Situating a Natural Language System in a Multimodal Environment

by

Katherine E. Koch

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

May 17, 2001

Author_____
Department of Electrical Engineering and Computer Science
May 17, 2001

Certified by_____
Howard E. Shrobe
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Situating a Natural Language System in a Multimodal Environment

by

Katherine E. Koch

Submitted to the
Department of Electrical Engineering and Computer Science

May 17, 2001

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

Abstract

In a human-centered computing environment, users interact with their computers in a natural way, including use of speech and gestures. In order for the computers to understand the user's commands, sophisticated natural language processing is needed. Natural language systems capable of analyzing sentences and answering questions have been developed, but have typically been confined to computer terminals. By situating a natural language system in a perceptually enabled environment, we begin to address the need for better natural language processing. This project uses research technologies developed at MIT's Artificial Intelligence Laboratory, the START Natural Language Question Answering System and the Intelligent Room, to demonstrate how a natural language system can be embedded in a multimodal environment.

Thesis Supervisor: Howard E. Shrobe
Title: Principle Research Scientist

ACKNOWLEDGEMENTS

I would like to thank Dr. Howard Shrobe for his guidance on this project, Dr. Boris Katz and Sue Felshin for their assistance with START, and Kalman Reti for his help with JLinker. Thanks to Andy, Steven, Krzysztof and the rest of the Intelligent Room gang for answering my questions and keeping me entertained.

Most especially, I thank my parents for their support and for making my MIT education possible, and I thank Dave for his friendship, patience, and encouragement.

TABLE OF CONTENTS

1 INTRODUCTION

Computers are in nearly every home and office in America and computing power is doubling every eighteen months. Why then are users constantly struggling with the machines on their desks to get them to do what they want? The image of someone huddled in front of his monitor muttering in frustration is certainly not unfamiliar to most computer users. If computers are becoming faster all the time, shouldn't users' productivity be increasing as well? Computing power is no longer the limiting resource. The problem is that the computer cannot effectively listen to and communicate with the user. Therefore humans are serving computers, learning the commands the computer expects and sifting through screens of text to find the information they requested, when it should be the computer who is serving the user. Users should be able to interact with the computer as they would a colleague. Instead of typing and clicking, why shouldn't he be able to *tell* the computer what he wants it to do?

Current research in human-centered computing and human-computer interaction are working to change this paradigm using multimodal spaces. Instead of bringing the user to the computer, the computers are embedded in the user's environment. By creating a perceptually enabled environment, the computer "sees" and "hears" the user's speech and gestures, allowing the user to interact in a more natural way. In addition, the computer can choose visual or auditory modes of communication to most effectively present information to the user.

This is the vision for the Intelligent Room, a multimodal space under development at MIT's Artificial Intelligence Laboratory. A user who has never entered the Room should be able to walk in and get any information he needs without knowing what menus to click or what commands to type. To fully achieve this goal, sophisticated natural language processing is needed to understand the user's verbal commands.

Natural language systems with the ability to understand and answer a user's questions have typically been confined to a computer terminal, but what if we situate this technology in a multimodal environment to fill the need for language processing?

The focus of this project is demonstrating the use of START, a natural language query system, in the Intelligent Room. By enabling these technologies to communicate and share information, we can explore the possibility of doing better language processing, thus bringing us closer to our vision.

Section 2 describes the Intelligent Room and START, the research technologies at the focus of this project. Section 3 describes the motivation for this research. Sections 4 and 5 discuss the system design and implementation, in addition to a detailed example. Section 6 evaluates the system, and possibilities for future work are presented in Section 7. Finally, Section 8 presents the contributions of this project to the Intelligent Room.

2 RESEARCH TECHNOLOGIES

2.1 INTELLIGENT ROOM

The Intelligent Room is a multimodal environment for studying human-computer interaction and human-centered computing. Our current research space is a conference room equipped with computer-controllable projectors, audio equipment, lights, cameras, and microphones. However, there are no computers visible inside the Room. By embedding the computers in the walls, we can create an Intelligent space in which the user interacts with the computer naturally and focuses on what he would like to do rather than how to make the computer do it.

2.1.1 METAGLUE AGENT SYSTEM

Metaglue [1, 2] is the Room's system of distributed software agents, written in Java. These agents act on behalf of the devices, services, and people available in the Room. For example, there are agents for controlling lights, recognizing speech input, and interfacing with a web browser. Because the agents are all interconnected, they exchange information and provide the user with control over the Room's operation.

Agents also broadcast information about themselves. Any agent interested in those messages registers with the notification system to be informed when this information is broadcast. In this way, agents share information with anyone who is interested.

This project uses devices in the Room to illustrate how START can be used to query and control agents. Device agents represent devices in the Room, such as lights and projectors. Devices use the notification system to broadcast information about their state. In addition, device managers are device agents that represent several similar devices. For example, a room may contain multiple lamps, so a Light Manager is used to control the lamps individually or as a group.

2.1.2 RESOURCE MANAGER

In order to intelligently handle a user's request, the Room needs to be able to effectively manage the available resources. Agents may request services, and the resource manager

provides the agent with the best resource available to fulfill the request [3]. For example, an agent that wants to give the user a brief message will request a resource that will display a short text message. Based on the resources that are in use and what the user is doing, the resource manager determines whether the message should be spoken or sent to a visual display such as the LED sign.

2.1.3 SPEECH

Agents are controlled via speech by providing a rule grammar specifying the phrases the user may say. When the speech system hears the user say a phrase from a grammar, the agent is notified and responds accordingly. One of the limitations of this approach is that the user is limited to the set of commands that has been included in the grammar. If the programmer wants to make interaction easier for the user, he can include a variety of ways to say each command, but it is impossible to predict every way a user will say something.

Below is an excerpt from a grammar for the lights:

```
<nearPrep> = by | near | in front of | adjacent to | next to;
<lampPhrase> = [the] <lampN> <nearPrep> [the] <location> |
               [the] <location> <lampN>;
<lampReference> = <lampPhrase> | <allPhrase>;

<turnOn> = (illuminate | turn on) {verb-turnOn};
<turnOff> = (extinguish | turn off) {verb-turnOff};
<brighten> = (brighten) {verb-brighten};
<dim> = (dim) {verb-dim};

<verb> = <turnOn> | <turnOff> | <brighten> | <dim>;

public <sentenceOne> = <verb> <lampReference>;
```

This grammar would allow the user to say “Turn on the lamp by the window.” However, additional rules are required to allow “Turn the lamp by the window on.” As the phrases become more complicated, the grammars become more difficult to write and understand.

The speech recognition system that we use in the Room also provides a free dictation mode. Here, the user is not constrained to a set of phrases, so he can say virtually anything in the recognizer’s vocabulary. However, because the system needs to understand many more words, it is more difficult for the recognizer to accurately

transcribe speech in dictation mode, and therefore dictation is more error-prone than using rule grammars.

2.2 START

START, or SynTanctic Analysis using Reversible Transformations [4], is a natural language system that provides users with answers to English questions. By visiting START's web site [5], users can ask a variety of questions including: "Who directed *Gone with the Wind*?" "How far is Boston from San Jose?" and "Who works on robots at the AI Lab?"

START analyzes single sentences and converts them into subject-relation-object-triples. For example, "The light by the door is on," would be translated into:

```
(light is on)
(light related-to door)
```

Now when the user asks "Which light is on?" this is translated into similar expressions, with *which* substituted for the missing information:

```
(light is on)
(light related-to which)
```

START then retrieves the correct answer by matching expressions, using *which* as a matching variable.

In addition to retrieving information about single sentences, START provides other types of information (including showing web pages, playing video clips, and executing functions) by using natural language annotations which summarize the information in a source. We create these annotations by writing schemata that associate the sentences with some information. The following schema is for accessing information about the MIT AI Lab:

```
(def-schema
  :phrases
   '("ANY-AILAB-GROUP's members")
  :sentences
   '("ANY-AILAB-PERSON works in ANY-AILAB-GROUP"
     "ANY-AILAB-PERSON remains a member of ANY-AILAB-GROUP")
  :long-text '((show-ailab-group-members 'any-ailab-group
                                         'any-ailab-person)))
```

The `phrases` are used to answer queries like "Tell me about *x*." Matching symbols are used in the schema to match values from a database. The above example recognizes the request "Tell me about the Infolab group's members," since "Infolab group" is listed in the database as a match for `ANY-AILAB-GROUP`.

START analyzes the `sentences` in the schema, so users can ask "Who works in the Infolab group?" Whenever START receives a question that matches these phrases or sentences, it will reply with the information designated in `long-text`. In this case, it executes the function `show-ailab-group-members`, which displays information about the members of a group.

It is possible to have more than one schema that matches a particular sentence, so a single query may generate multiple replies. For example, the question "Who is Sean Connery?" will be answered by a schema for the Internet Movie Database, which gives information about movies and actors, and also by a schema that accesses biographies of well-known people.

There are several ways START could be used in the Intelligent Room. We could use it to ask questions about the Room ("Is the lamp on the desk on?") or to control agents ("Turn on the light by the door."). In addition, START could be used to do other types of linguistic processing. For example, by keeping track of the dialog with the user, START could handle pronoun disambiguation. If START knows that the user just asked about the lamp on the desk, then said "Turn it on," START should know which light the user is talking about. This project deals only with querying and controlling agents through START, but lays the foundation for future work to address other uses of this system.

3 MOTIVATION

In order to come closer to our vision of an environment that allows natural human-computer interaction, we require better natural language processing. This project lays the foundation for our vision by enabling START and the Intelligent Room to communicate and share information about agents, illustrated by the ability to pose queries and issue commands to agents in natural language.

If a user is to be allowed to speak naturally to the computer, he should not be constrained to only use commands in a grammar. By using free dictation and a system like START to parse and understand the speech we come closer to this goal. Although speech recognition through free dictation is more error-prone than using rule grammars, in the future we may be able to use the linguistic knowledge from the natural language system to improve the performance of the speech recognition tools.

Query systems, such as START, process questions expressed in natural language, but these systems typically have been confined to a computer terminal. By situating START in the Intelligent Room, we not only gain the ability to answer a wide variety of questions via START, but we can also take advantage of the linguistic information that START contains. With knowledge about how language works it is possible to move away from a system with constraining grammars to allow users to give commands in their own natural ways.

Not only can the Room access the information provided by START, but since it has the ability to gather information about what it "sees" and "hears," the Room can offer this to START's knowledge base. For example, the Room can provide START with the knowledge that the lights have been turned on or that someone has entered the Room. Therefore, the immediate benefit of integrating START and the Intelligent Room is the ability to use START to ask questions about the Room. In addition, by embedding a powerful natural language system in the Room, better language processing is possible in the future.

4 SYSTEM OVERVIEW

Imagine a person sitting in the Room says to the computer, “Show me the world map.” This seems like a simple enough task, but the system needs to know a great deal to accomplish this. First, the Room needs to hear the user’s request and parse it to figure out what the user has said. Once the system has figured out that the speaker has requested to see a map, the Room finds the Map Agent; however, before it is ready to show the user, the system needs to figure out how to display the map. Using cameras and microphones, the Room determines where the user is sitting, then figures out which displays are appropriate for the map, available, and convenient for the user from his location. Therefore, even a simple request may require a great deal of reasoning to respond in a way that best serves the user.

In the current system we use rule grammars to listen for the phrase “Show me the map,” but what if the user wants the Room to show his colleague, Alice, the map instead? Then the grammar would have to contain Alice’s name as well as every other person that we might want to show the map to. Therefore this is not an ideal solution. By using dictation to capture the speech and a natural language system that parses the sentence, the Room will handle “Show Alice the map.” Even if the Room has never seen Alice before, the system still understands the sentence and, if needed, asks the user to show the camera who Alice is. Therefore, to handle even simple requests, the Room needs to communicate with a system such as START to understand and reason about requests.

4.1 DESIGN CONSIDERATIONS

The Room’s existing START Agent communicates with START by routing queries through a web browser to START’s web interface. However, for this system I needed a more direct way to communicate with START. Sending commands through a browser is slow since query requests are competing with requests from Internet users who also have access to START’s web interface. In addition, the web interface does not allow users to assert knowledge or obtain information about the sentence structure. For these reasons, I

needed a direct connection to START. This is challenging because START operates in Lisp while the agents use Java.

Another important design consideration is where the knowledge in the system is stored. In the existing system, agents store their state internally. For example a Light Agent stores information about the light intensity level and whether it is on or off. This allows the light to easily reason about its own state, but using information from multiple agents is more difficult. For example, if the Room should dim the lights whenever it is empty, then it needs knowledge from the Light Agent and from the agent that keeps track of the number of people in the room. By having knowledge available in a central place, any agent who knows how to interpret the information will be able to reason about it. It would also be possible to store this information in a format that START could access directly.

4.2 System Architecture

Figure 1 shows a diagram of the system. All communications with START go through a single agent, the START Interface Agent. When START receives a query or command, it contacts this agent which dispatches the requests and passes back the responses. Within the START Interface Agent is a central repository for storing knowledge and a rule-based system for reasoning about the information.

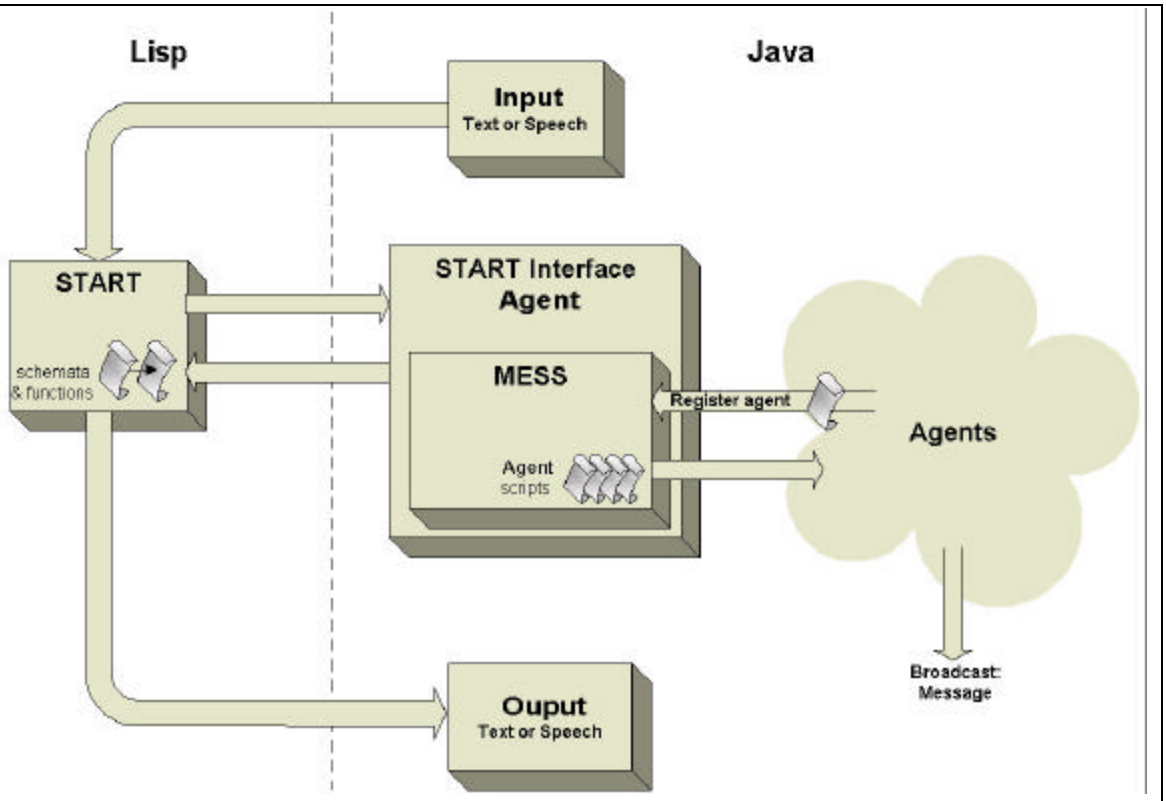

Figure 1. Diagram of START Interface System

5 SYSTEM IMPLEMENTATION

5.1 START TO METAGLUE INTERFACE

Enabling START and Metaglue to communicate is challenging because START operates in Lisp while Metaglue agents are written in Java. Since the existing START Agent communicates with START only through a web browser, the Intelligent Room group has not had to address this issue until now. However, JLinker [6], a tool made by Franz Inc., allows developers to dynamically link Lisp and Java applications. Therefore, JLinker removes the barrier between START and Metaglue by enabling invocation of methods in either environment.

JLinker creates a socket connection between two ports through which Lisp and Java communicate. Lisp will advertise on a port and wait for Java to connect or vice versa. In this way the connection can be initiated from either side.

5.1.1 LISP TO JAVA

START functions must communicate with Metaglue agents and, therefore, call Java methods. For example, the following Lisp expression calls the `doCommand` method in the `sia` object:

```
(jcall "doCommand" sia device state)
```

This evaluates to the return value of the equivalent Java code:

```
sia.doCommand(device, state);
```

Therefore, using `jcall` and other JLinker methods, programmers are able to manipulate Java objects in Lisp.

5.1.2 JAVA TO LISP

Lisp methods are invoked from Java in order to send input to START from Metaglue agents. The `JavaLinkDist` class, part of the JLinker tool, provides the interface to Lisp. For example:

```
JavaLinkDist.invokeInLisp(2,
                          "(initialize-start :server `ailab)");
```

executes the Lisp expression `(initialize-start :server `ailab)` and returns a copy of the result (indicated by the integer `2`). `invokeInLisp` returns an array of `TransStruct` objects (remote references to Java objects).

I built a layer on top of the low level functions provided by JLinker to abstract some of these operations. The `JavaToLisp` class provides methods to open a connection between Java and Lisp and send expressions to Lisp. `JavaToLisp` may be used by any agent that needs to communicate with Lisp. `StartConnection` is another layer on top of `JavaToLisp`. This class will launch a Lisp process, run START, and load files needed by START.

In this system, Java normally initiates the connection to START. Therefore, when `JavaToLisp` starts the Lisp process, it also tells Lisp the port on which to advertise:

```
(jl:jlinker-init :lisp-advertises :lisp-port lisp_port)
```

where `lisp_port` is set to an available port number. Java may then connect by calling:

```
JavaLinkDist.connect("localhost", lisp_port,
                     "localhost", java_port,
                     poll_interval, poll_count);
```

Once Java connects on its port, function calls are made across this channel.

5.2 INPUT TO START

Various Metaglue agents can provide users with an interface to START. An agent could provide a graphical interface into which the user types sentences or the system could listen for speech input through dictation, transcribe it into text, then pass the request to START through a `StartConnection`.

Allowing continuous speech input to START required some modifications to the speech system in the Room. Currently when an application uses dictation, it requires the user to say a specific command from a grammar before it will listen for free dictation. For example, a user might have to say “Query START” and wait for acknowledgement before asking his question. If all speech will be processed through START, forcing the user to say “Query START” each time would be unacceptable.

To address this issue, I enabled the speech system to listen continuously for dictation input. When the speech system processes the dictation, it broadcasts a message containing the spoken text. An agent listens for these broadcasts and sends the spoken text to START. In this way, the speech system listens to everything the user says and passes it to START. START determines if the sentence was a query, a command, or another type of utterance.

5.3 START

While START can be accessed via the web, this project requires a more direct connection to the START servers. Therefore, START is run in Allegro Lisp, and calls are made directly to START. Accessing START directly allows the Room to choose a server by launching a Lisp process using the appropriate START image file, indicating which sources of information to access.

Additional schemata and functions are loaded to tell START about the Room. When the user asks a question, START attempts to match the sentence with one of its schemata as described in Section 2.2. The matched schema will call a function which forwards the user's question to the START Interface Agent in Metaglue.

The following is a sample schema for the Intelligent Room:

```
(make-neuter-nouns
  (|CENTER LIGHT| :gens (light))
  (|CENTER ROW| :gens (center)))

(def-schema
  :sentences
   '("turn on the center light"
     "turn on the center row's light in center row"
     "turn on the center row of all lights")
  :long-text
   '((change-device-state "center lights" "on")))
```

The function `make-neuter-nouns` tells START that "center light" is a noun phrase and a type of "light." When this schema is loaded, START analyzes the sentences and creates the following expressions:

```
(*YOU* |TURN ON| |CENTER LIGHT-1|)

(LIGHT-1 RELATED-TO |CENTER ROW-1|)
(*YOU* |TURN ON| LIGHT-1)

(LIGHTS-1 QUANTIFIER ALL)
(|CENTER ROW-2| RELATED-TO LIGHTS-1)
(*YOU* |TURN ON| |CENTER ROW-2|)
```

The schema defined above will respond to requests including "Turn on the center light," "Turn on the light in the center," and "Turn the center row of lights on." Notice that separate schema are not needed for sentences of the form "Turn the light on," because START understands that this means the same as "Turn on the light." When any of these sentences trigger the schema, the pair of functions `change-device-state` and `change-device-state-aux`[1] are executed:

```
(defun change-device-state (device state)
  (change-device-state-aux device state))

(defun change-device-state-aux (device state)
  (if (jl::jcall "doCommand" sia device state)
      (recording-query-reply (t)
       (format t "<P>The ~A device is now ~A."
             (gen-np device)
             (gen-np state)))
      (recording-query-reply (t)
       (format t "<P>Changing the state ~A of device ~A failed."
             (gen-np state)
             (gen-np device)))))
```

This function forwards the command to the START Interface Agent (`sia`) through JLinker and formulates a reply to the user based on the value returned by the agent.

5.4 START INTERFACE AGENT

The START Interface Agent uses `StartConnection` to establish communication with START. As described above, this includes starting Allegro Lisp, running START, connecting through JLinker, and loading the necessary schemata and functions.

When START receives a request from the user, it analyzes the sentence, then passes the request to the START Interface Agent. The agent then asserts a fact to MESS (see

[1] The aux function exists because START stores the actual function definition with the schema, not just the function name. Therefore, by having an aux function, functions can be redefined without reloading the schema.

Section 5.5), the reasoning component of the system. For commands, the fact contains a list of arguments, the time the fact was asserted, a flag indicating whether the command has been executed, and the time the command finished. The `startDo` fact template for commands is shown below:

```
(deftemplate startDo
  (multislot args)
  (slot timestamp
    (default-dynamic (call (new java.util.Date) getTime)))
  (slot done (default FALSE))
  (slot finish-time))
```

The `startAsk` template for queries contains the same slots, plus a slot for the answer to the query:

```
(deftemplate startAsk
  (multislot args)
  (slot timestamp
    (default-dynamic (call (new java.util.Date) getTime)))
  (slot done (default FALSE))
  (slot finish-time)
  (slot answer))
```

When the rule engine runs, MESS sends a table of return values to the START Interface Agent. The agent checks this table for the result of the request and returns the result to START.

The agent also registers to receive notification of various events, such as device state changes. When the START Interface Agent receives notification of these events, it asserts this information to MESS. For device state change messages, the START Interface Agent asserts a `device-fact` containing the Agent ID of the device agent, the name of the state that has changed, the state's new value, the confidence of the value, the time the fact was asserted, and a flag indicating if this is the latest information about the device:

```
(deftemplate device-fact
  (slot device)
  (slot state (type STRING))
  (slot value)
  (slot confidence
    (type INTEGER)
    (default (get-member util.UncertainValue UNKNOWN)))
  (slot timestamp
    (default-dynamic (call (new java.util.Date) getTime)))
  (slot most-recent (default TRUE)))
```

The confidence indicates how certain the device is of its state. For example, when the device agent is created, it cannot tell what state it is in, so the confidence will have the value `GUESS`. Once the device agent changes the device's state, it knows what the state is, so the confidence value is `CERTAIN`. If no confidence is given, the fact uses `UNKNOWN` as the default value.

When a device's state changes, a new `device-fact` is asserted without deleting other `device-facts` about that state. By keeping a record of all the state changes for a fact, we can use this information for historical or statistical reasoning. As a result, there can be more than one fact about a given device and state. However, when the user asks about a device's state, sifting through all the facts to find the latest one would be time consuming. The `most-recent` flag helps alleviate this problem. When the `device-fact` is asserted the flag is set to `TRUE`. If there is another fact for the same device and state, initially, its flag may also be `TRUE`. The system contains rules for maintaining the `most-recent` flag, so if two facts indicate they have the most recent information for the device, the rule updates the older fact to indicate that it is no longer the newest data.

One of the implementation issues for this system was whether START should poll agents directly for information or whether the agents should broadcast information about themselves. For this initial version of the system, I chose to use the latter approach for several reasons. First, START only has to know about the START Interface Agent and not all the agents in the system, so it simplifies interactions with START. Secondly, by having the agents broadcast information, knowledge is collected in a central place.

5.5 REASONING IN MESS

The START Interface Agent uses a rule-based system called MESS, or the Metaglue Expert System Shell, to store and reason about the system's knowledge. MESS, developed by the Intelligent Room group, is an addition to JESS, the Java Expert System Shell [7].

As mentioned above, the START Interface Agent asserts a fact to MESS when it receives a query or command from START. These requests may be answered by any agent that

has registered with the START Interface Agent. Agents register themselves by giving MESS a script describing the requests the agent will handle. In this script the agent introduces itself by indicating its class. MESS takes this information and asserts a fact about the agent. For each agent that registers there is one `agent-info` fact that gives the agent's class (`occupation`), agent ID, and a pointer to the agent:

```
(deftemplate agent-info
  (slot occupation (type STRING))
  (slot agentID (type STRING))
  (slot agent))
```

Named agents, such as devices, store their names internally. For example, the user may refer to one lamp in the Room as both the "red lamp" and the "light on the desk." The agent that represents that light would store both of these names. When this agent introduces itself to MESS, it also gives its names, so START can refer to the agent by these names. Therefore, for each named agent, there may be several `agent-name` facts which associate a name with an agent ID:

```
(deftemplate agent-name
  (slot name (type STRING))
  (slot agentID (type STRING)))
```

The agent's script also contains rules for handling requests. Any agent can volunteer to handle a request from START by providing rules like the one shown below. The light script contains the following rule for handling requests to turn on the lights:

```
1  (defrule doLightOn
2    ?command <- (startDo (args ?name on) (done FALSE))
3    (name-agent-match (name ?name)
4                      (interface "agentland.device.light.Light")
5                      (agentID ?agentID) (agent ?agent))
6    =>
7    (bind ?did-it (call ?agent turnOn))
8    (modify ?command
9            (done ?did-it)
10           (finish-time (call (new java.util.Date) getTime)))
11   (addReturnValue (str-cat ?name " on") ?did-it))
```

Lines 2-5 of the above rule are the conditions for the rule to fire. There must be a `startDo` fact whose `args` contain any name and the word "`on`" (line 2). MESS will then backchain to find a `name-agent-match` fact for an agent with the given name and interface (lines 3-5). Because more than one agent can have the same name, the `interface` slot provides a way to ensure the rule only fires for the intended devices.

For example, there could be a command asserted with `(args "tape player" on)`. If both the VCR and cassette tape player have the name "tape player," then just fetching the agent with that name is not sufficient. The methods that turn on the VCR and cassette tape player may not have the same name, so the agents for these devices will have to provide rules that handle these commands differently. By using both the name and interface, the rule will retrieve the agent for the correct device. In addition, the `interface` slot in the `name-agent-match` facts takes advantage of the agent system's inheritance hierarchy. Therefore the rule above will look for agents with the given name and class, but also for agents that implement subclasses of `Light` such as `DimmableLight` and `X10Light`.

If the conditions for the rule are met, the consequences (lines 7-11 above) will be evaluated. In the sample rule shown, line 7 calls the `turnOn` method for the appropriate light. The original `startDo` fact is modified to indicate that the command has been executed and the time the command was finished (lines 8-10). Finally, MESS creates a return value using the command arguments as the key (line 11). After the rule engine runs, the START Interface Agent looks for this return value and passes it back to START.

Rules that query devices work in a similar vein. Instead of a `startDo` fact, queries use the `startAsk` fact. Rules for queries may have a condition requiring a fact that contains the answer:

```
(defrule askLightState
   ?query <- (startAsk (args ?name ?state) (done FALSE))
   (name-agent-match (name ?name)
                     (interface "agentland.device.light.Light")
                     (agentID ?agentID))
   (device-fact (device ?agentID) (state ?state)
                (value ?value) (most-recent TRUE))
   =>
   (modify ?query
           (answer ?value)
           (done TRUE)
           (finish-time (call (new java.util.Date) getTime)))
   (addReturnValue (str-cat ?name " " ?state) ?value))
```

Similar rules handle the case where there does not exist a `device-fact` containing the answer, and the agent must be asked directly.

While these examples illustrate commands and queries intended for a single device, rules can trigger more complex behaviors or perform more sophisticated queries.

5.6 OUTPUT

Once START receives a response from the START Interface Agent, it formats the answer as dictated by the START function that handled the request. START's responses are presented to the user with speech synthesis or visual display. The Room's Resource Manager can be used to direct the information to an appropriate output device based on the information to be displayed and the available output devices.

5.7 EXAMPLE INTERACTION

Figure 2 shows how a user's request is processed by the system. Even before the user talks to START, agents such as the Light Manager register with the START Interface Agent. The Light Manager introduces itself and the Lights it manages by asserting:

```
(introduce-manager-device
  (occupation "agentland.device.light.LightManager"))
```

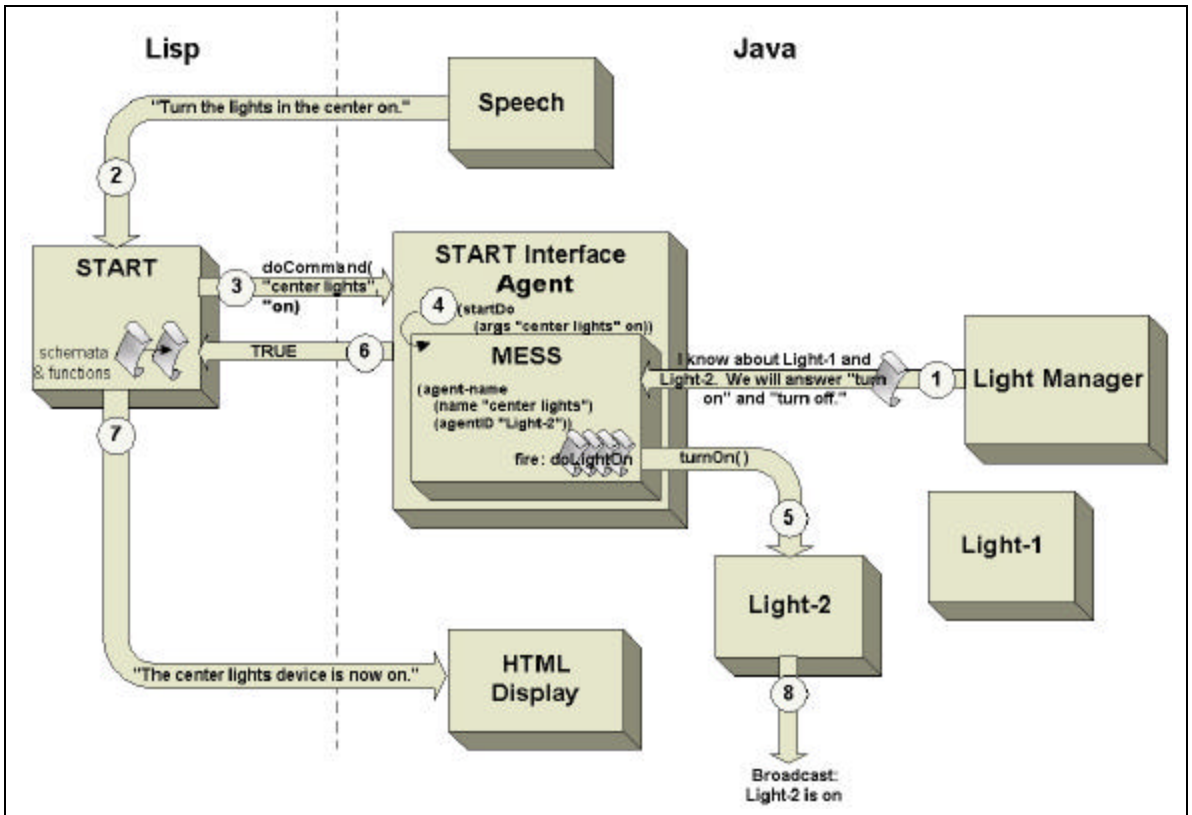

Figure 2. START Interface Example

MESS has rules that will handle the introductions and assert `agent-info` and `agent-name` facts for the Lights. The following is the `agent-info` fact for the Light Manager:

```
(agent-info
  (agent <External-Address:agentland.device.light.LightManagerEHA>)
  (agentID <External-Address:metaglue.AgentID>)
  (occupation "agentland.device.light.LightManager"))
```

For each Light and Light Manager, there are also one or more `agent-name` facts, including:

```
(agent-name (name "all lights")
  (agentID <External-Address:metaglue.AgentID>))

(agent-name (name "lights in the center")
  (agentID <External-Address:metaglue.AgentID>))

(agent-name (name "center lights")
  (agentID <External-Address:metaglue.AgentID>))

(agent-name (name "back lights")
  (agentID <External-Address:metaglue.AgentID>))
```

The script also contains rules for handling requests including `doLightOn` described in Section 5.5. In effect, the Light Manager's script says that it manages Light-1 and Light-2 and that all lights will respond to "turn on" and "turn off" requests (①).

Now, when the user says "Turn the lights in the center on" the speech system converts the spoken phrase into text and passes the request to START (②). START has a schema that tells it what to do with sentences with this meaning, so it follows the instructions in that schema by executing a function. START has identified that the user's sentence was a command intended for the center lights requesting them to turn on. The function passes these key parts of the user's request to the START Interface Agent (③) by calling:

```
doCommand("center lights", "on")
```

The START Interface Agent tells MESS about the command by asserting (④):

```
(startDo (args "center lights" on) (timestamp 989687533523))
```

MESS finds a `name-agent-match` fact indicating "center lights" implement the Light interface, so the `doLightOn` rule fires. MESS calls the *turnOn* method for Light-2 (⑤), creates a return value for "center lights on," and modifies the `startDo` fact:

```
(startDo (args "center lights" on)
         (timestamp 989687533523)
         (done TRUE)
         (finish-time 989687533619))
```

The START Interface Agent retrieves the value for “center lights on” and tells START the command has been completed (⑥). START tells the user the results of his request, in this case by displaying confirmation in the user’s browser window (⑦):

```
The center lights device is now on.
```

In the meantime, Light-2 has sent a notification that it has been turned on (⑧) and the START Interface Agent records this information in its knowledge base:

```
(device-fact
  (device "e21:agentland.device.light.X10DimmableLight-center")
  (state on) (value true) (confidence 50)
  (timestamp 989771927511 ))
```

6 EVALUATION OF IMPLEMENTATION

As described in the previous section, this system stores all the knowledge about the Room in a central location, so any agent can access the information. Because facts are stored with timestamps, there is a record of when events occur, and therefore, users could query the system for this information. In addition, no changes are required to existing agents to provide access via START. Programmers only need to write a script for the agent to handle appropriate requests.

Although this system provides the desired ability to control agents from START, there are several ways in which the system can be improved.

6.1 ERRORS IN DICTATION

One of the problems with speech input to START is that dictation is error-prone. Often the speech recognizer will mishear the user or recognize a word when the user has not spoken due to noise. When all of the speech input is directed to START, these errors result in many phrases that START does not understand. Therefore, the user will often hear or see "Sorry, I don't understand" from START, depending on whether speech or text output is used.

Improving performance of dictation is a hard task, but this issue can be addressed in a different way. Every reply from START has a reply quality value, which may indicate that START is certain of the answer, has no information about the question, or did not understand the question. By checking the reply quality, responses that were likely to have been generated by errors in dictation may be filtered.

6.2 ROBUSTNESS

The Metaglue agent system was designed to be robust, so that if one agent or the catalog fails, it does not bring down other agents. If an agent fails and is restarted, it will resume its interactions with other agents. Through JLinker, START communicates with these agents and the catalog. However, if START relies on an agent that dies, and the agent is restarted, START is not able to find the agent again. Therefore, it is necessary to

investigate whether JLinker can reconnect to an agent that has restarted or detect this condition and handle it automatically.

6.3 Multiple Agent Responses

Currently the system allows only one response for each query or command. In other words, if more than one agent has provided a rule that will handle the same request, only one will fire. In some cases it may be desirable to allow multiple answers to a query. For example, if the user asks "Which lights are on?" the system may choose to handle this with a rule that would be triggered for each `device-fact` for lights that are on. In this case, the rule should add the light to a list of return values. However, in the current implementation, queries are marked completed when one answer has been given. Therefore, this decision should be revisited to allow multiple agents to respond to a request.

6.4 Uncertainty in Device States

Device agents, such as the lights, maintain their state internally, but get no feedback from the physical device. When the agent is created, it is not certain of its initial state. Therefore, if the user asks about the state, the answer may only be a guess. In addition, if the user manually turns off the lights, the agent's state will not reflect these changes. This issue can be addressed by installing sensors in the Room to provide additional information about device state.

7 FUTURE DIRECTIONS

While this project primarily demonstrates the use of START to query and control devices, START and the reasoning in the system can be used in other interesting ways.

7.1 REASONING IN START

I chose to place most of the knowledge and reasoning on the Java side, because Metaglue agents already use MESS for some reasoning. However, some of the reasoning could be done by START. For example, notifications broadcast by agents could be asserted directly to START instead of using MESS to answer queries. With this method, the notifications would be formed into English sentences and analyzed by START. Then, queries would be answered by matching against the knowledge in START as described in Section 2.2.

7.2 RESOLVING AGENT REFERENCES

Currently the system only uses names to resolve references to agents. Therefore, either START or the agent itself needs to know what names the user is likely to use to refer to a device. Most of this is done by START, since it recognizes that several phrases refer to the same object. As shown in the schema in Section 5.3, START can then use just one name for the device. Therefore, even if the device has only one name, START allows the user to refer to the device in other ways as well.

A better way to resolve references to agents would be to use descriptions of the devices rather than fixed names. Work is currently being done in the Intelligent Room on investigating ways to achieve this using a rule-based system. Given a description of the object, the system would apply a set of rules and find the agent that represents the object. The description could be expressed as a fact including the type of device and its location relative to other objects in the Room. Since START parses the user's input, it will provide a way to fill in the description slots.

7.3 TROUBLESHOOTING FAILURES

This system uses MESS extensively for reasoning about requests from START. An interesting extension to this reasoning would be to provide the ability to troubleshoot failures of requests. If the user says, “turn on the center lights” and is told that the system is not able to complete the command, it would be useful to trace back and figure out what conditions were not met.

7.4 USE OF LANGUAGE AND GESTURES

Since the goal of human-centered computing is to provide natural interfaces for the user, this should include combined use of language and gestures to communicate with the computer. This project deals only with language, but it is feasible to expand this work to take gestures into consideration. If the user says, “Show me the map there,” the word “there” could refer to something he previously said or something he is pointing at. Therefore START would need to keep track of its dialog with the user in addition to being able to communicate with the vision system to watch the user’s gestures.

8 CONTRIBUTIONS

This project contributes to the Intelligent Room a process for easily creating a connection to START from Metaglue, allowing START to communicate with any agent. Based on this, reasoning about users' requests can be done on the Java side as I have shown, or alternatively the reasoning could occur in START.

This system illustrates how a natural language system such as START can be embedded in the Intelligent Room. This allows us to take advantage of the Room's perceptual abilities to provide information and START's linguistic knowledge to process users' speech. I have demonstrated this interaction by controlling agents through START. However, there is potential to do much more by utilizing more of START's knowledge of language. For example, START could be used to reinforce speech recognition. The speech recognizer provides a ranked list of possible ways to interpret what the user said. If START analyzes the sentences, it could determine which of these options are correct English sentences, and therefore are more likely to be the right interpretation.

While speech recognition through dictation is currently not as reliable as using rule grammars, this project gives us an idea what the whole system would look like with improved speech recognition. By using dictation to allow users to speak to the Room naturally and START to analyze the sentences, we can now investigate ways to use these technologies to perform better natural language processing in the Room.

9 REFERENCES

[1] Phillips, Brenton. *Metaglue: A Programming Language for Multi Agent Systems.* M.Eng. Thesis. Massachusetts Institute of Technology, Cambridge, MA, 1999.

[2] Warshawsky, Nimrod. *Extending the Metaglue Multi Agent System.* M.Eng. Thesis. Massachusetts Institute of Technology, Cambridge, MA, 1999.

[3] Gajos, Krzysztof. *A Knowledge-Based Resource Management System for the Intelligent Room.* M.Eng. Thesis. Massachusetts Institute of Technology, Cambridge, MA, 2000.

[4] Katz, Boris. "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.

[5] START Natural Language Question Answering System. http://www.ai.mit.edu/projects/infolab/

[6] JLinker: A Dynamic Link between Lisp and Java. http://www.ai.mit.edu/projects/infolab/

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

10 APPENDIX

room-schemata.lisp

```
;;; -*- Mode: LISP; Syntax: ANSI-Common-Lisp; Package: START -*-

(in-package :start)

;;; room-schemata.lisp

(make-neuter-nouns
  (|CENTER LIGHT| :gens (light))
  (|CENTER ROW| :gens (center))
  (|FRONT LIGHT| :gens (light))
  (|FRONT ROW| :gens (front))
  (|BACK LIGHT| :gens (light))
  (|BACK ROW| :gens (back))
  |MUX|
  (|VIDEO MUX| :gens (mux))
  (|VGA MUX| :gens (mux))
  (|AUDIO MUX| :gens (mux)))


;;;;;;;;;;;;;;
;;  LIGHTS  ;;
;;;;;;;;;;;;;;

(mapc #'smart-analyze-np-using-cache
      '((the center row^s light in the center row)
        (the front row^s light in the front row)
        (the back row^s light in the back row)))

;; LIGHTS ON

;; Answers:
;;   Turn the center lights on.
(def-schema
  :sentences
    '("turn on the center light"
      "turn on the center row's light in center row"
      "turn on the center row of all lights")
  :long-text '((change-device-state-simple "center lights" "on"))
  :sons '(*no-db-links*)
  :liza '()
  :function-call T)

;; Answers:
;;   Turn the front lights on.
(def-schema
  :sentences
    '("turn on the front light"
      "turn on the front row's light in front row"
      "turn on the front row of all lights"
      "turn on front light by the wall"
      "turn on front light by the board")
  :long-text '((change-device-state-simple "front lights" "on"))
  :sons '(*no-db-links*)
  :liza '()
```

```
  :function-call T)

;; Answers:
;;   Turn the back lights on.
(def-schema
  :sentences
    '("turn on the back light"
      "turn on back row's light in back row"
      "turn on back row of all lights")
  :long-text '((change-device-state-simple "back lights" "on"))
  :sons '(*no-db-links*)
  :liza '()
  :function-call T)

;; Answers:
;;   Turn all the lights on.
(def-schema
  :sentences
  '("turn on all the lights")
  :long-text '((change-device-state-simple "all lights" "on"))
  :sons '(*no-db-links*)
  :liza '()
  :supersedes-others T
  :function-call T)


;; LIGHTS OFF

;; Answers:
;;   Turn the center lights off.
(def-schema
  :sentences
    '("turn off the center light"
      "turn off the center row's light in center row"
      "turn off the center row of all lights")
  :long-text '((change-device-state-simple "center lights" "off"))
  :sons '(*no-db-links*)
  :liza '()
  :function-call T)

;; Answers:
;;   Turn the front lights off.
(def-schema
  :sentences
    '("turn off the front light"
      "turn off the front row's light in front row"
      "turn off the front row of all lights"
      "turn off the front light by the wall"
      "turn off the front light by the board")
  :long-text '((change-device-state-simple "front lights" "off"))
  :sons '(*no-db-links*)
  :liza '()
  :function-call T)

;; Answers:
;;   Turn the back lights off.
(def-schema
```

```
  :sentences
    '("turn off the back light"
      "turn off the back row's light in back row"
      "turn off the back row of lights")
  :long-text '((change-device-state-simple "back lights" "off"))
  :sons '(*no-db-links*)
  :liza '()
  :function-call T)

;; Answers:
;;   Turn all the lights off.
(def-schema
  :sentences
  '("turn off all the lights")
  :long-text '((change-device-state-simple "all lights" "off"))
  :sons '(*no-db-links*)
  :liza '()
  :supersedes-others T
  :function-call T)


;; IS LIGHT ON/OFF?

;; Answers:
;;   Are the center lights on?
;;   Are the center lights off?
(def-schema
  :phrases
    '("center row's light in the center row"
      "center row of all lights")
  :sentences
    '("center row's light in the center row remains on"
      "center light remains on"
      "center row of all lights remains on"
      "center row's light in the center row remains off"
      "center light remains off"
      "center row of all lights remains off")
   :long-text '((show-device-state-simple "center lights" "on"))
   :sons '(*no-db-links*)
   :liza '()
   :function-call T)


;; Answers:
;;   Are the front lights on?
;;   Are the front lights off?
(def-schema
    :phrases
    '("front row's light in front row"
      "front row of all lights")
    :sentences
    '("front row's light in the front row remains on"
      "front light remains on"
      "front row of all lights remains on"
      "front light by the wall remains on"
      "front light by the board remains on"
      "front row's light in the front row remains off"
```

```
      "front row of all lights remains off"
      "front light by the wall remains off"
      "front light by the board remains off")
   :long-text '((show-device-state-simple "front lights" "on"))
   :sons '(*no-db-links*)
   :liza '()
   :function-call T)

;; Answers:
;;   Are the back lights on?
;;   Are the back lights off?
(def-schema
   :phrases
    '("back row's light in the back row"
      "back row of all lights")
   :sentences
    '("back row's light in the back row remains on"
      "back light remains on"
      "back row of all lights remains on"
      "back row's light in the back row remains off"
      "back light remains off"
      "back row of all lights remains off")
   :long-text '((show-device-state-simple "back lights" "on"))
   :sons '(*no-db-links*)
   :liza '()
   :function-call T)


;;;;;;;;;;;;;;;;
;;  PROJECTOR  ;;
;;;;;;;;;;;;;;;;

;;; Answers:
;;;   Turn the projector on.
(def-schema
   :sentences
    '("Turn on the first projector")
   :long-text '((change-device-state-simple "projector 1" "on"))
   :sons '(*no-db-links*)
   :liza '()
   :function-call T)


;;; Answers:
;;;   Turn off the projector.
(def-schema
   :sentences
    '("Turn off the first projector")
   :long-text '((change-device-state-simple "projector 1" "off"))
   :sons '(*no-db-links*)
   :liza '()
   :function-call T)


;;; Answers:
;;;   Is the projector on?
(def-schema
```

```
   :sentences
    '("First projector remains on"
      "First projector remains off")
   :long-text '((show-device-state-simple "projector 1" "on"))
   :sons '(*no-db-links*)
   :liza '()
   :function-call T)


;;;;;;;;;;;;;;;;
;;  MESSENGER  ;;
;;;;;;;;;;;;;;;;

;;; Answers:
;;;   Send a message.
(def-schema
   :sentences
   '("send a message")
  :long-text '((send-command "sendMessage"))
   :sons '(*no-db-links*)
   :liza '()
   :function-call T)


;;;;;;;;;;;;;;;;
;;  SCHEDULER  ;;
;;;;;;;;;;;;;;;;

;;; Answers:
;;;   Send a reminder in 4 minutes.
(def-schema
   :sentences
    '("send a reminder to me in any-number any-legacy-time"
      "remind me in any-number any-legacy-time")
   :long-text '((send-reminder `any-number `any-legacy-time))
   :sons '(*no-db-links*)
   :liza '()
   :function-call T)


;;;;;;;;;;;;;;;;;;;;;;
;;  WEATHER FETCHER  ;;
;;;;;;;;;;;;;;;;;;;;;;

;;; Answers:
;;;   Tell me today's weather
(def-schema
    :phrases
    '("today's weather")
   :sentences
    '("read today's weather to me")
   :long-text '((send-command "readWeather"))
   :sons '(*no-db-links*)
   :liza '()
   :function-call T)
```

room-functions.lisp

```
;;; -*- Mode: LISP; Syntax: ANSI-Common-Lisp; Package: START -*-

(in-package :start)

;;; room-functions.lisp
(setq society (jl::jcall "getSociety" (cl-user::get-ma)))
(setq sia (jl::jcall "findAgent" (cl-user::get-ma)
                 (jl::jnew "metaglue.AgentID" society
                         "agentland.info.StartInterface")))

(defun make-lower-case (str)
  (map `string #'char-downcase str))

;;
;; Use this when asking about a device and using matching symbols
;;
(defun show-device-state (device-matching-symbol state)
  (show-device-state-aux device-matching-symbol state))

(defun show-device-state-aux (device-matching-symbol state)
  (let ((device (symbol-name (get-matching-value-root
                              device-matching-symbol))))
    (when device
      (let ((answer (jl::jcall "doQuery" sia (make-lower-case device)
                                             state)))
      (if answer
          (recording-query-reply (t)
            (format t "<P>The state of device ~A is ~A."
                  (gen-np device)
                  answer))
          (recording-query-reply (`know-dontknow)
            (format t "<P>I don't know about the state of device ~A."
                  (gen-np device)))))))))

;;
;; Use this when changing the state of a device and using matching
;; symbols
;;
(defun change-device-state (device-matching-symbol state &rest args)
  (change-device-state-aux device-matching-symbol state args))

(defun change-device-state-aux (device-matching-symbol state
                                                       &rest args)
  (let ((device (symbol-name (get-matching-value-root
                              device-matching-symbol))))
    (when device
      (recording-query-reply (t)
      (if (jl::jcall "doCommand" sia (make-lower-case device) state)
          (format t "<P>The state ~A of device ~A has successfully been
changed."
                (gen-np state)
                (gen-np device))
          (format t "<P>Changing the state ~A of device ~A failed."
                (gen-np state)
```

```
                (gen-np device)))))))


;;
;; Use this when asking about a device and not using matching symbols
;;
(defun show-device-state-simple (device state)
  (show-device-state-simple-aux device state))

(defun show-device-state-simple-aux (device state)
  (let ((answer (jl::jcall "doQuery" sia device state)))
    (if answer
      (recording-query-reply (t)
        (format t "<P>The state of device ~A is ~A."
              (gen-np device)
              answer))
      (recording-query-reply (`know-dont-know)
        (format t "<P>I don't know about the state of device ~A."
              (gen-np device))))))


;;
;; Use this when changing the state of a device and not using matching
;; symbols
;;
(defun change-device-state-simple (device state &rest args)
  (change-device-state-simple-aux device state args))

(defun change-device-state-simple-aux (device state &rest args)
  (if (jl::jcall "doCommand" sia device state)
      (recording-query-reply (t)
       (format t "<P>The state ~A of device ~A has successfully been
changed."
             (gen-np state)
             (gen-np device)))
      (recording-query-reply (t)
       (format t "<P>Changing the state ~A of device ~A failed."
             (gen-np state)
             (gen-np device)))))



;;
;; Use this to send a command that is not for any particular agent.
;; For example, "start the demo" will make a bunch of things happen,
;; but it does not correspond to a particular function in some agent.
;;
(defun send-command (command)
  (send-command-aux command))

(defun send-command-aux (command)
  (let ((answer (jl::jcall "doCommand" sia "" command)))
    (if answer
      (recording-query-reply (t)
                        (format t "Command successful."))
        (recording-query-reply (t)
```

```
                        (format t "Command failed.")))))

;;
;; Use this to send a reminder.  Checks the units and puts them in the
;; form that the agent expects.
;;
(defun send-reminder (number-match units-match)
  (send-reminder-aux number-match units-match))

(defun send-reminder-aux (number-match units-match)
  (let ((number (get-matching-value-root number-match))
      (units (get-matching-value-root-singular units-match)))
    (case units
      ((SEC SECOND) (setq unit "sec"))
      ((MIN MINUTE) (setq unit "min"))
      ((HOUR) (setq unit "hours"))
      ((DAY) (setq unit "days"))
      (otherwise (setq unit nil)))
    (if (and number unit)
      (let ((answer
             (jl::jcall "doCommand" sia ""
                    (concatenate `string "sendReminder "
                                 (symbol-name number) " " unit))))
        (if answer
            (recording-query-reply (t)
            (format t "I will send a reminder in ~A ~A."
                  number
                  unit))
            (recording-query-reply (t)
            (format t "I'm sorry, I can't send your reminder."))))
      "Please tell me when you would like to send a reminder.  For
example: \"Send a reminder in 4 hours.\"")))
```

JavaToLisp.java

```
package util;

import com.franz.jlinker.*;
import java.io.IOException;
import java.io.File;
import java.net.ServerSocket;
import java.text.StringCharacterIterator;
import java.util.LinkedList;
import java.util.List;
import javax.swing.JOptionPane;
import metaglue.LogStream;
import metaglue.PortableFileSystem;
import java.net.ServerSocket;

/**
 * This is a tool for connecting to Lisp through JLinker.  Use the
 * <code>connectToLisp()</code> method to make the connection.  Then
 * other methods such as <code>runInLisp</code> and
 * <code>loadFile</code> may be used.   The <code>main</code> method
 * will pop up a dialog box into which expressions may be typed and
 * evaluated.
 *
 * Created: Fri Jan 26 12:36:44 2001
 *
 * @author <a href="mailto:moltmans@ai.mit.edu">Michael Oltmans</a>
 * @author Katherine Koch
 * @version
 */

public class JavaToLisp {

  // time between poll attempts, measured in milliseconds
  private static int JLINK_POLL_INTERVAL = 40;
  // number of poll attempts to be made, measured in milliseconds
  private static int JLINK_POLL_COUNT = 600;

  // command to begin lisp application
  private static final String LISP_EXE = "alisp";

  // name of file to load when starting Lisp
  private static final String LOAD_FILE = "load-lisp-file.lsp";
  private static final String LOAD_PORT = "load-lisp-port.lsp";

  // directory containing the above files
  protected static final String PROGRAMDATA_DIR =
"hal/ProgramData/JLinker/";

  // name of file used to advertise ports
  private static final String ADVERTISE_FILE = "java-to-lisp.trp";

  // store the absolute file name
  private String m_loadFilePath;
  private String m_advertFilePath;
```

```
// absolute file path for files
protected String m_lispRoot;

private Process m_lispProcess;

// port for Lisp to advertise on
private static int m_lispPort = 5127;

// if false, connects on the port indicated by m_lispPort.
// otherwise uses file ADVERTISE_FILE to advertise ports
private boolean m_connectWithFile = false;


/**
 * Creates a new <code>JavaToLisp</code> instance.
 *
 */
public JavaToLisp (){
  this(false);
}


/**
 * Creates a new <code>JavaToLisp</code> instance.
 *
 * @param useFileToConnect if true, uses a file to advertise ports.
 * otherwise, connects directly on an open port
 */
public JavaToLisp(boolean useFileToConnect) {
  m_connectWithFile = useFileToConnect;

  m_lispRoot = new File(PortableFileSystem.homeFileSystem(),
                        PROGRAMDATA_DIR ).getAbsolutePath();

  if (m_connectWithFile) {
    m_loadFilePath = new File(m_lispRoot,
                              LOAD_FILE).getAbsolutePath();

    m_advertFilePath = new File(m_lispRoot,
                                ADVERTISE_FILE).getAbsolutePath();
  }
  else {
    m_loadFilePath = new File(m_lispRoot,
                              LOAD_PORT).getAbsolutePath();

    try {
      // find an open port
      ServerSocket tempSocket = new ServerSocket(0);
      m_lispPort = tempSocket.getLocalPort();
      tempSocket.close();
    } catch (IOException e) {
    }
  }

}
```

```
/**
 * Start a lisp process.
 *
 * @param command to run on startup
 * @param file to load on startup
 * @return <code>true</code> if lisp is started without error.
 */
boolean startLisp(String command, String file) {
  String[] args = {LISP_EXE, "-e", command, "-L", file};

  return startLisp(args);
}


/**
 * Start a lisp process using an image.
 *
 * @param command to run on startup
 * @param file to load on startup
 * @param image to load on startup
 * @return <code>true</code> if lisp is started without error.
 */
boolean startLisp(String command, String file, String image) {
  String[] args = {LISP_EXE, "-I", image, "-e", command, "-L", file};

  return startLisp(args);
}


/**
 * Start a Lisp process using the given arguments.
 *
 * @param args command line args to use
 * @return <code>true</code> if lisp is started without error.
 */
protected boolean startLisp(String[] args) {
  try {
    m_lispProcess = Runtime.getRuntime().exec(args);
  }
  catch (IOException e) {
    e.printStackTrace();
    return false;
  }

  return true;
}


/**
 * Start a Lisp process using the given image.
 *
 * @param image the image to use to load lisp.  Ignored if image is
 *   null.
 * @return  <code>true</code> if connected, <code>false</code> if the
 *   Lisp process or JLinker could not be started.
 */
```

```
public boolean connectToLisp(String image) {

  boolean result;
  String command;

  if (m_connectWithFile) {
    command = "(defvar *jlinker-advertise-file* \"" +
              escapeBackslashes(m_advertFilePath) + "\")";
    if (image == null) {
      result = startLisp(command, m_loadFilePath);
    } else {
      result = startLisp(command, m_loadFilePath, image);
    }
  }
  else {
    command = "(defparameter *lisp-port* " + m_lispPort + ")";
    if (image == null) {
      result = startLisp(command, m_loadFilePath);
    } else {
      result = startLisp(command, m_loadFilePath, image);
    }
  }

  if (result) {
    LogStream.log(LogStream.INFO, "Lisp process started");
  }
  else {
    LogStream.log(LogStream.ERROR, "Trouble starting Lisp process");
  }


  if (m_connectWithFile) {
    JavaLinkDist.connect(m_advertFilePath, "localhost", 0,
                         JLINK_POLL_INTERVAL, JLINK_POLL_COUNT);
  }
  else {
    JavaLinkDist.connect("localhost", m_lispPort, "localhost", 0,
                         JLINK_POLL_INTERVAL, JLINK_POLL_COUNT);
  }


  if (JavaLinkDist.query(true)) {
    LogStream.log(LogStream.INFO, "JLink established to lisp");
    return true;
  }
  else {
    LogStream.log(LogStream.ERROR, "Trouble starting JLinker");
    return false;
  }
}


/**
 * Start a Lisp process (if one is not already running) and connect
 * to JLinker.  (Note: If you have trouble connecting to JLinker,
 * check that old alisp processes have been killed (nicely) or wait
 * several minutes til they die on their own.  If you still have
```

```
 * trouble, try using a file to connect by using the constructor that
 * takes a boolean value.)
 *
 * @return <code>true</code> if connected, <code>false</code> if the
 *    Lisp process or JLinker could not be started.
 */
public boolean connectToLisp() {
  if (m_connectWithFile) {
    JavaLinkDist.connect(m_advertFilePath, "localhost", 0, 10, 2);
  }
  else {
    JavaLinkDist.connect("localhost", m_lispPort,
                         "localhost", 0, 10, 2);
  }

  if (JavaLinkDist.query(true)) {
    LogStream.log(LogStream.INFO, "Lisp already connected");
    return true;
  }

  return connectToLisp(null);
}


/**
 * Disconnect from JLinker, and kill the Lisp process if it was
 * started.
 *
 */
public void shutdown()
{
  if (JavaLinkDist.query()) {
    JavaLinkDist.disconnect();
  }

  // we started it so we should kill it...
  if (m_lispProcess != null) {
    m_lispProcess.destroy();
  }

}


////////////////////////////////
//
// Methods for Talking to Lisp
//
////////////////////////////////

public String readEvalPrint(String form)
{
  System.out.println(form);
  LispObj obj = runInLisp("read-eval-to-string", form);
  return obj.toString();
}

public LispObj readEval(String form)
```

```
{
  System.out.println(form);
  LispObj obj = runInLisp("read-eval", form);
  return obj;
}

public LispObj runInLisp(String command)
{
  TranStruct[] result = JavaLinkDist.invokeInLisp(2, command);
  return processResult(result, JavaLinkDist.newDistOb(command),
                       null);
}

public LispObj runInLisp(String command, String arg) {
  TranStruct[] args = new TranStruct[1];
  args[0] = JavaLinkDist.newDistOb(arg);
  TranStruct cmd = JavaLinkDist.newDistOb(command);
  return runInLisp(cmd, args);
}

public LispObj runInLisp(String command, String arg1, String arg2) {
  TranStruct[] args = new TranStruct[2];
  args[0] = JavaLinkDist.newDistOb(arg1);
  args[1] = JavaLinkDist.newDistOb(arg2);
  TranStruct cmd = JavaLinkDist.newDistOb(command);
  return runInLisp(cmd, args);
}

public LispObj runInLisp(String command, String arg1, String arg2,
                         String arg3)
{
  TranStruct[] args = new TranStruct[3];
  args[0] = JavaLinkDist.newDistOb(arg1);
  args[1] = JavaLinkDist.newDistOb(arg2);
  args[2] = JavaLinkDist.newDistOb(arg3);
  TranStruct cmd = JavaLinkDist.newDistOb(command);
  return runInLisp(cmd, args);
}

public LispObj runInLisp(String command, TranStruct[] args)
{
  TranStruct cmd = JavaLinkDist.newDistOb(command);
  return runInLisp(cmd, args);
}

public LispObj runInLisp(TranStruct cmd, TranStruct[] args) {
  TranStruct[] result = JavaLinkDist.invokeInLisp(2, cmd, args);
  return processResult(result, cmd, args);
}

public void runInLispNoReturn(String cmd) {
  JavaLinkDist.invokeInLisp(-1, cmd);
}

public void setq(String var, TranStruct value)
{
  runInLisp("java-setq",
```

```
            new TranStruct[] {JavaLinkDist.newDistOb(var),value});
}

public boolean loadFile(String path, String file)
{
  return loadFile(path + (path.endsWith("/") ? "" : "/") + file);
}

public boolean loadFile(String file)
{
  return readEval("(load \"" + escapeBackslashes(file) + "\")")
                 != null;
}


////////////////////////////////
////////////////////////////////

public static String list(List list)
{
  return list(list, false);
}

public static String list(List list, boolean isQuoted)
{
  String[] array = new String[list.size()];
  return list((String[])list.toArray(array), isQuoted);
}

public static String list(String[] items)
{
  return list(items, false);
}

public static String list(String[] items, boolean isQuoted)
{
  StringBuffer result = new StringBuffer(isQuoted ? "'(" : "(");
  for (int i=0; i<items.length; i++) {
    result.append(items[i]).append(" ");
  }
  return result.append(")").toString();
}

public static String quotes(String string)
{
  return "\"" + string + "\"";
}

static boolean isLispError(TranStruct[] result) {
  return (result.length == 1 && JavaLinkDist.errorP(result[0]));
}


/**
 * Returns input with backslashes escaped. ('\' becomes '\\')
 *
 * @param input a <code>String</code> value
```

```
 * @return a <code>String</code> value
 */
public String escapeBackslashes(String input) {

  String output = "";
  int index = input.indexOf('\\');
  if (index == -1) {
    // no \ are used
    output = input;
  } else {
    StringCharacterIterator sci = new StringCharacterIterator(input);
    for (char c = sci.first(); c != StringCharacterIterator.DONE;
         c = sci.next()) {
    // go through string one char at a time
    output += c;
    if (c == '\\') {
      // add an extra \
      output += c;
    }
    }
  }

  return output;
}



/**
 * Pops up a dialog box for evaluating expressions in Lisp.
 *
 * @param args (no arguments required)
 */
public static void main(String[] args){
  JavaToLisp l = new JavaToLisp();
  l.connectToLisp();

  boolean done = false;
  while (!done) {
    String inputValue =
    JOptionPane.showInputDialog("Enter a value to " +
                                "eval or \"quit\" to quit");
    if (inputValue == null ||
        inputValue.toLowerCase().equals("q") ||
        inputValue.toLowerCase().equals("quit") ||
        inputValue.toLowerCase().equals("exit"))
    {
      done = true;
    }
    else {
      String result = l.readEvalPrint(inputValue);
      System.out.println("eval: " + inputValue + ":\n");
      System.out.println(result);
    }
  }

  try {
    l.shutdown();
```

```
    }
    catch (Exception e){
    }

    System.exit(0);
  }


  protected void finalize() {
    shutdown();
  }


  private LispObj processResult(TranStruct[] result,
                                TranStruct command,
                                TranStruct[] args)
  {
    if (isLispError(result)) {
      LogStream.log(LogStream.ERROR, "Error evaling: " +
                    LispObj.toString(result[0]));
      String print = "  (" + LispObj.toString(command);

      if ( args != null ) {
        for (int i=0; i<args.length; i++) {
          print += " " + LispObj.toString(args[i]);
        } // end of for ()
      } // end of if ()
      System.out.println(print + ")");
      return null;
    }
    return LispObj.makeResult(result);
  }

}// JavaToLisp
```

StartConnection.java

```
package util;

import java.io.File;
import metaglue.LogStream;
import metaglue.PortableFileSystem;

/**
 * This is a tool for connecting to START. Use the
 * <code>connectToStart()</code> method to make the connection.  Then
 * other methods such as <code>fread</code> and <code>loadFile</code>
 * may be used.
 *
 * Created: Mon Mar 12 05:58:05 2001
 * @see JavaToLisp
 *
 * @author Katherine Koch
 * @version
 */

public class StartConnection extends JavaToLisp {

  private static final String IMAGE_FILE = "start/start-images/start-
ailab-allegro";
  private String m_imagePath;

  /**
   * Creates a new <code>StartConnection</code> instance.  You must
   * call <code>connectToStart()</code> to open the connection.
   *
   */
  public StartConnection () {
    m_imagePath = new File(m_lispRoot, IMAGE_FILE).getAbsolutePath();

  }


  /**
   * Opens a connection to START
   *
   * @return <code>true</code> if successful, <code>false</code>
   *    otherwise
   */
  public boolean connectToStart() {
    boolean success = connectToLisp(m_imagePath);
    if (!success) {
      LogStream.log(LogStream.ERROR,
                    "Could not open connection to Lisp.");
      return success;
    }

    return success;
  }
```

```
  /**
   * The <code>fread</code> function in START.
   *
   * @param input a string to give to <code>fread</code>
   * @return START's response
   */
  public String fread(String input) {
    return readEval("(with-output-to-string (out) " +
                    "(let ((*standard-output* out)) " +
                    "(with-input-from-string (in \"" +
                    input + "\") (start::fread in))))").toString();
  }


  public static void main(String[] args){
    StartConnection start = new StartConnection();
    start.connectToStart();
    System.out.println(start.fread("john loves mary"));
    System.out.println(start.fread("who does john love"));
  }


}// StartConnection
```

StartInterface.java

```
package agentland.info;

import metaglue.*;
import java.rmi.*;
import agentland.util.Secret;
import mess.*;


/**
 * Interface for <code>StartInterfaceAgent</code>.
 *
 * @author Katherine Koch
 *
 * @see Mess
 */
public interface StartInterface extends Mess {

  /**
   * Execute a command.
   *
   * @param object the object you are asking about
   * @param args other parts of the command
   * @return true if successful, false otherwise
   * @exception RemoteException if an error occurs
   */
  public boolean doCommand(String object, String args)
    throws RemoteException;


  /**
   * Execute a query and return the result.
   *
   * @param object the object you are asking about
   * @param args other parts of the query
   * @return the answer to the query if successful, null otherwise
   * @exception RemoteException if an error occurs
   */
  public String doQuery(String object, String args)
    throws RemoteException;


  /**
   * Make an assertion.
   *
   * @param assertion the fact to be asserted
   * @return true if successful, false otherwise
   * @exception RemoteException if an error occurs
   */
  public boolean doAssert(String assertion) throws RemoteException;


  public void hearStateChange(Secret s) throws RemoteException;
```

```
  /**
   * Prints the queries and commands that this agent can handle on
   *   System.out.
   *
   * @exception RemoteException if an error occurs
   */
  public void help() throws RemoteException;


  /**
   * Prints the facts in the system on System.out
   *
   * @exception RemoteException if an error occurs
   */
  public void showFacts() throws RemoteException;


  /**
   * Call this to set up connection to START
   *
   * @return <code>true</code> if successful
   * @exception RemoteException if an error occurs
   */
  public boolean connectToStart() throws RemoteException;


  /**
   * Call START's <code>fread</code> function.
   *
   * @param input a string to give to <code>fread</code>
   * @return START's response
   */
  public String fread(String input) throws RemoteException;


  /**
   * Close the connection to START.
   *
   * @exception RemoteException if an error occurs
   */
  public void closeConnectionToStart() throws RemoteException;


  public boolean isConnected() throws RemoteException;

} // StartInterface
```

StartInterfaceAgent.java

```
package agentland.info;

import agentland.util.*;
import agentland.device.*;
import java.io.File;
import java.rmi.*;
import java.util.*;
import jess.*;
import mess.*;
import metaglue.*;
import util.StartConnection;

/**
 * This is the agent that START should communicate with.
 * <code>StartInterfaceAgent</code> handles commands, queries, and
 * assertions issued by START.  This agent also listens for state
 * change notification from <code>DeviceAgent</code>s, and records the
 * state changes in Jess.  Scripts for agents that support START
 * queries and commands should be added to the script
 * agentland.info.scripts.knownAgents.  Use <code>fread</code> to send
 * queries or assertions to START.  Be sure to call
 * <code>connectToStart</code> to open the connection first.
 *
 * @author Katherine Koch
 *
 * @see AgentAgent
 * @see StartInterface
 */

public class StartInterfaceAgent extends MessAgent implements
StartInterface {

  protected Notifier notifier;

  private java.util.List validCommands;
  private java.util.List validQueries;

  private StartConnection start;
  private static final String PROGRAMDATA_DIR =
"hal/ProgramData/JLinker/";
  private String fileDirectory;

  private boolean connected = false;

  /**
   * Creates a new <code>StartInterfaceAgent</code> instance.
   *
   * @exception RemoteException if an error occurs
   */
  public StartInterfaceAgent() throws RemoteException {
    loadScript("agentland.info.scripts.startInterface");

    notifier = (Notifier) reliesOn("agentland.util.Notifier");
    notifier.addSpy(getAgentID(), "hearStateChange",
```

```
                  "device.*.stateChange");

  start = new StartConnection();

  fileDirectory = new File(PortableFileSystem.homeFileSystem(),
                           PROGRAMDATA_DIR ).getAbsolutePath();

  log(LogStream.INFO, "Done initializing StartInterfaceAgent");
}


/**
 * Execute a command.
 *
 * @param object the object you are asking about
 * @param args other parts of the command
 * @return true if successful, false otherwise
 * @exception RemoteException if an error occurs
 */
public boolean doCommand(String object, String args)
  throws RemoteException
{
  log(LogStream.INFO, "START told me to: " + object + " " + args);

  boolean success = false;

  String strFact = "(startDo (args ";
  String strReturnKey;
  boolean noObject = object.equals("");
  boolean noArgs = args.equals("");
  if ( !noObject && !noArgs) {
    strFact = strFact + "\"" + object + "\" " + args;
    strReturnKey = object + " " + args;
  }
  else if (noObject && !noArgs) {
    strFact = strFact + args;
    strReturnKey = args;
  }
  else if (!noObject && noArgs) {
    strFact = strFact + "\"" + object + "\"";
    strReturnKey = object;
  }
  else {
    return false;
  }
  strFact += ")(timestamp " + (new Date()).getTime() + "))";

  doAssert(strFact);
  Object retValue = getReturnValue(strReturnKey);

  log(LogStream.INFO, "Return value is : " + retValue);
  if (retValue == null) {
    success = false;
  } else {
    if (retValue instanceof Vector) {
    // got multiple replies
    Vector replies = (Vector) retValue;
```

```
    success = false;
    for (int i=0; i<replies.size(); i++) {
      success = success || ((Boolean) replies.get(i)).booleanValue();
    }
    } else {
    success = ((Boolean) retValue).booleanValue();
    }
  }

  if (success) {
    log(LogStream.INFO, "Command successful.");
  } else {
    log(LogStream.INFO, "Command failed.");
  }

  return success;
}


/**
 * Execute a query and return the result.
 *
 * @param object the object you are asking about
 * @param args other parts of the query
 * @return the answer to the query if successful, null otherwise
 * @exception RemoteException if an error occurs
 */
public String doQuery(String object, String args)
  throws RemoteException
{
  log(LogStream.INFO, "START aked me about: " + object + " " + args);

  boolean success = false;

  doAssert("(startAsk (args \"" + object + "\" " + args + ")" +
           " (timestamp " + (new Date()).getTime() + "))");
  Object answer = getReturnValue(object + " " + args);

  if (answer == null) {
    log(LogStream.INFO, "I could not answer your query.");
    return null;
  } else {
    log(LogStream.INFO, "The answer is: " + answer.toString());
    success = true;
    return answer.toString();
  }

}

/**
 * Make an assertion.
 *
 * @param assertion the fact to be asserted
 * @return true if successful, false otherwise
 * @exception RemoteException if an error occurs
 */
public boolean doAssert(String assertion) throws RemoteException {
```

```
  boolean success = false;
  // process anything that already there
  run();


  // assert the fact in Jess
  log(LogStream.INFO, "Asserting fact: " + assertion);

  if (assertStringFact(assertion) == null) {
    log(LogStream.ERROR, "Problem asserting fact: " + assertion);
  }
  else {
    success = true;
  }
  run();

  return success;
}


public void hearStateChange(Secret s) throws RemoteException {
  // assert what you heard in the database
  //log(LogStream.INFO, "Got a secret!");
  String source = s.getSource().toString();
  DeviceState state = (DeviceState) s.details();

  String fact = "(device-fact (device \"" + source + "\") (state " +
                state.getName() + ") (value " +
                state.getStringValue() + ") (confidence " +
                state.getConfidence() + ") (timestamp " +
                (new Date()).getTime() + " ))";

  log(LogStream.INFO, "Asserting fact: " + fact);
  assert(fact);
}


/**
 * Prints the queries and commands that this agent can handle on
 *   System.out.
 *
 * @exception RemoteException if an error occurs
 */
public void help() throws RemoteException {
  if ((validCommands == null) || (validQueries == null)) {
    if (doAssert("(getValidStatements)")) {
      validQueries = (java.util.List)getReturnValue("validQueries");
      validCommands = (java.util.List)getReturnValue(
                      "validCommands");
    }
  }

  if (validQueries != null) {
    System.out.println("Valid queries are:");
    Iterator queryIter = validQueries.iterator();
    while (queryIter.hasNext()) {
      System.out.println("  " + queryIter.next());
```

```
    }
  } else {
    System.out.println("There are no valid queries.\n");
  }

  if (validCommands != null) {
    System.out.println("Valid commands are:");
    Iterator commIter = validCommands.iterator();
    while (commIter.hasNext()) {
      System.out.println("  " + commIter.next());
    }
  } else {
    System.out.println("There are no valid commands.\n");
  }

  System.out.println("Any fact is a valid assertion.\n");


}


/**
 * Prints the facts in the system on System.out
 *
 * @exception RemoteException if an error occurs
 */
public void showFacts() throws RemoteException {
  Vector facts = facts();
  System.out.println("\nJust the facts:\n");
  for (int i=0; i<facts.size(); i++) {
    System.out.println(facts.get(i).toString());
  }
  System.out.println("\n");
}



//////////////////////
//
// For talking to START
//
//////////////////////

/**
 * Call this to set up connection to START
 *
 * @return <code>true</code> if successful
 * @exception RemoteException if an error occurs
 */
public boolean connectToStart() throws RemoteException {

  String functions = new File(fileDirectory,
                     "room-functions.lisp").getAbsolutePath();
  String schemata = new File(fileDirectory,
                     "room-schemata.lisp").getAbsolutePath();

  // Open connection
```

```
  boolean success = start.connectToStart();
  if (!success) {
    log(LogStream.ERROR, "Problem connecting to START");
    return success;
  }

  // load metaglue.lisp
  String file = new File(fileDirectory,
                         "metaglue.lisp").getAbsolutePath();
  success = start.loadFile(file);
  if (!success) {
    LogStream.log(LogStream.ERROR, "Error loading " + file);
    return success;
  }

  // start a metaglue agent for lisp
  String society = getAgentID().getSociety();
  String catalog = getCatalog().whereAreYou().getHostName();
  start.readEval("(start-metaglue-agent \"" + society +
                 "\" \"" + catalog + "\" \"\")");

  // load the schemata and functions
  success = start.loadFile(functions);
  if (!success) {
    log(LogStream.ERROR, "Problem loading " + functions);
    return success;
  }
  success = start.loadFile(schemata);
  if (!success) {
    log(LogStream.ERROR, "Problem loading " + schemata);
    return success;
  }

  log(LogStream.INFO, "Connected to START.");
  connected = true;
  return success;
}


/**
 * Call START's <code>fread</code> function.
 *
 * @param input a string to give to <code>fread</code>
 * @return START's response
 */
public String fread(String input) throws RemoteException {
  String reply = start.fread(input);
  System.out.println(reply);
  return reply;
}


/**
 * Close the connection to START.
 *
 * @exception RemoteException if an error occurs
 */
```

```
  public void closeConnectionToStart() throws RemoteException {
    connected = false;
    start.shutdown();
  }

  public boolean isConnected() throws RemoteException {
    return connected;
  }

} // StartInterfaceAgent
```

startInterface.clp

```
;; agentland.info.scripts.startInterface
;(watch all)

(deftemplate startAsk
  "Represents a question posed to START.  args gives the query.  answer
  will be the answer to the query.  done will be true when the question
  has been answered.  finish-time will tell when the query was
  answered."
  (multislot args)
  (slot answer)
  (slot done (default FALSE))
  (slot finish-time)
  (slot timestamp (default-dynamic
                    (call (new java.util.Date) getTime))))

(deftemplate startDo
  "Represents a command issued to START.  args gives the command.  done
  will be true when the command has been completed.  finish-time will
  tell when the command was done."
  (multislot args)
  (slot done (default FALSE))
  (slot finish-time)
  (slot timestamp (default-dynamic
                   (call (new java.util.Date) getTime))))

(deftemplate agent-info
  "Information about an agent.  agent is the Agent object.  agentID is
  the agent's AgentID.  occupation is the occupation part of the
  agentID."
  (slot agent)
  (slot agentID (type STRING))
  (slot occupation (type STRING)))

(deftemplate agent-name
  "Indicates a name an agent goes by.  One agent may have several
  names, so there will be an agent-name fact for each name."
  (slot name (type STRING))
  (slot agentID (type STRING)))

(deftemplate is-a
  "Indicates that an agent with the given occupation implements the
  given interface."
  (slot occupation (type STRING))
  (slot interface (type STRING)))
(do-backward-chaining is-a)

(deftemplate name-agent-match
  "Used to find information about an agent."
  (slot name (type STRING))
  (slot interface (type STRING))
  (slot agentID (type STRING))
  (slot agent))
(do-backward-chaining name-agent-match)
```

```
(deftemplate introduce-agent
  "Used to introduce an agent to the system using the given name"
  (slot occupation (type STRING))
  (slot name (type STRING)))

(deftemplate user-info
  "Information about the users."
  (slot name (type STRING))
  (slot initials (type STRING)))

(deftemplate current-speaker
  "Tells us who is currently speaking.  There should be only one of
  these facts present."
  (slot name))


(defglobal
    ?*validQueries* = (create$)
    ?*validCommands* = (create$))

(deffunction addValidQueries ($?newQueries)
  "Adds the newQueries to the list of valid queries."
  (bind ?*validQueries* (create$ ?*validQueries* $?newQueries))
  (setReturnValue "validQueries"
                  (call java.util.Arrays asList ?*validQueries*)))

(deffunction addValidCommands ($?newCommands)
  "Adds the newCommands to the list of valid commands."
  (bind ?*validCommands* (create$ ?*validCommands* $?newCommands))
  (setReturnValue "validCommands"
                  (call java.util.Arrays asList ?*validCommands*)))

(run) ; run so we can use the templates in the following rules


(defrule make-is-a-1
  "If A is-a B and B is-a C, then A is-a C."
  ?f <- (need-is-a (occupation ?A) (interface ?C&~?A))
  (is-a (occupation ?A) (interface ?B))
  (is-a (occupation ?B) (interface ?C))
  =>
  (retract ?f)
  (assert (is-a (occupation ?A) (interface ?C))))


(defrule make-is-a-2
  "Asserts A is-a B facts for the given A."
  (need-is-a (occupation ?A) (interface ?B&~A))
  =>
  (bind ?class (call Class forName ?A))
  (bind ?interfaces (call ?class getInterfaces))
  (foreach ?x ?interfaces
    (assert (is-a (occupation ?A) (interface (call ?x getName))))))


(defrule make-is-a-3
```

```
  "Asserts A is-a A."
  ?f <- (need-is-a (occupation ?A) (interface ?A))
  =>
  (retract ?f)
  (assert (is-a (occupation ?A) (interface ?A))))


(defrule make-name-agent-match
  "Finds agentID and agent for the given name and interface."
  ?f <- (need-name-agent-match (name ?name) (interface ?int))
  (agent-name (name ?name) (agentID ?agentID))
  (agent-info (agentID ?agentID) (agent ?agent) (occupation ?occ))
  (is-a (occupation ?occ) (interface ?int))
  =>
  (retract ?f)
  (assert (name-agent-match (name ?name) (agentID ?agentID)
                      (agent ?agent) (interface ?int))))


(defrule set-up-agent
  "Asserts agent-info and agent-name facts about the agent."
  ?f <- (introduce-agent (occupation ?occ) (name ?name))
  =>
  (bind ?agent (reliesOn ?occ))
  (bind ?agentID (call (call ?agent getAgentID) toString))
  (assert (agent-info (agent ?agent) (agentID ?agentID)
                      (occupation ?occ)))
  (assert (agent-name (name ?name) (agentID ?agentID)))
  (retract ?f))


(defrule clean-up-need-is-a
  "Removes old need-is-a rules if not needed."
  (not (exists (startDo (done FALSE))))
  (not (exists (startAsk (done FALSE))))
  ?f <- (need-is-a)
  =>
  (retract ?f))


;; include scripts for agents we know about
(includeScript "agentland.info.scripts.knownAgents")

(run)
```

knownAgents.clp

```
;; agentland.info.scripts.knownAgents
;; Include the scripts for all the agents that we want
;; START to be able to get information from.
(includeScript "agentland.device.scripts.device")

(includeScript "agentland.device.scripts.light")
(includeScript "agentland.device.scripts.projector")
(includeScript "agentland.util.scripts.messenger")
(includeScript "agentland.util.scheduler.scripts.scheduler")
(includeScript "agentland.info.scripts.weather")
```

device.clp

```
(deftemplate device-fact
  "Information about a device.  device is the agent's agentID.  The
  state called state has the indicated value with the given confidence.
  timestamp tells when the fact was asserted.  most-recent is true if
  this is the newest fact about this device."
  (slot device)
  (slot state (type STRING))
  (slot value)
  (slot confidence (type INTEGER)
                   (default (get-member util.UncertainValue UNKNOWN)))
  (slot timestamp (default-dynamic
                    (call (new java.util.Date) getTime)))
  (slot most-recent (default TRUE)))

(deftemplate device-state
  "Matches an integer indicating device state to a string describing
  the state.  Occupation indicates what type of device this is for.
  State indicates what type of state these values describe."
  (slot state (type STRING))
  (slot intvalue (type INTEGER))
  (slot strvalue (type STRING))
  (slot occupation (type STRING))
  (slot foo))

(deftemplate introduce-device
  "Used to introduce a device to the system."
  (slot occupation (type STRING))
  (slot agentID (default "not-specified")))

(deftemplate introduce-manager-device
  "Used to introduce a device that manages other devices."
  (slot occupation (type STRING)))

(run) ; run so we can use the templates


(defrule maintain-device-fact-most-recent
  "If two facts for the same device indicate they are both the most
  recent fact asserted about that device, change the flag for the older
  fact"
  ?f1 <- (device-fact (device ?d) (state ?s) (most-recent TRUE)
                      (timestamp ?t1))
  ?f2 <- (device-fact (device ?d) (state ?s) (most-recent TRUE)
                      (timestamp ?t2&~?t1))
  =>
  (if (< ?t1 ?t2)
    then
      (modify ?f1 (most-recent FALSE))
    else
      (modify ?f2 (most-recent FALSE))))


(defrule set-up-device-with-occupation
  "Finds the agentID for the agent with the given occupation."
```

```
  ?f <- (introduce-device (occupation ?occ) (agentID "not-specified"))
  =>
  ;; add the agentID to the introduce-device fact
  (bind ?agent (reliesOn ?occ))
  (bind ?agentID (call ?agent getAgentID))
  (modify ?f (agentID ?agentID))
)


(defrule set-up-device-with-agentID
  "Asserts agent-info and agent-name facts about this agent."
  ?f <- (introduce-device (occupation ?occ)
                          (agentID ?agentID&~"not-specified"))
  =>
  (bind ?agent (reliesOn ?agentID))
  ;; assert the agent-info fact
  (assert
    (agent-info (agent ?agent) (agentID ?agentID) (occupation ?occ)))
  ;; assert agent-name facts
  (bind ?names (call (call ?agent getNames) toArray))
  (foreach ?name ?names
    (assert (agent-name (name ?name) (agentID ?agentID))))
  ;; retract the introduce-device fact
  (retract ?f)
)


(defrule set-up-manager-device
  "Introduce the manager device and all the devices it manages."
  ?f <- (introduce-manager-device (occupation ?occ))
  =>
  ;; introduce manager
  (bind ?agent (reliesOn ?occ))
  (assert (introduce-device (occupation ?occ)))
  ;; introduce managed devices
  (bind ?devices (call (call ?agent getDevices) toArray))
  (foreach ?agentID ?devices
    (assert
       (introduce-device (agentID ?agentID)
                         (occupation (call ?agentID getOccupation)))))
  ;; retract the introduce-device fact
  (retract ?f))
```

light.clp

```
;; Describes queries expected by LightManagerAgent

(defglobal ?*addedLightStatements* = FALSE)


(deffacts light-state-info
   "Define some things we know about Lights."
   (introduce-manager-device
     (occupation "agentland.device.light.LightManager")))

(reset) ; reset to load these facts


;;;;;;;;;;;;;;;;;
;; handle queries
;;;;;;;;;;;;;;;;;

(defrule askLightState
   "We want to know the brightness level or whether the light is
   on/off, and we have a fact that tells us the answer."
   ?query <- (startAsk (args ?name ?state) (done FALSE))
   (name-agent-match (name ?name)
                     (interface "agentland.device.light.Light")
                     (agentID ?agentID))
   (device-fact (device ?agentID) (state ?state)
                (value ?value) (most-recent TRUE))
   =>
   (modify ?query
           (answer ?value)
           (done TRUE)
           (finish-time (call (new java.util.Date) getTime)))
   (addReturnValue (str-cat ?name " " ?state) ?value))


(defrule assertLightLevel
  "We want to know the brightness level, but we have no facts about the
  device.  Ask the device and assert a fact."
  (startAsk (args ?name dimLevel) (done FALSE))
  (name-agent-match
    (name ?name) (agentID ?agentID) (agent ?agent)
    (interface "agentland.device.light.X10DimmableLight"))
  (not (exists (device-fact (device ?agentID) (state dimLevel)
                            (most-recent TRUE))))
  =>
  (bind ?state (call ?agent getState dimLevel))
  (assert (device-fact (device ?agentID) (state dimLevel)
                       (value (call ?state getIntValue))
                       (confidence (call ?state getConfidence)))))


(defrule assertLightOnOff
  "We want to know if the light is on or off, but we have no facts
  about the device.  Ask the device and assert a fact."
  (startAsk (args ?name on) (done FALSE))
```

```
  (name-agent-match
    (name ?name) (agentID ?agentID) (agent ?agent)
    (interface "agentland.device.light.X10Light"))
  (not (exists (device-fact (device ?agentID) (state on)
                            (most-recent TRUE))))
  =>
  (bind ?state (call ?agent getState on))
  (assert (device-fact (device ?agentID) (state on)
                       (value (call ?state getStringValue))
                       (confidence (call ?state getConfidence)))))

;;;;;;;;;;;;;;;;;;
;; handle commands
;;;;;;;;;;;;;;;;;;

(defrule doLightOn
  "Turn the light on"
  ?command <- (startDo (args ?name on) (done FALSE))
  (name-agent-match (name ?name) (agentID ?agentID) (agent ?agent)
                    (interface "agentland.device.light.Light"))
  =>
  (bind ?did-it (call ?agent turnOn))
  (modify ?command
          (done ?did-it)
          (finish-time (call (new java.util.Date) getTime)))
  (addReturnValue (str-cat ?name " on") ?did-it))


(defrule doLightOff
  "Turn the light off"
  ?command <- (startDo (args ?name off) (done FALSE))
  (name-agent-match (name ?name) (agentID ?agentID) (agent ?agent)
                    (interface "agentland.device.light.Light"))
  =>
  (bind ?did-it (call ?agent turnOff))
  (modify ?command
          (done ?did-it)
          (finish-time (call (new java.util.Date) getTime)))
  (addReturnValue (str-cat ?name " off") ?did-it))


(defrule doLightSetLevel
  "Set the light to a certain brightness"
  ?command <- (startDo (args ?name setLevel ?level) (done FALSE))
  (test (and (>= ?level 0) (<= ?level 100)))
  (name-agent-match (name ?name) (agentID ?agentID) (agent ?agent)
                    (interface "agentland.device.light.DimmableLight"))
  =>
  (bind ?did-it (call ?agent setLevel ?level))
  (modify ?command
          (done ?did-it)
          (finish-time (call (new java.util.Date) getTime)))
  (addReturnValue (str-cat ?name " setLevel " ?level) ?did-it))


;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;
```

```
(defrule updateValidStatements-light
    (test (neq ?*addedLightStatements* TRUE))
    (getValidStatements)
    =>
    (addValidQueries (create$ "light on" "light dimLevel"))
    (addValidCommands
      (create$ "light on" "light off" "light setLevel <level>"))
    (bind ?*addedLightStatements* TRUE))
```

messenger.clp

```
;; agentland.util.scripts.messenger
;; Describes queries expected by MessengerAgent

(defglobal
    ?*addedMessengerStatements* = FALSE
    ?*SpeechTextOutput* = (reliesOn "agentland.text.SpeechTextOutput")
    ?*GrammarCenter* = (reliesOn "speech.GrammarCenter"))


(deffacts messenger-state-info
  "Define some things we know about Messengers."
  (introduce-agent (occupation "agentland.util.Messenger")
                   (name "messenger")))

(reset) ; reset to load these facts


;;;;;;;;;;;;;;;;;;;
;; handle commands
;;;;;;;;;;;;;;;;;;;

(defrule doSendMessage
  "Send a message"
  ?command <- (startDo (args sendMessage) (done FALSE))
  (name-agent-match (name "messenger") (agent ?agent)
                    (interface "agentland.util.Messenger"))
  (current-speaker (name ?name))
  (user-info (name ?name) (initials ?who))
  =>
  (bind ?priority 3) ; set a default priority value
  ; get text of message
  (call ?*SpeechTextOutput* outputText
        "What message would you like to send?")
  (bind ?message (call ?*GrammarCenter* getDictation))
  (printout t ?message crlf)
  (bind ?did-it (call ?agent deliver ?who ?message ?priority))
  (if ?did-it
      then
      (printout t "Delivered message: " ?message " " ?who
                  " " ?priority crlf)
      else
      (printout t "Message not delivered: " ?message " " ?who
                  " " ?priority crlf))
  (modify ?command
          (done ?did-it)
          (finish-time (call (new java.util.Date) getTime)))
  (addReturnValue "sendMessage" ?did-it))


;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;


(defrule updateValidStatements-messenger
```

```
(test (neq ?*addedMessengerStatements* TRUE))
(getValidStatements)
=>
(addValidQueries (create$ ))
(addValidCommands
  (create$ "sendMessage"))
(bind ?*addedMessengerStatements* TRUE))
```

projector.clp

```
;; Describes queries expected by ProjectorAgent

(defglobal
    ?*addedProjectorStatements* = FALSE
    ?*asker* = (reliesOn "speech.tools.Asker"))


(deffacts projector-state-info
  "Define some things we know about Projectors."
  (introduce-manager-device
     (occupation "agentland.device.ProjectorManager")))

(reset) ; reset to load these facts


;;;;;;;;;;;;;;;;;;
;; handle queries
;;;;;;;;;;;;;;;;;;

(defrule askProjectorState
   "We want to know whether the projector is on/off,
    and we have a fact that tells us the answer."
   ?query <- (startAsk (args ?name ?state) (done FALSE))
   (name-agent-match (name ?name)
                     (interface "agentland.device.Projector")
                     (agentID ?agentID))
   (device-fact (device ?agentID) (state ?state)
                (value ?value) (most-recent TRUE))
   =>
   (modify ?query
           (answer ?value)
           (done TRUE)
           (finish-time (call (new java.util.Date) getTime)))
   (addReturnValue (str-cat ?name " " ?state) ?value))


(defrule assertProjectorOnOff
  "We want to know if the projector is on or off, but we have no facts
  about the device.  Ask the device and assert a fact."
  (startAsk (args ?name on) (done FALSE))
  (name-agent-match (name ?name) (agentID ?agentID) (agent ?agent)
                    (interface "agentland.device.Projector"))
  (not (exists (device-fact (device ?agentID) (state on)
                            (most-recent TRUE))))
  =>
  (bind ?state (call ?agent getState on))
  (assert (device-fact (device ?agentID) (state on)
                       (value (call ?state getStringValue))
                       (confidence (call ?state getConfidence)))))


;;;;;;;;;;;;;;;;;;
;; handle commands
;;;;;;;;;;;;;;;;;;
```

```
(defrule doProjectorOn
  "Turn the projector on"
  ?command <- (startDo (args ?name on) (done FALSE))
  (name-agent-match (name ?name) (agentID ?agentID) (agent ?agent)
                    (interface "agentland.device.Projector"))
  =>
  (bind ?did-it (call ?agent turnOn))
  (modify ?command
          (done ?did-it)
          (finish-time (call (new java.util.Date) getTime)))
  (addReturnValue (str-cat ?name " on") ?did-it))


(defrule doProjectorOff
  "Turn the projector off"
  ?command <- (startDo (args ?name off) (done FALSE))
  (name-agent-match (name ?name) (agentID ?agentID) (agent ?agent)
                    (interface "agentland.device.Projector"))
  (test (eq? (get-member "speech.tools.Asker" "YES")
             (call ?*asker* ask
               (str-cat "Are you sure you want to turn off " ?name)
               5)))
  =>
  (bind ?did-it (call ?agent turnOff))
  (modify ?command
          (done ?did-it)
          (finish-time (call (new java.util.Date) getTime)))
  (addReturnValue (str-cat ?name " off") ?did-it))


;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;


(defrule updateValidStatements-projector
    (test (neq ?*addedProjectorStatements* TRUE))
    (getValidStatements)
    =>
    (addValidQueries (create$ "projector on"))
    (addValidCommands
      (create$ "projector on" "projector off"))
    (bind ?*addedLightStatements* TRUE))
```

scheduler.clp

```
;; agentland.util.scheduler.scripts.scheduler
;; Describes queries expected by SchedulerInterfaceAgent

(defglobal
    ?*addedSchedulerStatements* = FALSE
    ?*SpeechTextOutput* = (reliesOn "agentland.text.SpeechTextOutput")
    ?*GrammarCenter* = (reliesOn "speech.GrammarCenter"))


(deffacts scheduler-state-info
  "Define some things we know about Schedulers."
  (introduce-agent (occupation "agentland.util.scheduler.Scheduler")
                   (name "scheduler")))

(reset) ; reset to load these facts


;;;;;;;;;;;;;;;;;;;
;; handle commands
;;;;;;;;;;;;;;;;;;;

(defrule doSendReminder
  "Send a reminder"
  ?command <- (startDo (args sendReminder ?time ?unit) (done FALSE))
  (name-agent-match (name "scheduler") (agent ?agent)
                    (interface "agentland.util.scheduler.Scheduler"))
  =>
  ; get text of message
  (call ?*SpeechTextOutput* outputText "What should the reminder say?")
  (bind ?message (call ?*GrammarCenter* getDictation))
  (printout t ?message " " ?time " " ?unit crlf)
  (bind ?did-it TRUE)
  (bind ?reminder (new agentland.util.scheduler.OneTimeReminder
                   (call ?*SpeechTextOutput* getAgentID)
                   outputText
                   ?message
                   ?time
                   ?unit))
  (bind ?reminderID (call ?agent addReminder ?reminder))
  (printout t "Reminder set: " ?message " " ?time " " ?unit crlf)
  (call ?*SpeechTextOutput* outputText
        (str-cat "OK, I will remind you in " ?time " " ?unit))
  (modify ?command
          (done TRUE)
          (finish-time (call (new java.util.Date) getTime)))
  (addReturnValue (str-cat "sendReminder " ?time " " ?unit) TRUE))


;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;


(defrule updateValidStatements-reminder
    (test (neq ?*addedSchedulerStatements* TRUE))
```

```
(getValidStatements)
=>
(addValidQueries (create$ ))
(addValidCommands
  (create$ "sendReminder <time>"))
(bind ?*addedSchedulerStatements* TRUE))
```

weather.clp

```
;; agentland.info.scripts.weather

(defglobal
    ?*addedWeatherStatements* = FALSE
    ?*SpeechTextOutput* = (reliesOn "agentland.text.SpeechTextOutput"))

(reset) ; reset to load these facts


;;;;;;;;;;;;;;;;;;
;; handle commands
;;;;;;;;;;;;;;;;;;

(defrule doReadWeather
  "Read today's weather"
  ?command <- (startDo (args readWeather) (done FALSE))
  =>
  (bind ?weather
        (call (new agentland.info.weather.WeatherFetcher) getWeather))
  (call ?*SpeechTextOutput* outputText (call ?weather tinyWeather))
  (modify ?command
          (done TRUE)
          (finish-time (call (new java.util.Date) getTime)))
  (addReturnValue "readWeather" TRUE))


;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;


(defrule updateValidStatements-weather
    (test (neq ?*addedWeatherStatements* TRUE))
    (getValidStatements)
    =>
    (addValidQueries (create$ ))
    (addValidCommands
      (create$ "readWeather"))
    (bind ?*addedWeatherStatements* TRUE))
```