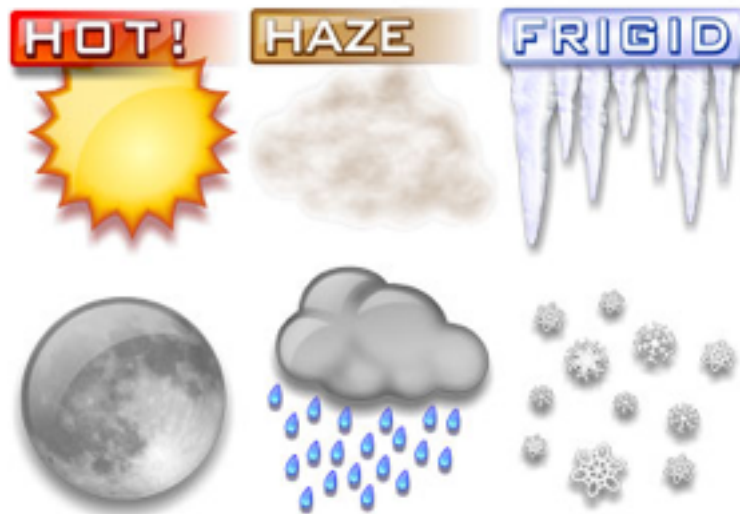


4005-750-01
Introduction to Artificial Intelligence
Environment Manager: Part 2

Kurt Alfred Kluever (kurt@klover.com)
Brian C. Renzenbrink (brenzenb@gmail.com)
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology

February 18, 2008



Abstract

A rule-based expert system has been developed that can autonomously govern a building's environment to optimize user comfort and energy consumption, whilst providing safety and monitoring functions. The expert system has been developed using the Java programming language and the Java Expert System Shell (JESS). Rules are stored as an external resource and can be modified in real time without requiring a rebuild of the entire project.

1 Product Testing

In order to test our Expert System, we designed a series of randomized general tests and a series of boundary tests.

1.1 Variable Definition

P_t = Preferred Temperature

P_h = Preferred Humidity

P_l = Preferred Illumination

S_t = Sensor Temperature

S_h = Sensor Humidity

S_l = Sensor Illumination

O_{he} = Heater Output

O_{ac} = A/C Output

O_{dh} = Dehumidifier Output

O_{hu} = Humidifier Output

O_{il} = Illumination Output

1.2 Boundary Tests

For our boundary tests, we tested that our system would not accept an invalid request for Occupants. For occupants, we used expert knowledge to set bounds on the values that could be requested. The goal behind these bounds was to prevent occupants from cheating the system by grossly understating or overstating their preferences in order to greater affect the aggregate preferences, and therefore affect the system. An occupant's preferences must meet the following requirements:

$$60.0^{\circ}F \leq P_t \leq 85.0^{\circ}F$$

$$0.0\% \leq P_h \leq 60.0\%$$

$$0 \leq P_l \leq 10$$

The test for these bounds is presented in the table below:

P_t	P_h	P_l	Result
40.0°F	50.0%	7	InvalidPreferenceException: Temperature must be between 60.0F and 85.0F.
70.0°F	80.0%	7	InvalidPreferenceException: Humidity must be between 0.0% and 60.0%.
70.0°F	60.0%	12	InvalidPreferenceException: Humidity must be between 0 and 0. ¹
90.0°F	80.0%	12	InvalidPreferenceException: Temperature must be between 60.0F and 85.0F. ²
70.0°F	80.0%	12	InvalidPreferenceException: Humidity must be between 0.0% and 60.0%. ³
70.0°F	50.0%	12	InvalidPreferenceException: Humidity must be between 0 and 0. ¹

After fixing the bug, the tests were run again. The table below shows the correct results:

P_t	P_h	P_l	Result
40.0°F	50.0%	7	InvalidPreferenceException: Temperature must be between 60.0F and 85.0F.
70.0°F	80.0%	7	InvalidPreferenceException: Humidity must be between 0.0% and 60.0%.
70.0°F	60.0%	12	InvalidPreferenceException: Illumination must be between 0 and 10.
90.0°F	80.0%	12	InvalidPreferenceException: Temperature must be between 60.0F and 85.0F.
70.0°F	80.0%	12	InvalidPreferenceException: Humidity must be between 0.0% and 60.0%.
70.0°F	50.0%	12	InvalidPreferenceException: Illumination must be between 0 and 10.

¹Indicates a bug in the program. The bug was due to sloppy copy and pasting of code. The fix was extremely simple to find and correct.

²Only the temperature exception is thrown because it is the first bound to be checked. If the temperature falls in the correct bounds, humidity is checked next, followed by illumination.

³Only the humidity exception is thrown because it is the first bound to be checked that is invalid. If the humidity falls in the correct bounds, illumination is checked next.

1.3 Random Tests

In the random testing, we tested five random scenarios. The scenarios were generated to fall within plausible bounds. These examples represent simulated real life conditions of an office building. The system performed as expected with no errors. The tests are summarized below in table form.

P_t	P_h	P_l	S_t	S_h	S_l	O_{he}	O_{ac}	O_{dh}	O_{hu}	O_{il}
72.0°F	37.0%	8	80.0°F	62.0%	0	0	8	10	0	8
70.0°F	39.0%	9	63.0°F	41.0%	4	7	0	2	0	9
69.0°F	41.0%	7	83.0°F	43.0%	3	0	10	2	0	7
69.0°F	44.0%	8	70.0°F	40.0%	10	0	1	0	4	8
71.0°F	35.0%	9	74.0°F	7.0%	9	0	3	0	10	9

1.4 Default Test

In this test, we wanted to test that if no occupants are present, that the system turns off all outputs (heater, a/c, humidifier, dehumidifier) and dim the lights to 1. More formally stated,

$$(numOccupants = 0) \Rightarrow O_{he} = 0, O_{ac} = 0, O_{dh} = 0, O_{hu} = 0, O_{il} = 1$$

The table below shows that the system performs as expected:

$numOccupants$	S_t	S_h	S_l	O_{he}	O_{ac}	O_{dh}	O_{hu}	O_{il}
0	80.0°F	62.0%	8	0	0	0	0	1
0	73.0°F	43.0%	3	0	0	0	0	1

1.5 Scalability Tests

In these tests, we wanted to make sure that our system would scale appropriately. We again used randomized values to simulate from 1 to 500,000 occupants and from 1 to 500,000 sensors. Below are the run times:

<i>numOccupants</i>	<i>numSensors</i>	Time
1	1	37ms
1	5000	212ms
5000	1	51ms
5000	5000	239ms
10000	10000	332ms
100000	100000	4206ms
500000	500000	OutOfMemoryError: Java heap space
500000	1	77ms
1	500000	OutOfMemoryError: Java heap space
500000	25000	908ms

This data validates that sensors are computationally more expensive than occupants. This can easily be explained because the rules engine must look at each sensor to issue warning or emergency messages, but it only looks at the single OccupantAggregator. This fits well with the proposed real life setups which suggest 1 sensor for every 20 occupants. Our system shows great scalability. Tests suggest that a building with half of a million occupants and 25,000 sensors (a 20:1 ratio) can be controlled in less than 1 second.

2 Independent Testing

We received the other team's project for the purpose of independent testing and comparison. For their expert system, they also chose to use Jess. Their system creates a GUI that allows the user to set the number of occupants, the number of rooms, the outside temperature, and the outside humidity. After setting the values, the user can choose to run the simulation and stop the simulation as needed. We decided to test their project using five realistic scenarios as well as testing perceived boundary cases.

2.1 Testing

In the random testing, we tested six random scenarios. The scenarios were generated to fall within plausible bounds. We entered random, plausible scenarios for the data in the random data set and selected to run the expert system. The system did not perform as expected. Below are the results:

User Defined Inputs				Results		
# People	# Rooms	Out Temp	Out Humidity	In Temp	In Humidity	Light
1	1	45	10	70	50	Off
1	10	45	10	70	50	Off
10	1	45	10	70	50	Off
10	10	45	10	70	50	Off
10	10	30	60	70	50	Off
10	10	90	60	70	50	Off

Note that the other team’s GUI appeared to show people moving in and out of rooms. However, this had no affect on the results and the results were the same for all rooms. Also all of the values (input and output) lacked units. We are assuming that the units were in °F and % humidity. The lights were always set to “Off”.

In our testing we deduced that their Jess rules engine was not firing properly. The GUI always contained the same values in each column, and the number of rooms defined the number of rows displayed. It seems that their system is intended to set the climate to the constant temperature of 70°F and 50% humidity with the lights off.

In the boundary testing, we tested for unreasonable values being input. We tested for extremely high heat and humidity as well as varying the number of occupants and rooms. Their system did not perform as expected:

User Defined Inputs				Results		
# People	# Rooms	Out Temp	Out Humidity	In Temp	In Humidity	Light
1	1	200	120	70	50	Off
1	10	200	120	70	50	Off
10	1	200	120	70	50	Off
10	10	200	120	70	50	Off
10	10	-30	-30	70	50	Off
10	10	-30	10	70	50	Off
10	10	10	-30	70	50	Off

Similar to the random testing, we concluded that their Jess rules engine was not firing. Once again it appears that the only value that affects output is the number of rooms shown for the simulation.

2.2 Screen Shots

Enter number of people:

Enter number of rooms:

Enter the outside temperature:

Enter the outside humidity:

Enter number of people:

Enter number of rooms:

Enter the outside temperature:

Enter the outside humidity:

Enter number of people:

Enter number of rooms:

Enter the outside temperature:

Enter the outside humidity:

Enter number of people:

Enter number of rooms:

Enter the outside temperature:

Enter the outside humidity:

Enter number of people:

Enter number of rooms:

Enter the outside temperature:

Enter the outside humidity:

Enter number of people:

Enter number of rooms:

Enter the outside temperature:

Enter the outside humidity:

Enter number of people:

Enter number of rooms:

Enter the outside temperature:

Enter the outside humidity:

Enter number of people:

Enter number of rooms:

Enter the outside temperature:

Enter the outside humidity:

Climate Control System

Room	# of People	Temperature	Humidity	Light
Room 0	0	70	50	Off

Climate Control System

Room	# of People	Temperature	Humidity	Light
Room 0	0	70	50	Off
Room 1	0	70	50	Off
Room 2	0	70	50	Off
Room 3	0	70	50	Off
Room 4	0	70	50	Off
Room 5	0	70	50	Off
Room 6	0	70	50	Off
Room 7	0	70	50	Off
Room 8	0	70	50	Off
Room 9	0	70	50	Off

2.3 Comparison

The other team's project had several good concepts. The use of a GUI was a good idea to allow for easy user interaction. However, the GUI was somewhat non-intuitive and lacked units for all of the input and output values. The notion of different climate zones or rooms was an idea that we had hoped to implement ourselves. However, they failed to properly implement different zones. The project also had the notion of time and they aimed at illustrating change over time in the environment. However, the only values that changed during the simulation were the number of people in the rooms. All climate values remained constant. Finally, the concept of outdoor readings affecting the indoor settings was a great idea.

If the project had been properly functioning (specifically the rules), the project could have been a great success. Unfortunately, the project failed to work. It failed to execute the obvious goals that they had tried to achieve.

In our testing, our project performed flawlessly. Our design was easily capable of controlling climate within a building based on user preferences and sensor readings. Our initial submission lacked a GUI and the notion of time. Therefore, in our second submission, we chose to implement these two necessary features.

3 Product Improvements

In Part 1 of this project, our system would accept multiple Occupants and aggregate their preferences to create a Goal State. It would then compare that Goal State to the current environment, as detected by the aggregate of multiple sensors placed throughout the building. When run, our system would output the changes that should be made to the environment controls at this instance to standard out.

In an effort to improve the efficiency and usefulness of our expert system, we have added a feature that allows changes to environment controls produced by our expert system to interact with the sensor readings over time. We chose to simulate changes in the environment at an increased rate. This was chosen so that changes in the environment would be immediately observable. Recognize that a real system would not affect the environment so rapidly.

In order to help the user visualize this, we have also created a GUI that will display these changes to the environment along with the controls that have been associated with them. The GUI allows the user to see and interpret all data at once through the intuitive line

graphs. This GUI is very useful in identifying that the system is reaching its goals without fail.

4 User Guide

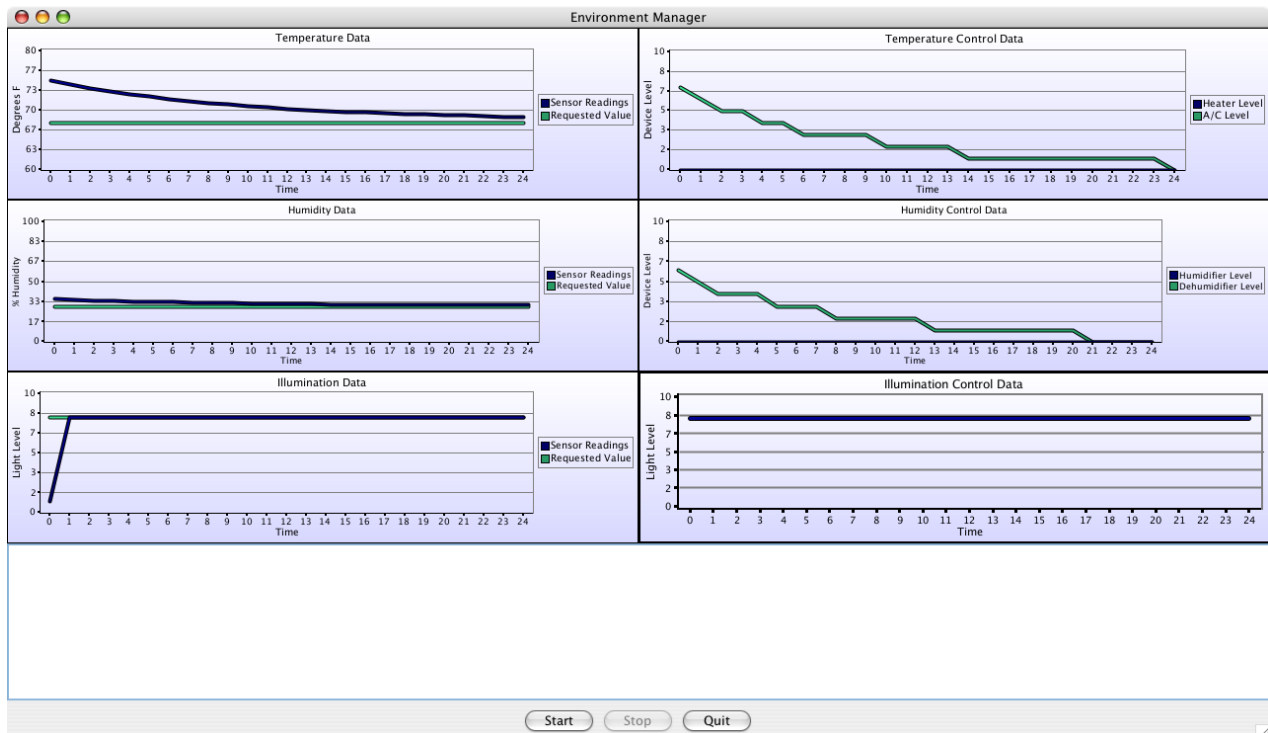
In order to run this project, please verify the rules file is named *rules.clp* and is in the same directory as the *EnvironmentManager.jar*. The project can be run using the following command and it will use random data for the inputs:

```
java -jar EnvironmentManager.jar
```

For testing purposes, the project can also be run with 6 optional parameters that specify the occupant's desired temperature, desired humidity, desired lighting level; the sensor's current temperature, current humidity, current light level. For example:

```
java -jar EnvironmentManager.jar 68 30 8 75 36 1
```

This will start the simulation with predefined values so that the user can see how the system sets the controls. Below is a screenshot of the completed simulation using the above arguments:



Warning or emergency messages appear in the large white box at the bottom of the GUI. An example of an incomplete simulation with emergency and warning messages can be created with the following command:

```
java -jar EnvironmentManager.jar 68 30 8 120 100 1
```

A screenshot of the above program running with the above argument is:

