previous help system for the Intelligent Room ........ 13
    2.3.2. Intelligent tutoring systems extend help systems ........ 13
3. Design Features ........ 15
  3.1. AFAIK is interactive ........ 15
  3.2. AFAIK is multi-modal ........ 16
  3.3. AFAIK uses structured information ........ 17
  3.4. AFAIK is easy to use ........ 18
4. Overview of AFAIK ........ 20
  4.1. Help files encapsulate knowledge ........ 20
    4.1.1. Help files are written in XML ........ 20
      4.1.1.1. XML help files support standard HTML elements ........ 21
      4.1.1.2. Action elements trigger actions in the Intelligent Room ........ 21
      4.1.1.3. Help files are referenced by name ........ 22
      4.1.1.4. Help files are also referenced by keyword ........ 22
      4.1.1.5. Help files are divided into subtopics ........ 22
    4.1.2. The HelpFile Object represents a help file ........ 24
      4.1.2.1. The main methods of HelpFile ........ 24
      4.1.2.2. The HelpFileExtractor extracts information from XML ........ 24
      4.1.2.3. The XMLTranslator translates XML into HTML ........ 25
  4.2. AFAIK's main agents ........ 28
    4.2.1. The JavaHelpAgent interfaces AFAIK and JavaHelp ........ 28
      4.2.1.1. Overview of Sun's JavaHelp system ........ 28
      4.2.1.2. The main methods of JavaHelpAgent ........ 29
    4.2.2. The HelpFileManagerAgent manages the knowledge base ........ 30
      4.2.2.1. The main methods of HelpFileManagerAgent ........ 30
    4.2.3. The AFAIKAgent is the heart of AFAIK ........ 31
      4.2.3.1. The main methods of AFAIKAgent ........ 31

4.2.4. The AFAIKActionAgent orchestrates the delivery of information ... 32
4.2.4.1. The main methods of AFAIKActionAgent ... 32
4.3. AFAIK's user interface agents ... 33
4.3.1. The AFAIKSpeechAgent provides speech interaction ... 33
4.3.1.1. AFAIKSpeechAgent's grammar specifies what the user can say 33
4.3.1.2. The main methods of AFAIKSpeechAgent ... 34
4.3.2. The AFAIKNavigator provides a graphical user interface ... 35
4.3.2.1. The design of AFAIKNavigatorAgent is similar to a standard web browser ... 35
4.3.2.2. The main methods of AFAIKNavigatorAgent ... 36
4.3.3. SpeechTextOutput provides speech synthesis ... 36
5. How AFAIK works ... 38
5.1. Scenario: Bob uses the Intelligent Room to make a phone call ... 38
5.2. What happens in AFAIK during Bob's scenario ... 39
5.2.1. When the Metaglue system initializes ... 39
5.2.2. When Bob asks, "What can I do in this Room?" ... 40
5.2.3. When Bob asks, "Ok them please tell me about the phone." ... 40
5.2.4. When Bob asks, "How do I use the telephone?" ... 41
5.2.5. When Bob clicks on the link, "Make a phone call." ... 42
6. Next steps ... 43
6.1. From a help system to a tutoring system ... 43
6.2. A demonstration system to demo the Intelligent Room ... 43
6.3. A user study to verify usability ... 44
6.4. Using START to access help files ... 44
6.5. Learning new information ... 44
7. Contributions ... 46
8. Appendix ... 47
8.1. Appendix A: A Sample XML Help File ... 47
8.2. Appendix B: A Sample XSL Document ... 48
8.3. Appendix C: A JSGF Grammar File ... 52
9. Bibliography ... 54

## LIST OF FIGURES

| Number | Page |
|---|---|
| 3.1.a | 12 |
| 3.1.b | 12 |
| 3.1.c | 12 |
| 3.1.d | 12 |
| 3.1.e | 12 |
| 3.1.f | 12 |
| 4.3.2.1.a | 35 |
| 4.3.2.1.b | 36 |

## ACKNOWLEDGMENTS

I wish to thank Howard Shrobe for his advice and guidance throughout the development of this project and during the writing of this thesis. In addition, I thank Krzysztof Gajos, Katherine Koch, and Steven Peters for supporting my research and for their friendship.

Finally, I thank my parents for making my education possible, and to Erica Peterson for her constant smiles.

*Chapter 1*

# INTRODUCTION

Imagine walking into a modern conference room for the first time. As you look around, you see a myriad of displays, cameras, microphones, and other high-tech gadgets. Being naturally curious, you wish to use and play with these resources in the room. However, there is no one around to ask for help. The people who built this environment are all out to lunch, and none of them brought their cell phones with them. On a desk in the center of the room sits a four-inch thick book entitled, "How To Operate This Environment." Aghast by the daunting size of this user's manual, you cry out, "Please help me!" Suddenly, the room reacts to your distress call, and responds by speaking to you. Surprised by this voice, you reply by asking, "What can I do in this room?" A projector turns on and displays a list of applications you can use and devices you can control.

## 1.1. OBJECTIVE: TO CREATE AN INTELLIGENT HELP SYSTEM

The conference room in the above scenario is not an ordinary conference room. It is an intelligent environment at MIT's Artificial Intelligence Laboratory called the Intelligent Room. However, even though the Intelligent Room can perform many impressive tasks, until recently, it could not provide the simple help described in the above scenario. My thesis is about technology that rectifies this problem.

My objective is to create an intelligent help system for the Intelligent Room. The help system I created is called AFAIK, and it enables a person in the Intelligent Room to interactively access a knowledge base containing helpful information. The information within AFAIK is structured, and is written by the developers of the

Intelligent Room in XML. This thesis documents my research, design, and implementation of AFAIK.

In Chapter 2, I provide some background information relating to the development of AFAIK. In Chapter 3, I describe the issues relating to the design of AFAIK. In Chapter 4, I provide a detailed description of AFAIK. In Chapter 5, I illustrate how AFAIK works using a sample scenario. In Chapter 6, I present the next steps for AFAIK. Finally, in Chapter 7, I conclude with the contributions of this thesis to the Intelligent Room Project.

## 1.2. AFAIK IS DRIVEN BY FOUR MOTIVATIONS

### 1.2.1. Smart environments are complex

The Intelligent Room is a complex synergy of software agents, hardware devices, and human researchers. New features are added daily, which increases the capability and functionality of the Intelligent Room. Unfortunately, the improvements in capability and functionality come at a cost of increased complexity. At the time of writing, there are over one hundred thousand lines of software code, a dozen computers, multiple input and output devices, and over twenty researchers. For example, there are more than a half dozen ways of turning on the lights in the Intelligent Room. Many tasks are daunting even for the most seasoned developer, let alone a novice user. Therefore, there is a need to convey simple information about operating and interacting with Intelligent Room in a straightforward manner.

### 1.2.2. The knowledge about the Intelligent Room is distributed

The people who know best how to operate the Intelligent Room are the researchers who install the hardware and write the software. Unfortunately, no single researcher knows every single feature or option; knowledge is distributed over a dozen researchers. In addition, the Intelligent Room incorporates projects from other research groups. The InfoLab, Vision Interface Project, and Design Rationale research groups routinely demonstrate their projects using the Intelligent Room as a base platform. For example, Design Rationale's Assist sketch-understanding tool [Alvarado00] is integrated into the Intelligent Room, and serves as an additional input modality. Therefore, there is a need to centralize the knowledge relating to the Intelligent Room.

### 1.2.3. An autonomous avatar is needed

In addition to centralizing the knowledge, presenting the knowledge to a user needs to be done autonomously. It is infeasible to staff the Intelligent Room with a knowledgeable researcher whenever a person wishes to use the Intelligent Room. Traditionally, in order to give demonstrations of the Intelligent Room, a researcher had to be present to inform the audience how to interact with the Intelligent Room. Ideally, the audience should be able to learn how to use the Intelligent Room by themselves without the presence of a researcher. Therefore, there is a need for an autonomous avatar containing a centralized knowledge base that continuously runs in the Intelligent Room.

### 1.2.4. The Intelligent Room is extending beyond a research platform

The above motivations relating to complexity, decentralization, and autonomy are relevant to a recent initiative of the Intelligent Room project. Beginning in the spring of 2001, the technology of the Intelligent Room was released to other groups within the Artificial Intelligence Laboratory. This release initiative packaged together the Metaglue software infrastructure, numerous software agents, and various hardware technologies. The package is then distributed to the different offices within the Artificial Intelligence Laboratory. In essence, each distribution is a miniature version of the Intelligent Room. This release initiative allows other people use Intelligent Room technology and increases the visibility of the Intelligent Room project.

The people who receive the Intelligent Room technology are not associated with the Intelligent Room project itself. As a consequence, these people need to be trained and instructed on the use of the technology. An interactive help system that is packaged with the released technology should alleviate this problem. For example, when a person first receives the Intelligent Room technology, the help system can lead him through the initial setup of the technology, and introduce him to some simple features of the Intelligent Room package.

*Chapter 2*

# BACKGROUND

## 2.1. THE INTELLIGENT ROOM IS A SMART ENVIRONMENT.

The Intelligent Room project at MIT's Artificial Intelligence Laboratory is a platform for researching the interaction between humans and computers.

The Intelligent Room itself is a conference room at the Artificial Intelligent Laboratory. The Room is equipped with numerous input and output devices, which enable people to interact naturally with the Room. For example, the Intelligent Room has:

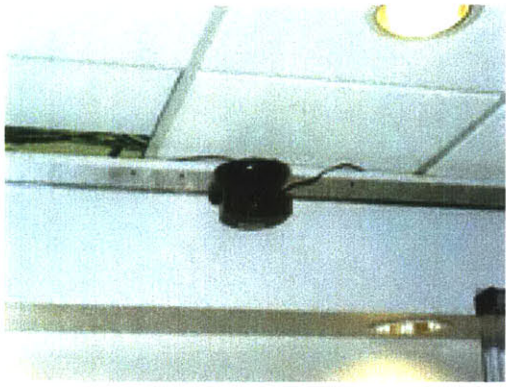
- Cameras to see people and events [3.1.a]
- Microphones to hear verbal commands and oral conversations [3.1.b]
- Speakers to output synthesized voice and play music [3.1.c]
- Projectors to display graphical information [3.1.d] [3.1.f]
- LED signs to display textual information [3.1.e]
- Various consumer technology such as VCRs, DVD players, and tuners

These devices work collaboratively with various computer vision[1], speech recognition, and natural language processing systems[2]. For example, instead of using a traditional keyboard and mouse to look up the current weather in Boston, a person can point to Boston on an interactive map projected onto the wall and say, "What is the

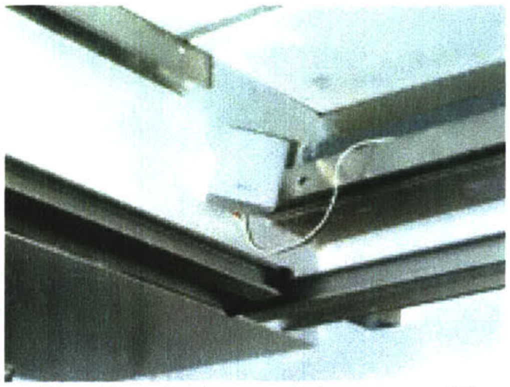
[1] Vision Interface Project, MIT Artificial Intelligence Laboratory

[2] Infolab, MIT Artificial Intelligence Laboratory

weather in Boston?" In another example, the Intelligent Room turns on the lights when a person walks into the room. When the person sits down at a chair, the Intelligent Room turns on a display that faces the person.


*2.1.a – Controllable Sony camera mounted on the ceiling of the Intelligent Room*

*2.1.b – Array of microphones in the ceiling of the Intelligent Room*


*2.1.c – Bose speakers mounted in a corner of the Intelligent Room provide sound output.*

*2.1.d – Projector mounted to the ceiling, which projects a display onto the opposing wall.*

*2.1.e – Alpha LED display, which displays textual data.*

*2.1.f – Three projectors provide a large "display wall."*

## 2.2. METAGLUE IS A SOFTWARE SYSTEM FOR PROGRAMMING AGENTS

Numerous software agents control the devices and interactions in the Intelligent Room. These agents are developed under the Metaglue Agent System. [Phillips99] The Metaglue Agent System is a Java-based programming environment for software agents, and relies on Java's RMI for inter-agent communication.

## 2.3. PREVIOUS WORK RELATING TO AFAIK

### 2.3.1. A previous help system for the Intelligent Room

The Sally [Groh99] system was a precursor help system for the Intelligent Room. Sally's approach was to start with an empty knowledge base and accumulate knowledge by learning information from users in the Intelligent Room. For example, when a user asks, "How do I use the telephone," Sally searches through its knowledge base for an answer. If Sally does not know, it will ask the user for the information and remember the user's response in its knowledge base. The next time a user asks, "How do I use the telephone," Sally's knowledge base contains the matching answer, and Sally responds with the information in its knowledge base.

However, Sally was not fully deployed in the Intelligent Room. This was partly due to the high time expense of the initial knowledge accumulation. In addition, while Sally supported multimedia data, the data within Sally was unstructured. As a result, it is difficult to browse and organize the information contained within Sally.

### 2.3.2. Intelligent tutoring systems extend help systems

While intelligent help systems such as AFAIK and Sally respond to requests for help, intelligent tutoring systems can proactively provide help to a user. This ability makes intelligent tutoring systems better suited towards the task of teaching. Tutoring systems deal with the research areas of knowledge representation, student modeling, and tutorial strategies [Byerley88]. The ability to change the information within a knowledgebase and change the information delivery strategy based on the properties of the user distinguishes intelligent tutoring systems from other forms of electronic help.

One such system is Dominie. [Byerley88] Dominie is an intelligent tutoring system designed for teaching procedural knowledge about a network switch. Dominie models the user based on three characteristics, such as the user's motivational state. Based on these characteristics of the user, Dominie employs one of six tutorial strategies, such as teaching with a top-down approach or using practice examples. [Byerley88] Another intelligent tutoring system is a system for learning in the context of air traffic control training. [Morrisroe88] This system consists of a Knowledge-Based Tutor (KBT) that monitors the user's actions and changes the operation of the system accordingly. If the KBT detects a problem, it can change the current scenario and provide assistance to the user, or remember the circumstances of the problem and present advice to the user at the end of the training session.

Intelligent tutoring systems are an interesting field of research, and share many of the same research issues as intelligent environments. Thus, intelligent tutoring systems represent a plausible next step for AFAIK (see Section 6.1).

# *Chapter 3*

# DESIGN FEATURES

## 3.1. AFAIK IS INTERACTIVE

Interactivity is an important design characteristic for systems that deal with human users. An interactive computer program can directly respond to a user, and the operation of an interactive program depends on the user's input.[3] Usually, the flow of communication with an interactive program is bi-directional between the program and the user. In contrast, a printed user's manual is not interactive. The flow of communication is usually in one direction: from the manual to the user. The printed manual reads the same regardless of the user.

One method of capturing information about the Intelligent Room is to write a printed user's manual. When a user wants to learn about the Intelligent Room, she can read through this manual. If she wants to learn about a specific topic, she can look through the index and find where the information for the topic is. However, this approach does not scale well. Reading through a three inch thick manual is tedious, especially if the user only wants to find out how to turn on the lights. Updating a printed manual may also be problematic for anyone who adds or modifies information. In addition, a printed manual does not take full advantage of the Intelligent Room's capabilities.

AFAIK avoids these pitfalls by stressing interactivity to provide a better experience for the user. AFAIK presents help information to the user in electronic

[3] Webster's Dictionary

form, which allows the information to by dynamically changed. Searching through a large knowledge base is much simpler when the knowledge base is in electronic form.

In addition, one consequence of interactivity is that it enables a user to learn procedural information, and then apply the information. For example, in the Microsoft Windows help system, a user can read about "emptying the recycle bin," and then actually have the Windows help system empty the recycle bin. AFAIK functions in much the same way. A user reads about how to "turn on the lights," and then is able to have the Intelligent Room actually turn on the lights.

## 3.2. AFAIK IS MULTI-MODAL

The Intelligent Room supports human-computer interaction through numerous modalities. As previously mentioned, the Intelligent Room's cameras and microphones enable gesture recognition and speech recognition. AFAIK takes advantage of these numerous modalities by presenting information through multiple channels and receiving information via multiple channels. For example, a user can query help by either typing in a question or speaking a question. Or, when the Intelligent Room incorporates gesture recognition in the near future, a user can query help by shrugging her shoulders. In addition, a user can select a link in a help file by using a mouse or pointing to the link with her finger. Similarly, the user receives the requested information via a projected display or by the computer speaking back to the user.

One benefit of using multiple modalities is increased flexibility. With multiple modalities, a user can select her preferred way of interacting with AFAIK, and control how AFAIK responds back. [Oviatt00] A user can use a single method, such as browsing with a keyboard and mouse, or a combination of multiple methods to efficiently interact with AFAIK. When a single modality is not available, having multiple modalities allows AFAIK to interact with a user through alternate modalities. For example, if all the projectors in the Intelligent Room are busy displaying information, then the user can still interact with AFAIK by using the speech interface. Therefore, AFAIK's use of multiple modalities accommodates a wider range of users and situations than a similar system with a single modality. [Oviatt00]

Another benefit of using multiple modalities is the possibility of better learning. There is biological evidence that sensory information is accumulated, and not averaged across multiple modalities; combining information from multiple modalities results in a stronger neural reaction than the effects of a single modality. [Sharma98] Therefore, using multiple modalities is an efficient way of transferring information from AFAIK to a user.

## 3.3. AFAIK USES STRUCTURED INFORMATION

There are many ways of classifying information. One common method of classifying information is to distinguish between procedural information and model information. [Jerrams-Smith88] Procedural information gives users procedures on how to perform tasks. For example, an ordered list of what buttons to push on a telephone to dial a number is procedural information. Model information gives users knowledge of the underlying model of the situation or application. For example, an explanation of why the telephone rings when someone dials its number is model information. Another way of classifying information is to arrange information by detail level. For example, the operation of a telephone can be described by a short introductory sentence, or by a detailed report.

AFAIK supports the classification of information, using any number of classifications, by using XML as the language for its help files. XML, or eXtensible

Markup Language, is the universally accepted format for structured data.[4] XML allows textual information to be "marked up" in an application independent and platform-independent manner. For example, the following XML fragment describes that the movie "AI" has an actor named Haley Joel Osment.

```
<movie>
  <title>AI</title>
  <year>2001</year>
  <actors>
    <actor>
      <name>Haley Joel Osment</name>
      <quote>I'm a real boy.</quote>
    </actor>
  </actors>
</movie>
```

With XML, AFAIK can distinguish between procedural information and model information, or between high-level information and low-level information. Section 4.1 describes the organization of information within a help file in more detail.

## 3.4. AFAIK IS EASY TO USE

As with almost every system, ease of use should be a major design goal. AFAIK has two categories of users, both of which require ease of use.

Information suppliers are usually researchers and developers in the Intelligent Room project. These researchers and developers write the software that controls the Intelligent Room, install the hardware that operate in the Intelligent Room, and plan the interactions in the Intelligent Room. Developers can easily add or modify information within help files because of the structured nature of AFAIK's help files, and the use of a standardized language (see Section 3.3).

[4] World Wide Web Consortium

The requestors of knowledge are the users of the Intelligent Room, namely the people who want to learn how to use the Intelligent Room. AFAIK is easy to use from their perspective as well. The user does not have to say more than a few phrases or click more than a few links to reach the desired information.

*Chapter 4*

# OVERVIEW OF AFAIK

AFAIK uses elements from Metaglue, XML, HTML, Java, and JavaHelp. This chapter describes in detail the major components of AFAIK, and how all the components work together to provide help to a user in the Intelligent Room.

## 4.1. HELP FILES ENCAPSULATE KNOWLEDGE

The help file is a basic unit of knowledge in AFAIK. Each help file contains knowledge about a specific topic, where a topic can be an agent, a device, or any other resource. For example, a telephone in the Intelligent Room might have a help file that describes how a person can use the telephone, and the lighting system might have a help file that describes how a person can operate the lights in the Intelligent Room. A help file essentially encapsulates the knowledge of a researcher regarding a specific topic.

### 4.1.1. Help files are written in XML

As mentioned in section 3.3, AFAIK's help files are written in XML. The following section describes how help files are written in XML, and a sample XML help file is given in Appendix A. The XML help file begins with the `<helpfile>` root element and ends with the `</helpfile>` element.

```
<helpfile>
  … contents go here …
</helpfile>
```

#### *4.1.1.1. XML help files support standard HTML elements*

XML help files support a number of standard HTML elements, such as bold, italics, underline, paragraph, unordered list, and line break. These tags format text in XML help files. XML help files also support embedded images, using the following format.

```
<img src="image.jpg">
  This is the caption for the image.
</img>
```

Finally, XML help files use the HTML hyperlink `<a>` tag to allow a person viewing a help file to instantaneously jump to a Uniform Resource Locator. The support of hyperlinks provides increased interactivity in the browsing of help files.

#### *4.1.1.2. Action elements trigger actions in the Intelligent Room*

In addition to a number of standard HTML elements, XML help files support customized "action" elements. These special elements, such as `<say>` and `<do>`, provide a great deal of interactivity in the help file. The `<say>` element denotes that AFAIK should say the specified text using speech synthesis, and the `<do>` element denotes that AFAIK should say and act upon specified text. In the following example of a command to a Telephone, AFAIK will both say "Dial a number" and perform the specified command when the user selects the action element.

```
<ul>
  <li>
    To dial a phone number, say <do>Dial a number</do>.
  </li>
</ul>
```

The AFAIKActionAgent (see Section 4.2.4) interprets the special action elements and acts upon them. A developer in the Intelligent Room can easily add a new action element to perform a custom action.

#### *4.1.1.3. Help files are referenced by name*

Each help file has a name, which is also used as the file name. For example, a help file for the Telephone is named Telephone, and is located in the file Telephone.xml. The name of the help file, and thus the name of the help topic, is specified in the XML source file using the following format.

```
<name>
  Telephone
</name>
```

In this example, the name of the help file is "Telephone." For simplicity, the name of a help file should be unique, although this uniqueness is not required.

#### *4.1.1.4. Help files are also referenced by keywords*

If help files were only referenced by their names, then the user would have to know the name of a help file in order to query it. Therefore, AFAIK's help files can also be referenced by keywords. These keywords, or key-phrases to be exact, are comma-delimited strings specified in the XML source file. For example, the following example specifies keywords for the Telephone help file.

```
<keywords>
  telephone, phone, ringer, telephone system
</keywords>
```

Thus, if a user requests help about a "ringer", then AFAIK will respond with the Telephone's help file, even though the user did not specifically say "telephone."

#### *4.1.1.5. Help files are divided into subtopics*

The information concerning a specific topic in a help file can be further separated into subtopics. As mentioned in Section 3.3, this structures the information within help files. For example, a help file for the topic of "Telephone" may have the subtopics of Overview, Details, Usage Examples, and Related Links. In this instance, Overview contains a one sentence overview of the Telephone, Details contains model information about the specifics of the Telephone, Usage Examples contains procedural information regarding the Telephone system, and Related Links contains a list of hypertext links to related help files and web sites. The subtopics element of the XML help file denotes the beginning of the subtopics section, which contains the subtopics of the help file.

```
<subtopics>
  … The subtopics go here…
</subtopics>
```

The writer of a help file creates individual subtopics using the subtopic element and the name attribute.

```
<subtopic name="Overview">
  This is the overview sentence.
</subtopic>
```

In the above example, the name attribute identifies the subtopic as the "Overview" subtopic. The subtopics section can contain multiple unique subtopics. For example, the Telephone XML help file might have the following subtopics.

```
<subtopics>

  <subtopic name="Overview">
    The telephone does foo.
  </subtopic>

  <subtopic name="Details">
    The telephone does bar.
  </subtopic>

  <subtopic name="Usage Examples">
    <ul>
      <li>
        To dial a phone number, say <think>Dial a number</think>.
      </li>
      <li>
        To end the phone call, say <think>Hang up</think>.
      </li>
    </ul>
  </subtopic>
```

```
</subtopics>
```

### 4.1.2. The HelpFile Object represents a help file

While AFAIK's help files are written in XML, help files are passed within the AFAIK as serializable Java objects. I created the HelpFile Java object to represent an XML help file in the Metaglue programming environment. The HelpFile Java object is created from the Uniform Resource Locator of the source XML help file.

#### *4.1.2.1. The main methods of HelpFile*

- `get()` returns the textual information within the HelpFile's current subtopic, `getNext()` returns the textual information within the next subtopic, and `get(String)` returns the textual information with the specified subtopic. The AFAIKActionAgent (see Section 4.2.4) uses these methods to determine what text should be spoken by the speech system.

- `getKeywords()` returns an ArrayList of this HelpFile's keywords (see Section 4.1.1.4).

- `getURL()` returns the Uniform Resource Locator of this HelpFile's corresponding HTML file. The AFAIKNavigatorAgent (see Section 4.3.2) uses this method to determine which URL to display on the graphical user interface.

- `toHTML()` generates an HTML equivalent of the XML help file. Section 4.1.2.3 explains why this transformation is useful.

#### *4.1.2.2. The HelpFileExtractor extracts information from XML*

I created the HelpFileExtractor java class to extract the information within an XML file, and store this information using standard Java utility classes, such as ArrayLists, Vectors, and Hashtables. This enables other Java objects and Java agents to efficiently access the information within a help file. The HelpFileExtractor performs the following tasks.

- Extracts the name of the help file, and stores it as a String object in the HelpFile.

- Extracts and parses the comma-delimited keywords. These keyword or key-phrases are stored as individual String objects in an ArrayList. Additionally, if the name of the help file is not listed as a keyword, it is added to the ArrayList as a keyword. Storing keywords in a ArrayList provides convenient and efficient access to the keywords by other Java objects or agents.

- Extracts the names of the subtopics in the help file. These subtopic names, such as "Overview" and "Details" are stored as String objects in an ArrayList.

- Creates Uniform Resource Locator objects for each subtopic section. These URLs allow a user to quickly jump to each subtopic section.

- Extracts the text within each subtopic, and stores the text as String objects in a Hashtable. AFAIK uses the String name of the subtopic as the key of each subtopic Hashtable entry.

- Saves the data within the XML help file as a Java Document Object Model structure. This DOM structure is a hierarchical tree, and is used by the XML Translater (see Section 4.1.2.3) to translate XML into HTML.

#### *4.1.2.3. The XMLTranslator translates XML into HTML*

In addition to extracting information from a source XML help file, AFAIK also translates a source XML help file into an HTML file. This translation is necessary because the design of AFAIK distinguishes the representation of information from the presentation of information. XML is meant to encapsulate information in a meaningful manner, and cannot be displayed by itself. For example, in XML `<li>` could mean a list item element, or the atomic element Lithium. HTML on the other hand, is the standard display formatting language on the Internet.[5] HTML provides numerous text formatting capabilities, and is well suited to the visual presentation of information. Therefore, AFAIK translates XML help files into HTML in order to properly display the information within a help file. An additional benefit of using HTML to display help file information is the availability of third party HTML browsers. If information is in HTML form, a user can use Microsoft's Internet Explorer or Netscape's Communicator, in addition to the AFAIK's JavaHelp AFAIKNavigatorAgent (see Section 4.3.2), to view help files.

I created the XMLTranslator to translate an XML help file into an HTML help file. XMLTranslator obtains a DOM structure from the HelpFileExtractor (see Section 4.1.2.2) and transforms the DOM using XSLT. XSLT, or eXtensible Stylesheet Language Transformations, is a World Wide Web Consortium standard, and is based on XSL style sheets. XSL defines how XML elements will look in HTML, and is itself an XML file. The following example illustrates how XSLT transforms an XML help file into its HTML equivalent; the full XSL used in AFAIK is in Appendix B.

```
<xsl:template match="/">
  <html>
    <xsl:apply-templates select= "helpfile" />
  </html>
</xsl:template>
```

The above code fragment is the first part of the help file XSL document. XSLT looks for the presence of the root element in the XML file, and when found, inserts the beginning `<html>` and ending `</html>` elements into the HTML

[5] World Wide Web Consortium

document. Between these two elements, the XSL document specifies that rules in the template with the name "helpfile" should be used if the `<helpfile>` element is present in the XML file. This template is shown below.

```
<xsl:template match="helpfile">
  <head>
    <title>
      <xsl:apply-templates select="name" />
    </title>
  </head>
  ...
</xsl:template>
```

XSLT looks for the element `<helpfile>` and replaces it with the HTML head and title elements. Next, the XSL document specifies that the rules in the template with the name "name" should be used if the <name> element (see Section 4.1.1.3) is present in the XML file. This template is shown below.

```
<xsl:template match="/name">
  <xsl:value-of select= "text()" />
</xsl:template>
```

The / before "name" specifies that only <name> elements that XSLT should only match <name> elements that are children of the root element. Upon finding a <name> element in the XML file, XSLT adds the text within the <name> element to the HTML file. For the example in Section 4.1.1.3, the following is added to the HTML file.

```
<html>
  <head>
    <title>
      Telephone
    </title>
  </head>
  ...
</html>
```

XSLT transforms the rest of the XML help file, including the subtopics, in a similar manner. In most cases, the XSL transformations simply match XML elements with their HTML equivalent. For example, bold text marked by <b> elements in the XML file is transformed into bold text using <b> elements in HTML.

Transforming XML into HTML can be computationally expensive, and can take on the order of seconds for a large help file. Therefore, to avoid this computation, AFAIK first checks to see if the HTML equivalent of an XML help file exists. If the HTML file exists and is up to date, then AFAIK will not regenerate it.

## 4.2. AFAIK'S MAIN AGENTS

### 4.2.1. The JavaHelpAgent interfaces AFAIK and JavaHelp

AFAIK uses Sun's JavaHelp system to visually display and navigate through help files. I created the JavaHelpAgent to enable AFAIK to use Sun's JavaHelp system. This section gives an overview of JavaHelp, and describes how the JavaHelpAgent generates the files needed by JavaHelp to display help files. Section 4.3.2 describes how AFAIK uses JavaHelp to display and navigate help files.

#### *4.2.1.1. Overview of Sun's JavaHelp system*

Sun Microsystems developed JavaHelp as an extensible help system. It is similar to Microsoft's Windows Help system, and is designed to display help files in HTML format. JavaHelp is written completely in Java, and thus integrates well into the Java-based Metaglue Agent environment. JavaHelp provides a platform independent and an easy to use front end for navigating through help files. Platform independence is a necessary feature for AFAIK, because the Intelligent Room uses a mixture of Windows and Unix based computers. JavaHelp enables a user to view help files from either operating system.

#### *4.2.1.2. The main methods of JavaHelpAgent*

The JavaHelpAgent provides four methods, each of which generates a specific file needed by Sun's JavaHelp System. These four methods are called when AFAIK is started, or whenever a help file is added or removed from AFAIK.

- `generateHelpSet()` generates the JavaHelp helpset file. The helpset file is an XML file that specifies the overall look and organization of the JavaHelp AFAIKNavigatorAgent (see Section 4.3.2). The helpset file typically specifies two adjacent windows, with the table of contents in the left window, and the content of help files in the right window.

- `generateTOC()` generates the JavaHelp table of contents file. The table of contents file is an XML file that lists all the help file topics and subtopics in AFAIK. This file also specifies the ordering of the topics and subtopics listed in the JavaHelp navigator.

- `generateIndex()` generates the JavaHelp index file. The index file is similar to the table of contents file, for it contains the same list of help file topics as the table of contents files. However, for convenience, the table of contents is alphabetized. The alphabetical sorting is accomplished by storing the topics in a `SortedTree` container.

- `generateMap()` generates the JavaHelp map file. The map file is an XML file that specifies the correspondence between pages in the JavaHelp navigator with the HTML URLs of the help file topics. AFAIK iterates through the all help files in AFAIK, and stores the URLs of all the topics and subtopics found.

### 4.2.2. The HelpFileManagerAgent manages the knowledge base

The HelpFileManagerAgent is responsible for adding, maintaining, and removing help files in AFAIK. The HelpFileManagerAgent is essentially a database of help files, and acts as the central knowledge base in AFAIK.

When AFAIK starts, the HelpFileManagerAgent loads a predefined set of help files. A programmer specifies these initial help files in a simple text list, and the HelpFileManagerAgent loads these help files into the knowledge base using the `add(java.util.Container)` method.

#### *4.2.2.1. The main methods of HelpFileManagerAgent*

- `add(String)` adds a help file to the knowledge base. When the HelpFileManagerAgent adds a help file, it creates a HelpFile object from the source XML help file, stores the help file's keywords, and adds the HelpFile object to a SmartVector container. I wrote the SmartVector container as an extension of Java's Vector container class. SmartVector adds the ability to query the access times and access counts of the items in the container; this is useful for example for finding the last accessed HelpFile, or for finding the most popular HelpFile.

- `add(java.util.Container)` adds multiple help files to the knowledge base. This is similar to `add(String)`, and is used upon the startup of AFAIK to load the initial help files.

- `get(String)` returns an ArrayList of all HelpFile objects that have a given keyword. There could be no, one, or more than one HelpFile that has a given keyword. This method will also retrieve HelpFiles with a specified name, because the name of a help file is also a keyword of that help file. By storing loaded HelpFiles in a SmartVector container, the HelpFileManagerAgent can also retrieve the previously requested HelpFile and the most requested HelpFile. The `getLast()` and `getMost()` methods respectively perform these tasks.

- `contains(String)` returns whether or not the HelpFileManagerAgent contains a HelpFile with the given keyword or name.

- `getKeywords()` returns an ArrayList of all the known keywords in the HelpFileManagerAgent. This set of keywords is the union of the keywords of all individual help files loaded into the HelpFileManagerAgent.

- `getSubtopicNames()` method returns an ArrayList of all the subtopic names in the HelpFileManagerAgent. The AFAIKSpeechAgent, described in section 4.3.1, uses both `getKeywords()` and `getSubtopicNames()` for the speech system.

### 4.2.3. The AFAIKAgent is the heart of AFAIK

The AFAIKAgent bridges agents within AFAIK with agents not in AFAIK. For example, the Telephone agent can ask the AFAIKAgent to provide help for a certain feature of the telephone. Or, the speech system hears a question from a person in the Intelligent Room, and asks the AFAIKAgent to respond to the question. In each case, the AFAIKAgent queries the other agents in AFAIK, such as the HelpFileManagerAgent and the AFAIKActionAgent, to come up with a coherent response to the request for help.

#### *4.2.3.1. The main methods of AFAIK*

- `ask(String)` queries AFAIK for help about a given keyword. For example, `ask("telephone")` will trigger AFAIK to respond with help for the Telephone. This method returns the relevant information, and requests the

AFAIKActionAgent (see Section 4.2.4) to present the information to the user. In addition, the `ask(String,String)` method queries AFAIK about a given keyword and a given subtopic. For example, `ask("telephone","usage examples")` will trigger AFAIK to respond with usage examples for the topic of Telephone. If there is more than one help file that corresponds with a given keyword, then the AFAIKAgent will ask the user to select from the matching help files. For example, if the keyword "telephone" corresponds to both the Telephone and the CordlessTelephone, then the AFAIKAgent will prompt the user to select either of the two topics.

### 4.2.4. The AFAIKActionAgent orchestrates the delivery of information

The AFAIKActionAgent has two roles. First, the AFAIKActionAgent distributes information in a HelpFile to agents that display the information to the user. When the AFAIKAgent needs to present information to the user, it instructs the AFAIKActionAgent to present a given HelpFile. The AFAIKActionAgent in turn relies on output agents such as the AFAIKNavigatorAgent and the SpeechTextOutputAgent to display the information in the HelpFile. The second role of the AFAIKActionAgent is interpreting the special action elements in a help file. These action elements, such as <do> and <say>, are described in section 4.1.1.2.

#### *4.2.4.1. The main methods of AFAIKActionAgent*

- `present(agentland.help.helpfile.HelpFile,string)` and `present(agentland.help.helpfile.HelpFile)` channel information within the specified HelpFile to a user. Each of these methods in turn relies on various agents to output information in different forms. For example, section 4.3.2 describes how AFAIK uses the AFAIKNavigatorAgent to graphically display the HTML version of a help file. Section 4.3.3 describes how AFAIK outputs the spoken text to the user.

- `interpretHyperlink(String)` interprets the action elements described in section 4.1.1.2. The AFAIKNavigatorAgent converts the XML action elements described in section 4.1.1.1 to HTML hyperlinks for display. For example, `<say>Hello World</say>` becomes `<a href="say://Hello World">Hello World</a>`. This formatting allows the user to click and activate an action element in the AFAIKNavigatorAgent. When an action element is selected, the AFAIKNavigatorAgent calls `interpretHyperlink(String)` with the text in the `href` field. The AFAIKActionAgent then determines what type of action to take based on the text in the `href` field.

## 4.3. AFAIK'S USER INTERFACE AGENTS

### 4.3.1. The AFAIKSpeechAgent provides speech interaction

The AFAIKSpeechAgent extends the AFAIKAgent by adding speech recognition capabilities. The AFAIKSpeechAgent interfaces AFAIK with Metaglue's speech system, and provides the user with a speech interface to AFAIK. When a user verbally requests help, the Metaglue speech system passes the user's spoken utterance to the AFAIKSpeechAgent. The AFAIKSpeechAgent in turn determines whether this spoken command refers to querying a help topic, or navigating within a help topic. For example, a user can say "Please tell me about the telephone," or "Tell me more about Metaglue." After processing the spoken utterance, the AFAIKSpeechAgent queries the appropriate agents in AFAIK to come up with a coherent response to the request for help.

#### *4.3.1.1. AFAIKSpeechAgent's grammar specifies what the user can say*

An integral part of the AFAIKSpeechAgent is the associated JSGF grammar. A Java Speech Grammar Format compliant grammar file is similar to a context-free grammar, and specifies the verbal phrases that the AFAIKSpeechAgent understands.

This is in contrast to allowing the AFAIKSpeechAgent to understand any English phrase. By limiting the phrases and words that the AFAIKSpeechAgent understands, the AFAIKSpeechAgent ignores irrelevant words and thus performs more accurately. The AFAIKSpeechAgent's grammar file recognizes the following situations.

- User requests general help: "I need assistance," or "Please help me."
- User requests help about a specific topic: "Please tell me about the Telephone," "What is the Telephone," or "I want to know about the Telephone."
- User requests help about a specific subtopic of a specific topic: "Give me an overview of the Telephone," or "Tell me about the usage example's of the Telephone."
- User requests more information about the current topic: "Kindly tell me more."

The above situations are only a few examples of what the AFAIKSpeechAgent's grammar recognizes. The full JSGF grammar is given in appendix C.

#### *4.3.1.2. The main methods of AFAIKSpeechAgent*

- `updateKeywords()` updates the AFAIKSpeechAgent's grammar file with the known keywords in AFAIK. Whenever the knowledge base of AFAIK changes, such as when the HelpFileManagerAgent adds a help file, `updateKeywords()` ensures that any keyword in AFAIK can be recognized by the Metaglue speech system by adding any new keywords to the grammar file.

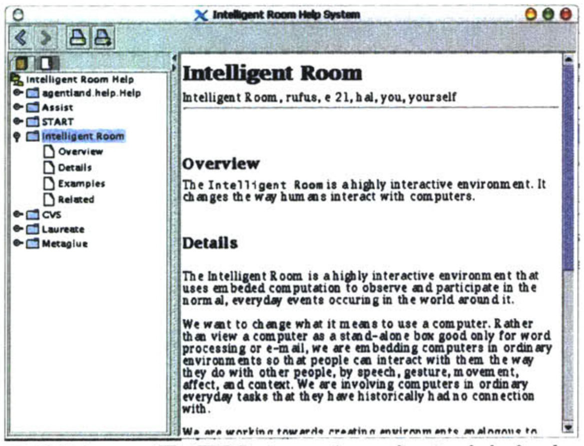
- `updateSubtopicNames()` updates the AFAIKSpeechAgent's grammar file with the known subtopics in AFAIK. This is similar to `updateKeywords()`, and allows the user to ask about any subtopic by voice. For example, a user can say, "show me the details about Metaglue," where "details" is a subtopic of the topic Metaglue.

### 4.3.2. The AFAIKNavigatorAgent provides a graphical user interface

The AFAIKNavigatorAgent is the graphical front end of AFAIK. This agent uses Sun's JavaHelp system (see Section 4.2.1), and creates a JavaHelp navigator to display help files. In addition, the navigator enables a user to have AFAIK perform actions using special action elements (see Section 4.1.1.2). For example, the user can click on the action link `<do>Dial a number</do>`, and AFAIK will dial a number on the telephone.

#### *4.3.2.1. The design of the AFAIKNavigatorAgent is similar to a standard web browser*

The AFAIKNavigatorAgent uses a JavaHelp navigator to display help files. Figure 4.3.2.1.a shows a navigator displaying the help file for the Intelligent Room. The left hand window displays either the table of contents or the index. These two lists are generated by the JavaHelpAgent (see Section 4.2.1), and list the help file topics and subtopic available.



**Intelligent Room**

Intelligent Room, rufus, e 21, hal, you, yourself

**Overview**

The Intelligent Room is a highly interactive environment. It changes the way humans interact with computers.

**Details**

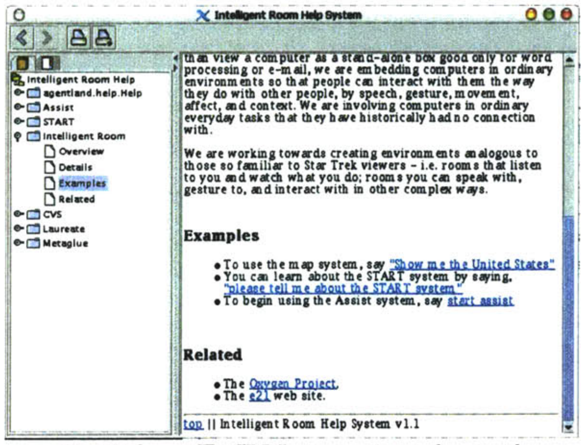
The Intelligent Room is a highly interactive environment that uses embeded computation to observe and participate in the normal, everyday events occuring in the world around it.

We want to change what it means to use a computer. Rather than view a computer as a stand-alone box good only for word processing or e-mail, we are embedding computers in ordinary environments so that people can interact with them the way they do with other people, by speech, gesture, movement, affect, and context. We are involving computers in ordinary everyday tasks that they have historically had no connection with.

We are working towards creating environments analogous to

*4.3.2.1.a – AFAIKNavigatorAgent showing help for the "Intelligent Room"*

When a user selects a topic from the left hand window, or when a user requests a help topic by voice (see Section 4.3.1), the AFAIKNavigatorAgent displays



than view a computer as a stand-alone box good only for word processing or e-mail, we are embedding computers in ordinary environments so that people can interact with them the way they do with other people, by speech, gesture, movement, affect, and context. We are involving computers in ordinary everyday tasks that they have historically had no connection with.

We are working towards creating environments analogous to those so familiar to Star Trek viewers - i.e. rooms that listen to you and watch what you do; rooms you can speak with, gesture to, and interact with in other complex ways.

**Examples**

- To use the map system, say "Show me the United States"
- You can learn about the START system by saying, "please tell me about the START system"
- To begin using the Assist system, say start assist

**Related**

- The Oxygen Project.
- The e21 web site.

top || Intelligent Room Help System v1.1

*4.3.2.1.b – AFAIKNavigatorAgent showing the Examples section of "Intelligent Room"*

the relevant help file in HTML form on the right hand window of the JavaHelp navigator. This window is a customized HTML browser, and can display plain text, formatted text, images, lists, and other standard HTML elements. Figure 4.3.2.1.b shows that when a user selects a subtopic of a help file topic, the AFAIKNavigatorAgent focuses the navigator's content window on the specific subtopic.

*4.3.2.2. The main methods of AFAIKNavigatorAgent*

- `display(agentland.help.helpfile.HelpFile)` displays the specified help file to the user via the JavaHelp navigator. The JavaHelpNavigatorAgent gets the current subtopic of the HelpFile via the `get()` method (see Section 4.1.2.1), and outputs the information in the right hand content window.

- `display(java.net.URL)` displays any generic HTML page. As mentioned in section 4.2.1, the JavaHelp browser is capable of displaying generic HTML pages. This ability is useful for display third-party information, such as a manual from a manufacturer's web site.

### 4.3.3. SpeechTextOutput provides speech synthesis

The SpeechTextOutputAgent is part of the Metaglue speech system, and is used as one of the primary means of conveying information to the user. The SpeechTextOutputAgent receives text from other agents, and speaks this information to a user using Metaglue's text-to-speech synthesizer. As of the time of writing, Metaglue's text-to-speech synthesizer uses a combination of IBM's ViaVoice text-to-speech synthesizer and British Telecom's Laureate text-to-speech synthesizer.

The SpeechTextOutputAgent handles large amounts of text by periodically asking the user if he or she would like to continue. If the user answers "no," then the SpeechTextOutputAgent silences the speech synthesizer. This feature is necessary because AFAIK sometimes outputs a large amount of information via speech. While other modes of interaction, such as visual displays, are more appropriate for presenting large amounts of information, this feature allows AFAIK to use speech out without encumbering the user.

*Chapter 5*

# HOW AFAIK WORKS

## 5.1. Scenario: Bob uses the Intelligent Room to make a phone call

[Bob walks into the Intelligent Room.]

Bob: "What can I do in this Room?"

[A projector turns on and projects a display on a wall in the Intelligent Room. The display shows the following information, while the Intelligent Room speaks the same information through speakers in the Intelligent Room.]

AFAIK: "You can ask about a help topic by saying, tell me about, and then the topic. For example, you can find out about the lights in the Intelligent Room by saying, tell me about the lights. Or, you can find out about the telephone by saying, tell me about the telephone."

Bob: "Ok then, please tell me about the phone."

[The display on the wall changes, and shows the Telephone help file. At the same time, the Intelligent Room speaks the following.]

AFAIK: "The Telephone is located on the main desk in the Intelligent Room. You can make phone calls using the touch pad or by using your voice."

Bob: "How do I use the telephone?"

[The display scrolls down the Telephone help file to a section called Examples. At the same time, the Intelligent Room speaks the following.]

AFAIK: "To make a phone call, say make a phone call, followed by the name of the person you want to call, or the phone number that you want to dial. To end the phone call, say hang up. If you would like, you can also click on these links, and I will show you."

[Bob clicks on the link that says "make a phone call," and the speakers in the Intelligent Room play the sound of a dial tone.]

## 5.2. WHAT HAPPENS IN AFAIK DURING BOB'S SCENARIO

### 5.2.1. When the Metaglue system initializes

- The agents in the AFAIK system start up. This includes the AFAIKAgent, AFAIKSpeechAgent, HelpFileManagerAgent, JavaHelpAgent, AFAIKActionAgent, and the AFAIKNavigatorAgent.

- The HelpFileManagerAgent loads the initial help files, which includes IntelligentRoom.xml and Telephone.xml. AFAIK also translates the XML help files into HTML so that they are compatible with Sun's JavaHelp navigator.

- The AFAIKSpeechAgent obtains all the keywords in the help files from the HelpFileManagerAgent, and dynamically adds these keywords to the AFAIKSpeechAgent's grammar. This allows the user to verbally refer to any of the help files in AFAIK.

- The HelpFileManagerAgent instructs the JavaHelpAgent to generate the index and table of contents files needed by Sun's JavaHelp system.

### 5.2.2. When Bob asks, "What can I do in this Room?"

- The HelpSpeechAgent receives the sentence, "What can I do in the room," which matches one of the rules in the AFAIKSpeechAgent's grammar. This rule signals the AFAIKSpeechAgent to call the method `get("Intelligent Room")` from the HelpFileManagerAgent.

- The HelpFileManagerAgent passes the HelpFile object that has the keyword, "Intelligent Room," to the AFAIKSpeechAgent. The HelpFileManagerAgent notes that this HelpFile has just been accessed, and the AFAIKSpeechAgent notes that this HelpFile is the current help topic.

- The AFAIKSpeechAgent sets the current subtopic to, "Usage Examples," and instructs the AFAIKActionAgent to present this HelpFile.

- The AFAIKActionAgent receives the HelpFile from the AFAIKSpeechAgent, and gets the text of the current subtopic. The AFAIKActionAgent passes this text to the SpeechTextOutputAgent to verbally speak the text to the user. In addition, the AFAIKActionAgent instructs the AFAIKNavigatorAgent to display the HelpFile.

- The AFAIKNavigatorAgent starts a new JavaHelp navigator with the index files provided by the JavaHelpAgent. After the navigator starts up, it displays the HelpFile for the "Intelligent Room," and scrolls to the subtopic, "Usage Examples."

### 5.2.3. When Bob asks, "Ok then, please tell me about the phone."

- The AFAIKSpeechAgent receives the sentence, "Ok then, please tell me about the phone." This sentence matches a rule in the AFAIKSpeechAgent's grammar, and triggers the AFAIKSpeechAgent to call `get("phone")` from the HelpFileManagerAgent.

- The HelpFileManagerAgent passes the HelpFile object that has the keyword, "Intelligent Room," to the AFAIKSpeechAgent. The HelpFileManagerAgent notes that this HelpFile has just been accessed, and the AFAIKSpeechAgent notes that this HelpFile is the current help topic.

- The AFAIKSpeechAgent sets the current subtopic to, "Overview," and instructs the AFAIKActionAgent to present this HelpFile.

- The AFAIKActionAgent receives the HelpFile from the AFAIKSpeechAgent, and gets the text of the current subtopic. The AFAIKActionAgent passes this text to the SpeechTextOutputAgent to verbally speak the text to the user. In addition, the AFAIKActionAgent instructs the AFAIKNavigatorAgent to display the HelpFile.

- The AFAIKNavigatorAgent displays the HelpFile for the "Intelligent Room," on the current JavaHelp navigator, and scrolls to the subtopic, "Overview."

### 5.2.4. When Bob asks, "How do I use the telephone?"

- The AFAIKSpeechAgent receives the sentence "How do I use the telephone," from the Metaglue speech system. This triggers the AFAIKSpeechAgent to instruct the AFAIKActionAgent to present the current help file with the subtopic "Examples."

- The AFAIKActionAgent receives the HelpFile from the AFAIKSpeechAgent, and gets the text of the "Examples," subtopic. The AFAIKActionAgent passes this text to the SpeechTextOutputAgent to verbally speak the text to the user. In addition, the AFAIKActionAgent instructs the AFAIKNavigatorAgent to display the HelpFile.

- The AFAIKNavigatorAgent displays the HelpFile for the Telephone on the current JavaHelp navigator, and scrolls to the subtopic, "Examples."

### 5.2.5. When Bob clicks on the link, "Make a phone call."

- The AFAIKNavigatorAgent detects that the user clicked on a hyperlink, which in this case is `do://Make a phone call`. The AFAIKNavigatorAgent sends this hyperlink to the AFAIKActionAgent.

- The AFAIKActionAgent interprets the do command, and relies on other agents in the Metaglue system to perform the command "make a phone call." Essentially, the Metaglue system "pretends" that the user actually said "make a phone call," while the user only clicked on the link. If another agent's grammar understands "make a phone call," then that agent will act upon the utterance.

- The TelephoneAgent's grammar has a rule that matches the utterance, and initiates a phone call.

# *Chapter 6*

# NEXT STEPS

## 6.1. FROM A HELP SYSTEM TO A TUTORING SYSTEM

AFAIK is an intelligent help system for the Intelligent Room. It provides help to a user when the user requests help. The next step is to evolve AFAIK into an intelligent tutoring system (see Section 2.3.2). This involves not only providing information on request, but also proactively providing helpful information.

This improvement will lead to the Intelligent Room becoming a better learning environment. The Intelligent Room can attempt to predict the user's goals, and when the user falters, provide assistance to the user. However, intelligent tutoring systems involve numerous research issues such as user modeling and tutorial strategy. It is difficult to determine the user's goals [Carrol88], and especially difficult when the user is placed into a complex environment such as the Intelligent Room.

## 6.2. A DEMONSTRATION SYSTEM TO DEMO THE INTELLIGENT ROOM

During the development of AFAIK, the researches in the Intelligent Room project gave numerous demonstrations of the Intelligent Room to supporters and sponsors. As a next step, AFAIK can be modified to give interactive and autonomous demonstrations of the Intelligent Room.

For example, a number of help files can be combined to present various features of the Intelligent Room to users viewing the demonstration. The Intelligent

Room can prompt the users to speak certain commands or select certain active links to trigger the Intelligent Room to perform certain actions.

## 6.3. A USER STUDY TO VERIFY USABILITY

One of the design goals of this project was ease of use (see Section 3.4). The best method of determining AFAIK's usability is to conduct a usability study. Depending on the results of the usability study, the AFAIK's user interface and user interactions can be fine-tuned.

A test group composed of researchers in the Artificial Intelligence Laboratory can give insight into how well AFAIK performs with its target audience. On the other hand, a test group composed of random people from the general population can give insight into the suitability of the general concepts of AFAIK, such as interactivity and the use of multiple modalities.

## 6.4. USING START TO ACCESS HELP FILES

Infolab's START system [Katz97] is a natural language query system that has a more powerful language model than the Metaglue speech system's grammars. Incorporating START in the AFAIK can improve access to the information within help files. For example, the user can ask a natural language query, as opposed to what is contained in AFAIKSpeechAgent's grammar (see Section 4.3.1.1), to obtain help from AFAIK.

## 6.5. LEARNING NEW INFORMATION

AFAIK only knows about information within its XML help files. Developers in the Intelligent Room Project write these help files, and AFAIK loads them upon starting. However, AFAIK is not able to dynamically learn information like a system such as Sally. [Groh99] Therefore, a next step in the development of AFAIK is to enable it to learn new information. New information would be categorized as model-based or procedural-based, and dynamically entered into an XML help file. This ability to dynamically learn information results in easier input of knowledge into a help system.

*Chapter 7*

# CONTRIBUTIONS

This thesis describes the development of a help system called AFAIK. As of April 2001, AFAIK is operating in the Intelligent Room at MIT's Artificial Intelligence Laboratory. Within the Intelligent Room project, AFAIK provides autonomous, interactive, and multi-modal help to users of the Intelligent Room.

As a result of this thesis, developers can add knowledge to AFAIK by writing simple XML files. Users of the Intelligent Room can request and receive help using either a graphical user interface or a speech interface. AFAIK was constructed using the Metaglue agent programming system and takes advantage of Metaglue's software infrastructure. In addition, AFAIK provides a foundation for the next help system for the Intelligent Room. Much of the software written for AFAIK can be used for future software projects using XML or JavaHelp.

Finally, the design ideas of this thesis extend beyond AFAIK and the Intelligent Room project. Many software systems, including help and tutorial systems, can benefit from using multiple modalities to interface with a human user. Software systems that focus on interactivity will improve the interface even further. Add autonomy as a feature, and the resulting system will be more "intelligent" and bring the focus back on the human user.

# *Chapter 8*

## 8.1. Appendix A: A Sample XML Help File

```
<?xml version='1.0' encoding='utf-8'?>
<helpfile version="1.0">
  <name>Intelligent Room</name>

  <keywords>Intelligent Room, rufus, e 21, hal, you,
  yourself</keywords>

  <subtopics>
    <subtopic name="Overview">The
    <code>Intelligent Room</code>

    is a highly interactive environment. It changes the way humans
    interact with computers.</subtopic>

    <subtopic name="Details">
      <p>The Intelligent Room is a highly interactive environment
      that uses embeded computation to observe and participate in
      the normal, everyday events occuring in the world around
      it.</p>

      <p>We want to change what it means to use a computer. Rather
      than view a computer as a stand-alone box good only for word
      processing or e-mail, we are embedding computers in ordinary
      environments so that people can interact with them the way
      they do with other people, by speech, gesture, movement,
      affect, and context. We are involving computers in ordinary
      everyday tasks that they have historically had no connection
      with.</p>

      <p>We are working towards creating environments analogous to
      those so familiar to Star Trek viewers - i.e. rooms that
      listen to you and watch what you do; rooms you can speak
      with, gesture to, and interact with in other complex
      ways.</p>
    </subtopic>

    <subtopic name="Examples">
      <ul>
        <li>To use the map system, say
        <think>Show me the United States</think>.
        </li>

        <li>You can learn about the START system by saying,
        <think>please tell me about the START system</think>.
        </li>

        <li>To begin using the Assist system, say
        <think>start assist</think>.
        </li>
      </ul>
    </subtopic>

    <subtopic name="Related">
      <ul>
        <li>The <a href="http://oxygen.lcs.mit.edu">Oxygen
Project</a>.</li>
        <li>The <a href="http://e21.ai.mit.edu">e21</a> web
site.</li>
      </ul>
    </subtopic>
  </subtopics>
</helpfile>
```

## 8.2. Appendix B: A Sample XSL Document

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" omit-xml-declaration="yes" indent="yes"
  doctype-public="-//W3C//DTD HTML 4.0 Transitional//EN" />

  <!-- removes white space -->
<!--  <xsl:strip-space elements="*" /> -->


  <!-- first template to be matched -->
  <xsl:template match="/">
    <html>
      <xsl:apply-templates select="helpfile" />
    </html>
  </xsl:template>


  <!-- root helpfile element match -->
  <xsl:template match="helpfile">
    <head>
      <title>
        <xsl:apply-templates select="name" />
      </title>
    </head>

    <body>
```

```
     <h1>
       <a name="name">
         <xsl:apply-templates select="name" />
</a>
     </h1>

     <xsl:apply-templates select="keywords" />

     <hr />

     <xsl:apply-templates select="subtopics/*" />

     <hr />
     <a href="#name">top</a> ||  Intelligent Room Help System v1.1
   </body>
 </xsl:template>


 <!-- match name element that is a child of helpfile -->
 <xsl:template match="/name">
   <xsl:value-of select="text()" />
 </xsl:template>


 <!-- match keywords element that is a child of helpfile -->
 <xsl:template match="/keywords">
   <p>Keywords:
   <i>
     <xsl:apply-templates />
   </i>
   </p>
 </xsl:template>


 <!-- match all subtopics -->
 <xsl:template match="subtopics/*">
   <p>
     <h2>
       <a name="{@name}">
         <xsl:value-of select="@name" />
       </a>
     </h2>

     <xsl:apply-templates />
   </p>
 </xsl:template>



 <!-- match standard HTML tags -->
```

```
<xsl:template match="p">
  <p>
    <xsl:apply-templates />
  </p>
</xsl:template>

<xsl:template match="b">
  <b>
    <xsl:apply-templates />
  </b>
</xsl:template>

<xsl:template match="i">
  <i>
    <xsl:apply-templates />
  </i>
</xsl:template>

<xsl:template match="u">
  <u>
    <xsl:apply-templates />
  </u>
</xsl:template>

<xsl:template match="image">
  <center>
    <img src="{@src}" />

    <br />

    <xsl:value-of select="text()" />
  </center>
</xsl:template>

<xsl:template match="a">
  <a href="{@href}">
    <xsl:value-of select="text()" />
  </a>
</xsl:template>

<xsl:template match="br">
  <br>
    <xsl:apply-templates />
  </br>
</xsl:template>

<xsl:template match="pre">
  <pre>
    <xsl:apply-templates />
  </pre>
```

```
</xsl:template>

<xsl:template match="ul">
  <ul>
    <xsl:apply-templates />
  </ul>
</xsl:template>

<xsl:template match="li">
  <li>
    <xsl:apply-templates />
  </li>
</xsl:template>

<xsl:template match="code|tt">
  <tt>
    <xsl:apply-templates />
  </tt>
</xsl:template>

<xsl:template match="table">
  <table>
    <xsl:apply-templates />
  </table>
</xsl:template>

<xsl:template match="tr">
  <tr>
    <xsl:apply-templates />
  </tr>
</xsl:template>

<xsl:template match="td">
  <td>
    <xsl:apply-templates />
  </td>
</xsl:template>


<!-- match custom action elements -->
<xsl:template match="subtopics//think|subtopics//do">
  <a href="think://{text()}"><xsl:value-of select="text()"/></a>
</xsl:template>

<xsl:template match="subtopics//print">
  <a href="print://{text()}"><xsl:value-of select="text()"/></a>
</xsl:template>

<xsl:template match="subtopics//say">
  <a href="say://{text()}"><xsl:value-of select="text()"/></a>
  </xsl:template>

</xsl:stylesheet>
```

## 8.3. Appendix C: A JSGF Grammar File

```
grammar agentland.help.AFAIK;

// import polite, articles, prepositions, etc
import <speech.lib.language.*>;
import <speech.lib.polite.*>;
import <speech.lib.verbs.*>;
import <speech.lib.states.*>;

// these will be dynamically updated by the HelpSpeech system
<subtopic> = overview | body;
<keyword> = help | tutorial;

// a wrapper to go around the keywords, so you can say "the lights"
<keywordWrapper> = ( [<preposition>] [<article>] ( <keyword>
{keyword} ));

// a wrapper to go around the subtopics, so you can say "an overview"
<subtopicWrapper> = ( [<preposition>] [<article>] (<subtopic>
{subtopic}));

// tell me about the lights
<searchByKeyword> = ( <queryVerb> <person> <keywordWrapper> ) {
keywordSearch };

// show me the overview
<searchBySubtopic> = ( <queryVerb> <person> <subtopicWrapper> ) {
subtopicSearch };

// show me examples of the lights
<searchByAll> = ( <queryVerb> <person> <subtopicWrapper>
<keywordWrapper>) { fullSearch };

<help> = ( help | assist | aid | assistance );

// tell me more
<moreInformation> = ( <queryVerb> <person> ( more ) ) { more };

// help me
// I need assistance
<helpSystem> = ( <help> me | <person> need <help> ) { helpSystem } ;
```

```
// wrapper for all commands, so that you can say something polite
like please
public <politeWrapper> = [<polite>] ( <searchByKeyword> |
<searchBySubtopic> | <searchByAll> | <helpSystem> | <moreInformation>
| <how> | <what> | <where>) [<polite>];

// tell me how do I turn off the projector
// how can we activate the ESP
<how> = ( [<queryVerb> <person>] how ( ([<person>] ( do | can |
should | could | would | may ) [<person>]) | to) <actionVerb>
<keywordWrapper>) { how };

<actionVerb> = ( turn ( on | off) ) | activate | deactivate | play |
show | use | operate |engage | disable | kill | run | do | ask;

// what can I ask (note: what can I say is reserved)
<what> = ( what ( can | may ) <person> <actionVerb>) { what };

// where am I
// where is the projector
<where> = ( where ( am | are | is ) ( <person> | <keywordWrapper> ) )
{ where };
```

# BIBLIOGRAPHY

[Alvarado00] Alvarado, Christine. A natural sketching environment: Bringing the computer in early stages of mechanical design. Master's Thesis. Massachusetts Institute of Technology, Cambridge, MA, 2000.

[Byerley88] Byerley, P.F., Brooks, P., Elsom-Cook, M., Spensley, F., Scaroni, C., and Federici, M. An intelligent tutoring system for procedural skills 'Dominie'. In *Proceedings of the IEEE Colloquium on Intelligent Tutorial Systems.* 5/1-5/9, 1988.

[Carroll88] Carroll, John M., and Aaronson, Amy P. Learning by doing with simulated intelligent help. In *Communications of the ACM*, vol. 31, no 9, 1064-1079, 1988.

[Coen97] Coen, Michael. Building Brains for Rooms: Designing Distributed Software Agents. In *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence*, Providence, R.I. 1997.

[Coen98] Coen, Michael. Design Principles for Intelligent Environments. In *Proceedings of AAAI 1998 Spring Symposium on Intelligent Environments*, AAAI Technical Report SS-98-02.

[Groh99] Groh, Marion L. An Interactive Multimedia Continuously Learning Helpdesk System (When Hal Met Sally). Master's Thesis. Massachusetts Institute of Technology, Cambridge, MA, 1999.

[Jerrams-Smith88] Jerrams-Smith, J. Provision of Explanations by an Intelligent Tutorial System. In *Proceedings of the IEEE Colloquium on Intelligent Tutorial Systems.* 1/1-1/2, 1988.

[Katz97] Katz, Borris. "From Sentence Processing to Information Access on the World Wide Web," *AAAI Spring Symposium on Natural Language Processing for the World Wide Web*, Stanford University, Stanford, CA, 1997.

[Morrisroe88] Morrisroe, G.C. An application of intelligent tutoring systems to air traffic control training. In *Proceedings of the IEEE Colloquium on Intelligent Tutorial Systems.* 3/1-3/3, 1988.

[Oviatt00] Oviatt, S.L., Cohen, P.R., Wu, L., Duncan, L. Suhm, B., Bers, J., Holzman, T., Winograd, T., Landay, J., Larson, J. & Ferro, D. Designing the user interface for multimodal speech and gesture applications: State-of-the-art systems and research directions. *Human Computer Interaction*, 2000, vol. 15, no. 4, 263-322.

[Sharma98] Sharma, Rajeev, Pavlovic, Vladimir I., Huang, Thomas S. Toward Multimodal Human-Computer Interface. In *Proceedings of the IEEE*, vol. 86, no. 5, May 1998.

[Stein98] Stein, Adelheit. Active Help and User Guidance in a Multimodal Information System: A Usability Study. Workshop Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen. 87-98, 1998.