Inquiry: The University of Arkansas Undergraduate Research Journal

Volume 6 Article 22

Fall 2005

Interfacing Agents with Natural Language

Jared Allen
University of Arkansas, Fayetteville

INTERFACING AGENTS WITH NATURAL LANGUAGE

By Jared Allen
Department of Computer Science

Faculty Mentor: Dr. Craig Thompson
Department of Computer Science

Acknowledgments:

I would first like to thank my mentor, Dr. Craig Thompson, for sharing the inspiration, the knowledge, and the experience that began this project. I would also like to offer my gratitude to each member of my Honors Defense Committee for their cooperation under a hectic schedule: Dr. Thompson, CSCE; Dr. Fred Davis, ISYS; Dr. Fiona Davidson, GEOG; and Dr. Russell Deaton, CSCE. And finally, I would like to thank everyone that I worked with on this project: Micah Byers, Quang Duong, Hieu Nguyen, Edmundo Ruiz, Minh Vu, and all the other students on the EiA team.

Abstract

Technology is leading us to a world where computers are everywhere. A new breed of machines capable of sensing and reacting to stimuli in the real world is under development. Unfortunately, these new, powerful devices can oftentimes be difficult for the average person to understand. It is imperative that an easy to use interface be implemented to usher in this new world. Natural language (speech) would be an ideal solution. However, it has proven implausible on a large scale. A Menu-Based Natural Language Interface (MBNLI) could retain the intuitiveness of speech, while eliminating the obstacles impeding implementation. This research paper describes the development and applications of a MBNLI.

1. Introduction

1.1 Problem

The world is becoming increasingly dependent upon machines. Electronic devices are incorporated into almost every aspect of our lives. TV's, microwaves, and washing machines are common household objects. Grocery stores are including self-checkout lanes and electronic information kiosks to their list of services. Small computers and navigation systems are finding their way into vehicles. The problem with this trend is that many of these devices can be difficult for users to understand and use. Consider a modern household thermostat. One of these machines is capable of regulating the temperature to different settings in every room of the house, changing

these settings based on the time of day, and even adjusting the temperature for different days of the week. Although a very powerful tool, it lacks ease of use. A novice user could have great difficulty determining which sequence of buttons is needed to lower the temperature by a few degrees as opposed to, say, setting the living room temperature to 68° on Tuesday. For someone who simply wants to cool off, the complexity of the device only gets in the way.

Another problem that comes with the incorporation of machines into daily life is the added hardware that comes with each entity. TVs, VCRs, stereos, DVD players, and a host of others all come equipped with their own remote control. Assuming technology continues to progress at its current pace, automated devices and custom controllers could overrun the personal and business worlds. The scenario of a different controller for every device does not scale. Clearly, a more space and cost-efficient solution is needed.

1.2 Thesis

The central thesis of this paper is that Menu-Based Natural Language Interfaces (MBNLI) can simplify the interface to machines and can eliminate the need for excess hardware. The potential impact is that MBNLI technology could greatly simplify interactions with electronic devices.

1.3 Introduction to the Everything is Alive Agent System

This research project has been conducted in collaboration with the University of Arkansas "Everything is Alive" (EiA) Agent System Project. The team consists of several undergraduate and graduate students in the Department of Computer Science and Computer Engineering, and is overseen by Dr. Craig Thompson. The EiA vision is a world in which computing is pervasive and any real-world entity has the capability to communicate and interact with its surroundings. The idea is to equip all objects with sensors and means of sending and receiving data. Thus, anything can become an "agent" in a system, including people, pets, machines, furniture, places, and even abstractions.

Consider a "Smart Tree," for example, which is outfitted with sensors and wireless communications. The tree can sense when the ground is drying out, and inform a nearby sprinkler agent that it is in need of water. The sprinkler could then check the local weather forecast and see no rain projected for the upcoming week. It would then calculate and dispense a measured amount of water.

This EiA system is an architecture for agent wrappers that enable interoperation among things. It is targeted toward what is being referred to as "The Internet of Things," when ordinary real-world things can have URLs, be found by search engines, can send and receive messages, and can sense, react, and reason [1]. With smaller and less expensive communications devices, processors, memories, and sensors, a world where things communicate with humans and with each other is a foregone conclusion. This is the EiA vision.

1.4 Introduction to E2

In pursuit of this vision, the EiA team is developing a plug-in architecture to facilitate rapid agent construction [2]. The architecture is based on an existing software framework called Eclipse [3]. The basic idea is to provide a platform whose main functionality is to accept new software component and capability plug-ins and services. So, if every agent is equipped with the E2 backbone, new capabilities can be installed or removed at any time. For example, suppose there is a device somewhere with both a means of sending and receiving data, and a running version of the E2 platform. An accounting service could then be plugged into this device, effectively making it an "accountant agent," capable of helping with taxes, keeping track of bank accounts, and offering information about the stock exchange. If, later on, a "calculus tutor agent" is needed instead, the new software and information needed to teach derivatives could be sent to the agent, and the accounting service could be removed.

The power of the system is in its versatility. Instead of developing a wide variety of machines intended for a single purpose, E2 offers the ability to adapt any existing machine to suit new needs. New capabilities can be added or removed from a single agent, making it an "all in one" package.

1.5 Introduction to the AmigoBots

The Department of Computer Science and Computer Engineering owns several mobile robots called AmigoBots [4]. These robots come equipped with several gadgets and lots of functionality. They have gripper claws in front, allowing them to pick up small objects like ping-pong balls. They also come standard with wireless Internet adaptors, so they can send and receive information while moving and running routines. They also come with extensive software libraries. These libraries allow the AmigoBots to perform such functions as navigating rooms, detecting obstacles, and keeping a record of what their surrounding terrain is like. These functions are expressed in the C++ programming language, but the robots' behavior can be manipulated using languages like Java and Python as well.

Agents in the EiA system use Extensible Markup Language (XML) messages for communicating messages among themselves. Agent A sends an XML-encoded message to Agent B, and Agent B decides whether or not to retrieve, interpret, react to, and respond to the message. Another member of the EiA team, Quang Duong, has developed an XML dialect for communicating with the robots using a predefined Document Type Definition (DTD). This DTD expresses the robots' core functionality in a structured format. So, using XML messages in this robot language, an instruction can be issued to the robot, which will in turn carry out the command. This approach is beneficial in that it establishes a standard template for communication. Other EiA-enabled devices have their own device-specific XML dialects, and so can be controlled using the same means. Since the XML format is employed, commands can also be easily sent across a network by any transport protocol, including email, and the robots (or other devices) that receive these messages can be controlled remotely. Those devices can also send information, control, or query messages to other EiA devices.

2. Speech and Language as an Approach

A main purpose of the EiA project is to provide a means of controlling electronic devices without prior knowledge or experience of the system. Perhaps the easiest method of control that could be established would be speech or language-based. Language is how humans communicate with each other (but so far not with things). It seems reasonable that this would be an easy way to communicate with machines as well. In fact, computation linguistics, including speech recognition and natural language interfaces, has been an active research field for the nearly forty years [5]. Indeed, if such a system could be successfully implemented, it would solve our problem. But unfortunately, after more than three decades, no robust natural language interface systems can satisfactorily meet our requirements.

Existing speech recognition and natural language interface software is not yet general or extensible enough to serve in the required capacity. These programs are usually tailored to a specific purpose. Quite often they only recognize a small set of words and grammar constructions in limited domains, and are hard to port to new applications. Problems with these implementations fit into two categories, both of which arise when the user is unfamiliar with the system. The person could *overshoot* the natural language capabilities of the system, attempting to issue an instruction that is not recognized. Or the user could *undershoot* the natural language capabilities of the system by not being aware of all the possible commands that are permitted. In either case, the interface has failed to achieve its objectives due to lack of user-friendliness.

3. Menu-Based Natural Language Interface

To summarize the list of problems, an intuitive means of controlling machines must be developed. The system must be easy to use for users of all levels of experience and applicable to all types of devices, both real and theoretical (e.g. the "Smart Machines" from the EiA vision). This research project has focused on developing such a solution. The introduction of a MBNLI could serve to eliminate the above problems while being as intuitive as speech recognition and natural language interfaces.

3.1 Format

From the end user viewpoint, the design consists of a series of cascading windows. Using these menus, the user will be taken through a completion dialogue to build up a complete sentence. Each menu will represent a word or phrase; a part of a complete command. Selecting an option or phrase in one menu will cause a new menu to be created dynamically, listing choices for the next portion of the final command (see Figure 1 below).

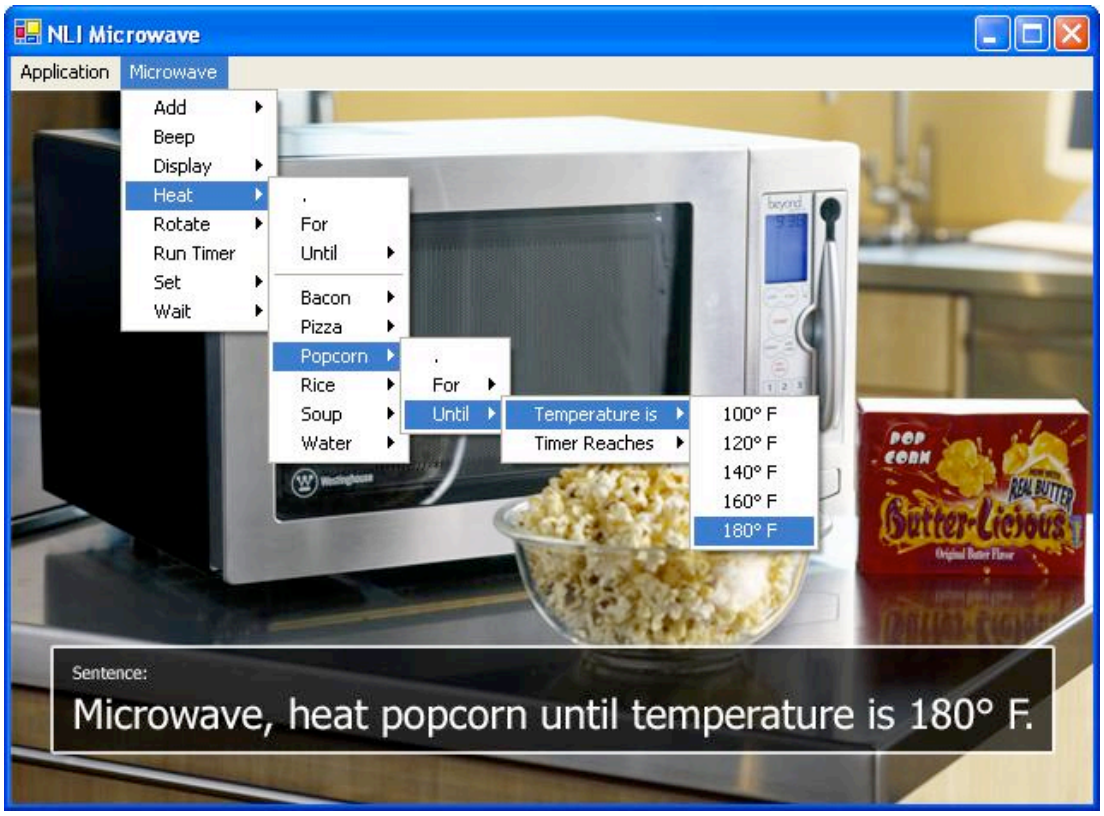

Figure 1 – Example MBNLI format

As an example, consider a MBNLI to establish communication with a picture viewer. Suppose a user wishes to tell the picture viewer to rotate some picture 90°, and zoom in by 50%. Upon entering into the program, the user sees a list of commands: *ROTATE, ZOOM, FLIP, OPEN*, and *CLOSE*. The user selects *ROTATE*. The system displays a list or menu and the user enters the amount to rotate, *90°*. Then another menu displays the choices *CLOCKWISE* or *COUNTER-CLOCKWISE*. The user can either end the command at this point, or select the *AND THEN* extender followed by successive choices *ZOOM - IN - 50%*, to complete the command. The end result would be an instruction to the picture viewer looking something like this: "Viewer, rotate picture 90° clockwise, and then zoom in 50%." After being recognized by the system and translated to a form that the viewer could understand, the command would be executed, and the functionality could be observed. Notice that, because the system is menu-driven, the user cannot state any command that the system will not understand. Also, users can explore the menus to discover commands that they might not have been aware of, so neither will they will undershoot the system's capabilities.

3.2 Components

The MBNLI consists primarily of a parser, a user interface, and a collection of grammars:

- The parser keeps track of progress through a command, prompts further options to complete the command, translates the end sentence into a form that can be understood by the real-world device, and ties the other

components together.

- The user interface is what the client of the system sees. It takes the form of a series of cascading menus depicting what elements are allowed in a command string.
- Eventually, a vast collection of grammars will exist, one for every entity making use of the MBNLI. Each of these grammars should adopt a standard form. The grammars also include the translation language for the given device. So, any single grammar will consist of a structured description of any command that can be issued to its corresponding apparatus, as well as an outline of exactly what that apparatus will receive upon completion of the dialogue (e.g. the actual instruction to be carried out).

3.2.1 Parser

The parser is the center of the MBNLI architecture. It can accept any EiA grammar and coordinate with a variety of user interfaces including cascaded menus. It is designed to be extensible enough that any inputs adhering to a few restrictions can be adopted and utilized.

A parser instance is invoked and initialized with a grammar for some device. Any grammar for any device can be relayed to the parser, assuming it is in the appropriate, predefined form. In EiA, command grammars and target languages (input and output) are expressed as XML documents. Once initialized, a grammar is ready to display initial menu selections to a user via its user interface.

Using what is referred to as a "One-step predictive parser," the system loops. First it displays a collection of choices and the user selects one. Then it predicts the next choices and displays these, and so on until a complete sentence is displayed. As elements of the instruction are selected, they will be fed back into the parser. The parser will then check the grammar to determine what options and phrases are allowed next. A new, dynamically-generated menu will then be created and filled with these new phrases to be presented to the user. The ability to know in advance what syntax will be permitted before input is accepted is what makes this software "predictive."

As the sentence is being created, the parser also keeps track of all the instruction segments that have already been selected. Each of these pieces is stored as a node in a tree. This *parse tree structure* is used at the end of the instruction. The tree structure allows for easy identification of how the individual segments are related to each other via the grammar rules, and in what order elements appear.

Upon completion of the sentence, the parse tree structure generates a translation into a command in a target language. Associated with each grammar rule are fragments of translation into the target language. The parse tree is interpreted to compose the translation fragments together into a command in the target language. Beginning with a simple English sentence, the parser analyzes each portion of the instruction piece by piece, and produces an XML command that can be understood by an EiA agent. This command is now ready to be sent to the agent and carried out.

3.2.2 User Interface

A user interface (UI) is the part of a system that a user sees and how the user communicates with the system. The MBNLI user will see a series of cascading menus. Each menu will be populated by any phrase that can contribute to the end command at that time (see Figure 1 above). The system implements a guided completion dialogue to build up an instruction string piece by piece. The goal of the setup is to retain the ease of use of natural language interfaces while avoiding the problems described in Section 2. A MBNLI interface could provide an ideal means of communication with machines.

The UI is closely tied to the parser. It receives all of its information from that source. The UI itself will have no means of intelligence or problem solving. The options displayed in each menu will be fed in directly from the parser. The UI will merely display the information it receives. Upon selection of a new item, the parser needs not do anything other than relay the new phrase that has been chosen. The sole purpose of the interface is to display information to the user. The system has been deliberately designed to render the UI separable so that it can be implemented in a variety of ways. This design contributes to the overall extensibility of the system.

Although the MBNLI cascaded menu UI is the main one described, such an interface is by no means required. For instance, if a user becomes skilled in dealing with a particular device, it might be quicker to bypass the menus and simply type in a command. Other alternative user interfaces could be "placed on top" of the rest of the software. Options such as text input, speech, and a series of buttons, although perhaps not ideal for some situations, could just as easily be implemented in place of the cascaded menu UI.

Speech, in particular, is interesting. Another student, Micah Byers, is developing a speech interface to MBNLI. His E2 speech plug-in uses the cascaded menus to display choices to the user. These choices are also provided to a Via Voice speech module. When the user speaks a selection, the E2 speech plug-in identifies the choice selected and returns that to the parser. Thus speech is an optional modular add-in to the MBNLI system.

3.2.3 Grammars

Grammars are what define exactly what languages (e.g., command or queries) are accepted and understood by a parser. The grammars that we are dealing with consist of two parts. The first is the definition of what commands are valid when communicating with the corresponding device. The second part is the target language command definitions; that is, the list of machine commands that will actually be interpreted and carried out. This part of the grammar, the target language, is highly variable. Such a language could take almost any form. The creators of the device could interface to it using binary, assembly instructions, a higher level language such as C or Java, or some other abstraction that merely describes the desired functionality, and later on undergoes yet another transformation into executable code. Anything the designers of the device have in mind could be specified as a target language. Although each entity will still possess its own target language, the scenario has been simplified for dealing with a test bed of theoretical agents by structuring all output as XML files.

MBNLI can use variants of Context Free Grammars (CFGs). A CFG rule consists of a rule name and a

collection of non-terminals (other rule names) or terminals (words or phrases from the language being recognized) [6]. Tracing through a variable's definition eventually yields a list of terminals, or a list of items that can be broken down no farther (see Appendix B). Once a grammar has been established in this form, it can easily be specified in an XML format. The collection of grammar rules constitutes a grammar and is the input to the parser.

MBNLI grammars are formatted in XML (see Appendix C). Furthermore, the grammar must extend a predefined template (see Appendix A). However, once these initial conditions are met, the grammar author can define a wide variety of languages, one for each device or family of devices. Defining a grammar that is both efficient and intuitive is something of an art form, requiring human factors, testing, and refinement to improve usability. Meeting the goal of providing a user-friendly form of communication with electronic devices is largely up to the developer of these languages.

4. Controlling Robots

To illustrate the idea of MBNLI interfaces to control devices, a MBNLI grammar targeted on the AmigoBot command set was constructed with the following goals. First, it should show the ease-of-use of a correctly implemented MBNLI system. Being designed as a part of the EiA Agent team, it was also intended to exhibit an E2 plug-in service in action, as well as to show how agents could be applied to the real world. The ultimate ambition was for the project to provide not only a realistic demonstration for a single real world problem, but also to illustrate a path that could be extended to deal with natural language communications on a global scale.

4.1 Design

The scope of the EiA project goes far beyond an interface to a robot. A number of individuals worked on many aspects of the project, including the E2 plug-in architecture, a means of data exchange across a network, and security issues arising when not all agents are privy to the same information. A variety of individual tasks all contribute to the EiA system as it exists. But for the purposes of this thesis, only the MBNLI plug-in design will be described.

First of all, the design required a method by which commands could be passed from the MBNLI plug-in to the robots. The EiA system had adopted XML as a standard method of passing information across a network. Therefore, an XML interface to the robots was required to fit this structure. Quang Duong, an undergraduate member of the EiA team, developed a DTD defining how instructions could be passed from a remote source to the robots for execution. Any XML document meeting the requirements of the provided DTD could express some functionality for the robots to carry out. So, this DTD effectively became the target translation language for the robot grammar.

This, then, left the MBNLI command grammar to be created. The first stage of this grammar design consisted of only English sentences. Every command that could be sent to the robots, including modifiers and

compound instructions, was taken into consideration. After identifying all potential instructions, the sentences were re-structured as a context free grammar (CFG). This step groups instructions into rules, instead of individual statements. CFG rules are of the form LHS → RHS, where the left hand side (LHS) consists of a non-terminal name of a rule and the right hand side (RHS) consists of one or more non-terminals (the names of grammar rules) or terminals (words or phrases in the language being defined). A CFG allows for a vast number of commands to be expressed by just a few rules. This format also allows for easy translation into the default form of an XML document (which follows similar rules and structuring).

The user interface was not a major concern in this project because such an interface had already been developed for an earlier prototype of a MBNLI [7].

The parser for this kind of system also exists in some form [7]. It was assumed early on that this parser, with only minimal modifications, would serve as the backbone for the MBNLI. However, the software, upon analysis, proved not to be extensible enough to suit our purposes. A new parser has been designed, compatible with the E2 architecture. The approach is to design everything about the parser modularly, and then simply connect all of these modules together. Separating the grammars, the interface, the parse tree, and any other contributor to the system allows for easy maintenance and updates for the future. This methodology allows some aspects of the configuration to be modified without disrupting functionality of any of the other segments.

4.2 Initial Implementation

The initial implementation of the AmigoBot MBNLI system met with limited success and was only partially functional. The robot interface was implemented and demonstrated. Furthermore, a grammar for the robots was fully developed. However, the parser was never adequately constructed. So— a command language does exist. Instructions following the guidelines of this language could theoretically be contrived from the translation definitions provided. And these instructions could be given to the robots, which would be capable of carrying them out. But transitioning from each of these steps is not yet possible. Without a functioning parser, not only can a user not traverse through the dialogue to create a command string, but that string can not be translated into the target language corresponding to the existing DTD. So, while each aspect of the system has been manufactured, the pieces of this puzzle are not yet fitting together.

4.3 Analysis

The first iteration of the MBNLI met with only limited success. A methodical means of communicating with the AmigoBots was completed. And an intuitive means of controlling these robots was developed. These components met the goals of being straightforward and easy to modify. However, the overall project fell short of success. The original parser was in need of an update, so the individual pieces of the system remained disjoint. Most of the necessary components had successfully been implemented, but the sum entity could demonstrate no

functionality.

5. Universal Approach

A second generation of the project then began. The vision at this point shifted away from a single example (i.e. the robot demo), and instead focused on delivering a universal system of control. The project envisioned an entire house or business place of agents. For example, a household could have agents controlling the TV, stove, lights, thermostat, answering machine, and any other appliance. To realistically apply natural language to all of these devices, the same backbone would have to hold constant for each particular instance. The previously described modular approach was stressed more than ever. Each component of the system was completely re-developed to ensure extensibility and scalability. The parser and the UI were constructed from scratch. Furthermore, an entire collection of grammars was created to serve as a test bed for the system.

This stage of the project was handled by a new team of students. Hieu Nguyen was put in charge of developing the new parser. Edmundo Ruiz was in charge of the user interface. Micah Byers, as mentioned above, was to develop a speech recognition service to augment the interface. It became my task, then, to construct the new device grammars. Each of these grammars (in CFG and XML formats), as well as definition templates are provided at the conclusion of this thesis (see Appendices, sections A, B, and C).

In the same way that the robot grammar was devised, so were grammars for a variety of household devices. Holding true to the theme of scalability, each grammar extended a basic definition. The standard form for a command was established as…

Instruction => DEVICE command

This means that a valid instruction begins by specifying what it is interfacing to, and then specifies the command to be executed. The command variable must then be extended for each entity to define what contributes to a legal instruction. An additional structure for styling the grammar is provided and encouraged in order to foster convention, but any grammar that adheres to the above rule is considered legitimate.

6. Conclusion

Natural language interfaces could very well become pervasive in the computing world in years to come. Designed correctly, these interfaces might become the easiest way for people to communicate with machines. A MBNLI is particularly attractive because it bypasses the pitfalls that have historically hampered free-form natural language recognition. Such a system becomes even more appealing when designed to provide a universal interface to virtually any electronic device. The system described here could be easily adopted to interface to anything that provides a sufficient grammar. Such an implementation could prove to be an ideal solution applicable to many real world scenarios.

These EiA and MBNLI projects are still under development. The design promises scalability, however

this design has not come to a realization yet. One major problem is that few agents, as expressed in the EiA vision, actually exist yet. The project envisions household objects capable of accepting and sending messages, and even addresses agents capable of sensing their surroundings and responding to certain environmental triggers. Although these sorts of machines are wholly feasible, and might even come to realization in the near future, they nevertheless are in early stages of development. The EiA infrastructure, however, is beginning to provide a route for their development.

The real strength of this project is in demonstrating what kinds of applications could be implemented. The current version of the project offers a reasonable and efficient method of controlling appliances, even if those appliances are yet to be assembled. A growing test bed of theoretical agents is in place; it is simply a matter of time before the industrial world puts these agents to use. The Internet of Things could very well be just beyond the horizon.

References

[1] Gershenfeld, Neil, Krikorian, Raffi, and Cohen, Danny. "Internet 0: Interdevice Internetworking." Scientific American Oct. 2004: 76-81.

[2] Vu, Minh and Thompson, Craig. "E2 Agent Plugin Architecture." IEEE Integration of Knowledge Intensive Multi-Agent Systems (KIMAS-05), Waltham, MA, April 18-21 2005.

[3] Object Technology International, Inc. "Eclipse Platform Technical Overview." February 2003. <http://www.eclipse.org/articles/index.html>.

[4] ActivMedia Robotics, LLC. "AmigoBot Technical Manual." 2002. <http://www.amigobot.com>.

[5] Weizenbaum, J. 1966. ELIZA--A Computer Program for the Study of Natural Language Communication Between Man and Machine. Communications of the ACM, 9 (1): 36-45.

[6] Hopcroft, John E., Motwani, Rajeev, and Ullman, Jeffrey D. Introduction to Automata Theory, Languages, and Computation. 2nd ed. New York: Addison Wesley, 2001.

[7] Pazandak, Paul, and Thompson, Craig. "AgentGram: Natural Language Interface for Agents." June 2002. <http://www.objs.com/agility/final/AgentGram/AGENTGRAM-PROJECT-SUMMARY.html>.

Appendices

A. Grammar Templates

Context-Free Grammar Definition

```
==================================
========= DEVICE GRAMMAR =========
==================================

Instruction => DEVICE command

//That's it.  Enforcing more structure = reducing extensibility

==================================
=== ENCOURAGED COMMAND STRUCTURE ==
```

==========

//Although not required, grammars are encouraged to adopt this form (or something very similar)

```
command => simple
        | complex
        | compound

compound => simple THEN command
        | complex THEN command

simple => action

complex => function UNTIL ...
            | function IF ...
            | function FOR ...
            | ...

action => ...
           | ...
           | ...
           | shared

shared => ...
           | ...
           | ...

function => shared
            | ...
            | ...
            | ...
```

/*This structure will keep the intuitiveness of the MBNL system, while still allowing for levels of abstraction and efficient grammars.*/

XML Grammar Document Type Definition

```
<?xml version="1.0"?>
<!DOCTYPE GrammarSpec[
   <!ELEMENT Grammar (START, (production)+)>
   <!ELEMENT START (#PCDATA)>
   <!ELEMENT Production (Rule*)>
     <!ATTLIST Production Symbol CDATA #REQUIRED>
   <!ELEMENT Rule ( ((variable)*, (TERMINAL)*) +)>
   <!ELEMENT variable (#PCDATA)>
   <!ELEMENT TERMINAL (#PCDATA)>
     <!ATTLIST TERMINAL Type (enum, integer, real, data, string) #IMPLIED>
     <!-- the DATA type will catch any types that aren't specified -->
]>
```

B. Context-Free Grammars

Microwave Grammar

/*Grammar for Microwave controller*/

/*Example: Set power to max, then heat for 2 minutes and 30 seconds, then wait for 1 minute, then heat and rotate disk for 2 minutes, then wait until temperature is 100, then beep for 5 seconds.*/

```
command =>  simple
           |  complex
           |  compound

compound => simple THEN command
           |  complex THEN command

simple => action
```

```
complex => function UNTIL event
          | function FOR time

==========================================
action => SET POWER TO power
          | SET TIMER TO time
          | ADD ONE MINUTE
          | HEAT food_item
          | STOP HEATING
          | ROTATE DISK 90 DEGREES
          | DISPLAY setting
          | shared

shared => HEAT
          | RUN TIMER
          | ROTATE DISK
          | HEAT AND ROTATE DISK
          | BEEP

function => shared
           | WAIT
==========================================

power => MIN | WEAK | MEDIUM | STRONG | MAX

temperature => 100 | 120 | 140 | 160 | 180

setting => CURRENT POWER
          | CURRENT TEMPERATURE

food_item => SOUP |  WATER | POPCORN | RICE | BACON | PIZZA

event => TEMPERATURE IS temperature
     | TIMER REACHES time

time => INTEGER SECONDS
    | INTEGER MINUTES
    | INTEGER MINUTES AND INTEGER SECONDS
```

Picture Viewer Grammar

```
/*Grammar for Picture Viewer controller*/

/*Example: Rotate right 90.0 degrees.*/

command => file
          | action

file => OPEN filename
    | CLOSE filename
    | HIDE filename
    | SHOW filename
    | SELECT filename AS WORKSPACE

action => ZOOM zoomDirection percentage
          | ROTATE rotateDirection degrees
          | FLIP flipDirection
          | RESIZE method
          | VIEW changeDirection PICTURE

filename => STRING extension

extension => .JPG | .BMP | .GIF

zoomDirection => IN | OUT

percentage => FLOAT%
```

```
rotateDirection => LEFT | RIGHT

degrees => FLOAT DEGREES

flipDirection => HORIZONTALLY | VERTICALLY

method => percentage
        | coordinates

coordinates => INTEGER X, INTEGER Y

changeDirection => NEXT | PREVIOUS
```

Stereo Grammar

```
/*Grammar for Stereo controller*/

/*Example: Set station to 104.9.*/

command          => generic
          | radio
          | disk

generic => TURN toggle
          | SELECT device
          | ADJUST VOLUME volume

radio => SET STATION TO frequency
     | SCAN direction
     | STOP SCANNING

disk => PLAY
    | PAUSE
    | FAST FORWARD
    | REWIND
    | cd_command

cd_command => SKIP TO skipDirection
        | TURN ESP toggle

toggle => ON | OFF

device => RADIO | CASETTE PLAYER | CD PLAYER

volume => LOUDER
          | SOFTER
          | INTEGER

frequency => FLOAT

direction => UP | DOWN

skipDirection => NEXT | PREVIOUS
```

Thermostat Grammar

```
/*Grammar for Thermostat controller*/

/*Example: On saturday and sunday at 9:30 pm, set temperature in the upstairs bedroom and downstairs bedroom
to 68 degrees.*/

/*note- definitions must be provided for enumerations: DAY_OF_WEEK, HOUR, MINUTE, and AMPM*/


command => simple
          | complex
```

```
simple => action

complex => immediate
         | routine

action => DISPLAY SETTINGS
        | ERASE SETTINGS

immediate => ADJUST TEMPERATURE IN THE rooms TO temp_control

routine => ON days AT time SET TEMPERATURE IN THE rooms TO temp

rooms => ENTIRE HOUSE
    | room_list

room_list => room
           | room AND room_list

room   => LIVING ROOM
        | DINING ROOM
        | KITCHEN
        | DOWNSTAIRS BEDROOM
        | DOWNSTAIRS BATHROOM
        | UPSTAIRS BEDROOM
        | UPSTAIRS BATHROOM
        | BASEMENT

days => EACH DAY
   |      day_list

day_list => DAY_OF_WEEK
          | DAY_OF_WEEK AND day_list

time => HOUR:MINUTE AMPM

temp_control => temp
              | alteration

temp => INTEGER DEGREES

alteration => BE variation BY INTEGER DEGREES

variation => COOLER
           | WARMER
```

C. XML-Formatted Grammars

Microwave Grammar

```
<Grammar>
  <START>instruction</START>

  <Production Symbol="instruction">
    <Rule>
        <TERMINAL>MICROWAVE</TERMINAL>
        <variable>command</variable>
    </Rule>
  </Production>

  <Production Symbol="command">
    <Rule>
        <variable>simple</variable>
    </Rule>

    <Rule>
        <variable>complex</variable>
```

```
 </Rule>

 <Rule>
        <variable>compound</variable>
 </Rule>
</Production>

<Production Symbol="compound">
 <Rule>
        <variable>simple</variable>
        <TERMINAL>THEN</TERMINAL>
        <variable>command</variable>
 </Rule>

 <Rule>
        <variable>complex</variable>
        <TERMINAL>THEN</TERMINAL>
        <variable>command</variable>
 </Rule>
</Production>

<Production Symbol="simple">
 <Rule>
        <variable>action</variable>
 </Rule>
</Production>

<Production Symbol="complex">
 <Rule>
        <variable>function</variable>
        <TERMINAL>UNTIL</TERMINAL>
        <variable>event</variable>
 </Rule>

 <Rule>
        <variable>function</variable>
        <TERMINAL>FOR</TERMINAL>
        <variable>time</variable>
 </Rule>
</Production>

<Production Symbol="action">
 <Rule>
        <TERMINAL>SET POWER TO</TERMINAL>
        <variable>power</variable>
 </Rule>

 <Rule>
        <TERMINAL>SET TIMER TO</TERMINAL>
        <variable>time</variable>
 </Rule>

 <Rule>
        <TERMINAL>ADD ONE MINUTE</TERMINAL>
 </Rule>

 <Rule>
        <TERMINAL>HEAT</TERMINAL>
        <variable>food_item</variable>
 </Rule>

 <Rule>
        <TERMINAL>STOP HEATING</TERMINAL>
 </Rule>

 <Rule>
        <TERMINAL>ROTATE DISK 90 DEGREES</TERMINAL>
 </Rule>
```

```
 <Rule>
      <TERMINAL>DISPLAY</TERMINAL>
      <variable>setting</variable>
 </Rule>

 <Rule>
      <variable>shared</variable>
 </Rule>
</Production>

<Production Symbol="shared">
 <Rule>
      <TERMINAL>HEAT</TERMINAL>
 </Rule>

 <Rule>
      <TERMINAL>RUN TIMER</TERMINAL>
 </Rule>

 <Rule>
      <TERMINAL>ROTATE DISK</TERMINAL>
 </Rule>

 <Rule>
      <TERMINAL>HEAT AND ROTATE DISK</TERMINAL>
 </Rule>

 <Rule>
      <TERMINAL>BEEP</TERMINAL>
 </Rule>
</Production>

<Production Symbol="function">
 <Rule>
      <variable>shared</variable>
 <Rule>

 <Rule>
      <TERMINAL>WAIT</TERMINAL>
 </Rule>
</Production>

<Production Symbol="power">
 <Rule>
      <TERMINAL>MIN</TERMINAL>
 </Rule>

 <Rule>
      <TERMINAL>WEAK</TERMINAL>
 </Rule>

 <Rule>
      <TERMINAL>MEDIUM</TERMINAL>
 </Rule>

 <Rule>
      <TERMINAL>STRONG</TERMINAL>
 </Rule>

 <Rule>
      <TERMINAL>MAX</TERMINAL>
 </Rule>
</Production>

<Production Symbol="temperature">
 <Rule>
      <TERMINAL Type="integer">TEMP_SELECTION</TERMINAL>
```

```
  </Rule>
</Production>

<Production Symbol="setting">
  <Rule>
        <TERMINAL>CURRENT POWER</TERMINAL>
  </Rule>

  <Rule>
        <TERMINAL>CURRENT TEMPERATURE</TERMINAL>
  </Rule>
</Production>

<Production Symbol="food_item">
  <Rule>
        <TERMINAL>SOUP</TERMINAL>
  </Rule>

  <Rule>
        <TERMINAL>WATER</TERMINAL>
  </Rule>

  <Rule>
        <TERMINAL>POPCORN</TERMINAL>
  </Rule>

  <Rule>
        <TERMINAL>RICE</TERMINAL>
  </Rule>

  <Rule>
        <TERMINAL>BACON</TERMINAL>
  </Rule>

  <Rule>
        <TERMINAL>PIZZA</TERMINAL>
  </Rule>
</Production>

<Production Symbol="event">
  <Rule>
        <TERMINAL>TEMPERATURE</TERMINAL>
        <TERMINAL>IS</TERMINAL>
        <variable>temperature</variable>
  </Rule>

  <Rule>
        <TERMINAL>TIMER</TERMINAL>
        <TERMINAL>REACHES</TERMINAL>
        <variable>time</variable>
  </Rule>
</Production>

<Production Symbol="time">
  <Rule>
        <TERMINAL Type="integer">INTEGER</TERMINAL>
        <TERMINAL>SECONDS</TERMINAL>
  </Rule>

  <Rule>
        <TERMINAL Type="integer">INTEGER</TERMINAL>
        <TERMINAL>MINUTES</TERMINAL>
  </Rule>

  <Rule>
        <TERMINAL Type="integer">INTEGER</TERMINAL>
        <TERMINAL>MINUTES</TERMINAL>
        <TERMINAL>AND</TERMINAL>
```

```
        <TERMINAL Type="integer">INTEGER</TERMINAL>
        <TERMINAL>SECONDS</TERMINAL>
    </Rule>
  </Production>
 </Grammar>
```

Picture Viewer Grammar

```
 <Grammar>
  <START>instruction</START>

  <Production Symbol="instruction">
   <Rule>
        <TERMINAL>PICTURE VIEWER</TERMINAL>
        <variable>command</variable>
   </Rule>
  </Production>

  <Production Symbol="command">
   <Rule>
        <variable>file</variable>
   </Rule>

   <Rule>
        <variable>action</variable>
   </Rule>
  </Production>

  <Production Symbol="file">
   <Rule>
        <TERMINAL>OPEN</TERMINAL>
        <variable>filename</variable>
   </Rule>

   <Rule>
        <TERMINAL>CLOSE</TERMINAL>
        <variable>filename</variable>
   </Rule>

   <Rule>
        <TERMINAL>HIDE</TERMINAL>
        <variable>filename</variable>
   </Rule>

   <Rule>
        <TERMINAL>SHOW</TERMINAL>
        <variable>filename</variable>
   </Rule>

   <Rule>
        <TERMINAL>SELECT</TERMINAL>
        <variable>filename</variable>
        <TERMINAL>AS WORKSPACE</TERMINAL>
   </Rule>
  </Production>

  <Production Symbol="action">
   <Rule>
        <TERMINAL>ZOOM</TERMINAL>
        <variable>zoomDirection</variable>
        <variable>percentage</variable>
   </Rule>

   <Rule>
        <TERMINAL>ROTATE</TERMINAL>
        <variable>rotateDirection</variable>
        <variable>degrees</variable>
   </Rule>
```

```
 <Rule>
        <TERMINAL>FLIP</TERMINAL>
        <variable>flipDirection</variable>
 </Rule>

 <Rule>
        <TERMINAL>RESIZE</TERMINAL>
        <variable>method</variable>
 </Rule>

 <Rule>
        <TERMINAL>VIEW</TERMINAL>
        <variable>changeDirection</variable>
        <TERMINAL>PICTURE</TERMINAL>
 </Rule>
</Production>

<Production Symbol="filename">
 <Rule>
        <TERMINAL Type="string">STRING</TERMINAL>
        <variable>extension</variable>
 </Rule>
</Production>

<Production Symbol="extension">
 <Rule>
        <TERMINAL>.JPG</TERMINAL>
 </Rule>

 <Rule>
        <TERMINAL>.BMP</TERMINAL>
 </Rule>

 <Rule>
        <TERMINAL>.GIF</TERMINAL>
 </Rule>
</Production>

<Production Symbol="zoomDirection">
 <Rule>
        <TERMINAL>IN</TERMINAL>
 </Rule>

 <Rule>
        <TERMINAL>OUT</TERMINAL>
 </Rule>
</Production>

<Production Symbol="percentage">
 <Rule>
        <TERMINAL Type="real">FLOAT</TERMINAL>
        <TERMINAL>%</TERMINAL>
 </Rule>
</Production>

<Production Symbol="rotateDirection">
 <Rule>
        <TERMINAL>LEFT</TERMINAL>
 </Rule>

 <Rule>
        <TERMINAL>RIGHT</TERMINAL>
 </Rule>
</Production>

<Production Symbol="degrees">
 <Rule>
        <TERMINAL Type="real">FLOAT</TERMINAL>
```

```
        <TERMINAL>DEGREES</TERMINAL>
    </Rule>
  </Production>

  <Production Symbol="flipDirection">
    <Rule>
        <TERMINAL>HORIZONTALLY</TERMINAL>
    </Rule>

    <Rule>
        <TERMINAL>VERTICALLY</TERMINAL>
    </Rule>
  </Production>

  <Production Symbol="method">
    <Rule>
        <variable>percentage</variable>
    <Rule>

    <Rule>
        <variable>coordinates</variable>
    </Rule>
  </Production>

  <Production Symbol="coordinates">
    <Rule>
        <TERMINAL Type="integer">INTEGER</TERMINAL>
        <TERMINAL>X, </TERMINAL>
        <TERMINAL Type="integer">INTEGER</TERMINAL>
        <TERMINAL>Y</TERMINAL>
    </Rule>
  </Production>

  <Production Symbol="changeDirection">
    <Rule>
        <TERMINAL>NEXT</TERMINAL>
    </Rule>

    <Rule>
        <TERMINAL>PREVIOUS</TERMINAL>
    </Rule>
  </Production>
</Grammar>
```

Stereo Grammar

```
<Grammar>
  <START>instruction</START>

  <Production Symbol="instruction">
    <Rule>
        <TERMINAL>STEREO</TERMINAL>
        <variable>command</variable>
    </Rule>
  </Production>

  <Production Symbol="command">
    <Rule>
      <variable>generic</variable>
    </Rule>

    <Rule>
        <variable>radio</variable>
    </Rule>

    <Rule>
        <variable>disk</variable>
    </Rule>
  </Production>
```

```
<Production Symbol="generic">
 <Rule>
       <TERMINAL>TURN</TERMINAL>
       <variable>toggle</variable>
 </Rule>

 <Rule>
       <TERMINAL>SELECT</TERMINAL>
       <variable>device</variable>
 </Rule>

 <Rule>
       <TERMINAL>ADJUST VOLUME</TERMINAL>
       <variable>volume</variable>
 </Rule>
</Production>

<Production Symbol="radio">
 <Rule>
       <TERMINAL>SET STATION TO</TERMINAL>
       <variable>frequency</variable>
 </Rule>

 <Rule>
       <TERMINAL>SCAN</TERMINAL>
       <variable>direction</variable>
 </Rule>

 <Rule>
       <TERMINAL>STOP SCANNING</TERMINAL>
 </Rule>
</Production>

<Production Symbol="disk">
 <Rule>
       <TERMINAL>PLAY</TERMINAL>
 </Rule>

 <Rule>
       <TERMINAL>PAUSE</TERMINAL>
 </Rule>

 <Rule>
       <TERMINAL>FAST FORWARD</TERMINAL>
 </Rule>

 <Rule>
       <TERMINAL>REWIND</TERMINAL>
 </Rule>

 <Rule>
       <variable>cd_command</variable>
 </Rule>
</Production>

<Production Symbol="cd_command">
 <Rule>
       <TERMINAL>SKIP TO</TERMINAL>
       <variable>skipDirection</variable>
 </Rule>

 <Rule>
       <TERMINAL>TURN ESP</TERMINAL>
       <variable>toggle</variable>
 </Rule>
</Production>
```

```
<Production Symbol="toggle">
 <Rule>
      <TERMINAL>ON</TERMINAL>
 </Rule>

 <Rule>
      <TERMINAL>OFF</TERMINAL>
 </Rule>
</Production>

<Production Symbol="device">
 <Rule>
      <TERMINAL>RADIO</TERMINAL>
 </Rule>

 <Rule>
      <TERMINAL>CASETTE PLAYER</TERMINAL>
 </Rule>

 <Rule>
      <TERMINAL>CD PLAYER</TERMINAL>
 </Rule>

<Production Symbol="volume">
 <Rule>
      <TERMINAL>LOUDER</TERMINAL>
 </Rule>

 <Rule>
      <TERMINAL>SOFTER</TERMINAL>
 </Rule>

 <Rule>
      <TERMINAL Type="integer">INTEGER</TERMINAL>
 </Rule>
</Production>

<Production Symbol="frequency">
 <Rule>
      <TERMINAL Type="real">FLOAT</TERMINAL>
 </Rule>
</Production>

<Production Symbol="direction">
 <Rule>
      <TERMINAL>UP</TERMINAL>
 <Rule>

 <Rule>
      <TERMINAL>DOWN</TERMINAL>
 </Rule>
</Production>

<Production Symbol="skipDirection">
 <Rule>
      <TERMINAL>NEXT</TERMINAL>
      <TERMINAL>PREVIOUS</TERMINAL>
 </Rule>
</Production>
</Grammar>
```

Thermostat Grammar

```
<Grammar>
 <START>instruction</START>

 <Production Symbol="instruction">
  <Rule>
   <TERMINAL>MICROWAVE</TERMINAL>
```

```
    <variable>command</variable>
  </Rule>
</Production>

<Production Symbol="command">
  <Rule>
        <variable>simple</variable>
  </Rule>

  <Rule>
        <variable>complex</variable>
  </Rule>
</Production>

<Production Symbol="simple">
  <Rule>
        <variable>action</variable>
  </Rule>
</Production>

<Production Symbol="complex">
  <Rule>
        <variable>immediate</variable>
        <variable>routine</variable>
  </Rule>
</Production>

<Production Symbol="action">
  <Rule>
        <TERMINAL>DISPLAY SETTINGS</TERMINAL>
  </Rule>

  <Rule>
        <TERMINAL>ERASE SETTINGS</TERMINAL>
  </Rule>
</Production>

<Production Symbol="immediate">
  <Rule>
        <TERMINAL>ADJUST TEMPERATURE IN THE</TERMINAL>
        <variable>rooms</variable>
        <TERMINAL>TO</TERMINAL>
        <variable>temp_control</variable>
  </Rule>
</Production>

<Production Symbol="routine">
  <Rule>
        <TERMINAL>ON</TERMINAL>
        <variable>days</variable>
        <TERMINAL>AT</TERMINAL>
        <variable>time</variable>
        <TERMINAL>SET TEMPERATURE IN THE</TERMINAL>
        <variable>rooms</variable>
        <TERMINAL>TO</TERMINAL>
        <variable>temp</variable>
  </Rule>
</Production>

<Production Symbol="rooms">
  <Rule>
        <TERMINAL>ENTIRE HOUSE</TERMINAL>
  </Rule>

  <Rule>
        <variable>room_list</variable>
  </Rule>
</Production>
```

```
<Production Symbol="room_list">
 <Rule>
       <variable>room</variable>
 </Rule>

 <Rule>
       <variable>room</variable>
       <TERMINAL>AND</TERMINAL>
       <variable>room_list</variable>
 </Rule>
</Production>

<Production Symbol="room">
 <Rule>
       <TERMINAL>LIVING ROOM</TERMINAL>
 </Rule>
 <Rule>
       <TERMINAL>DINING ROOM</TERMINAL>
 </Rule>
 <Rule>
       <TERMINAL>KITCHEN</TERMINAL>
 </Rule>
 <Rule>
       <TERMINAL>DOWNSTAIRS BEDROOM</TERMINAL>
 </Rule>
 <Rule>
       <TERMINAL>DOWNSTAIRS BATHROOM</TERMINAL>
 </Rule>
 <Rule>
       <TERMINAL>UPSTAIRS BEDROOM</TERMINAL>
 </Rule>
 <Rule>
       <TERMINAL>UPSTAIRS BATHROOM</TERMINAL>
 </Rule>
 <Rule>
       <TERMINAL>BASEMENT</TERMINAL>
 </Rule>
</Production>

<Production Symbol="days">
 <Rule>
       <TERMINAL>EACH DAY</TERMINAL>
 </Rule>

 <Rule>
       <variable>day_list</variable>
 </Rule>
</Production>

<Production Symbol="day_list">
 <Rule>
       <TERMINAL Type="enum">DAY_OF_WEEK</TERMINAL>
 </Rule>

 <Rule>
       <TERMINAL Type="enum">DAY_OF_WEEK</TERMINAL>
       <TERMINAL>AND</TERMINAL>
       <variable>day_list</variable>
 </Rule>
</Production>

<Production Symbol="time">
 <Rule>
       <TERMINAL Type="enum">HOUR</TERMINAL>
       <TERMINAL>:</TERMINAL>
       <TERMINAL Type="enum">MINUTE</TERMINAL>
       <TERMINAL Type="enum">AMPM</TERMINAL>
```

```
  </Rule>
 </Production>

 <Production Symbol="temp_control">
  <Rule>
        <variable>temp</variable>
  </Rule>

  <Rule>
        <variable>alteration</variable>
  </Rule>
 </Production>

 <Production Symbol="temp">
  <Rule>
        <TERMINAL Type="integer">INTEGER</TERMINAL>
        <TERMINAL>DEGREES</TERMINAL>
  </Rule>
 </Production>

 <Production Symbol="alteration">
  <Rule>
        <TERMINAL>BE</TERMINAL>
        <variable>variation</variable>
        <TERMINAL>BY</TERMINAL>
        <TERMINAL Type="integer">INTEGER</TERMINAL>
        <TERMINAL>DEGREES</TERMINAL>
  </Rule>
 </Production>

 <Production Symbol="variation">
  <Rule>
        <TERMINAL>COOLER</TERMINAL>
  </Rule>

  <Rule>
        <TERMINAL>WARMER</TERMINAL>
  </Rule>
 </Production>
</Grammar>
```