Metaglue: An Agent Development System

Stephen Peters[1]
Krzysztof Gajos[2]
Kevin Quigley[3]

Metaglue version 0.5 , released August 2004

[1]portnoy@csail.mit.edu
[2]kgajos@csail.mit.edu
[3]quig@csail.mit.edu

This file documents the Metaglue Development Environment

Contents

Chapter 1

Overview

Metaglue is a Java-based system designed to make the development of distributed, software *agents* easy and straightforward. It can be used (and indeed, has been used) as the base infrastructure for controlling intelligent offices and conference rooms, creating a network of interconnected kiosks and providing resource management and personalization services in both the preceding environments. Included agents can control physical hardware (such as: lights, video projectors and drapes) through a variety of ways, including serial ports, X10 devices, bluetooth and direct communication with network appliances.

Agents in Metaglue can communicate with each other, collaborating whether on the same computer or across a network, across a wide variety of different hardware. Agents can also specify that they run on a specific computer, or that they run only in conjunction with other agents.[1] The configuration of each agent can be created and maintained based on the individual requirements, so resource management is also available. Metaglue also has facilities for agents to persistently store their own internal state, broadcast messages and events to any interested parties, and automatically restart if the host or computer suddenly stops functioning.

With the possibility of a large, distributed network of agents operating, the availability of resources changing constantly, people coming and going, and a good chance for hardware and software failure, Metaglue must be capable of responding to such a dynamic environment. There should be no need to stop all the current work, just to a new agent or device. Within the system, it is easy to insert or remove new agents in a running environment. It is also possible to change or swap an agent in an interaction without disrupting the other agents involved or even having them notice.

[1] For another description of the Metaglue system, see "Meeting the Computational Needs of the Metaglue System" in the 1999 *Proceedings on Managing Interactions in Smart Environments*.

Chapter 2

System Architecture

2.1 Agents to Applications

Agents run on top of the *Metaglue Virtual Machine* (MVM). Also known as *the MetaglueAgent* after its Java class name, the MVM must be running on every host computer where agents will be running. The MVM provides the environment for starting new agents, including information about the properties of the host computer.

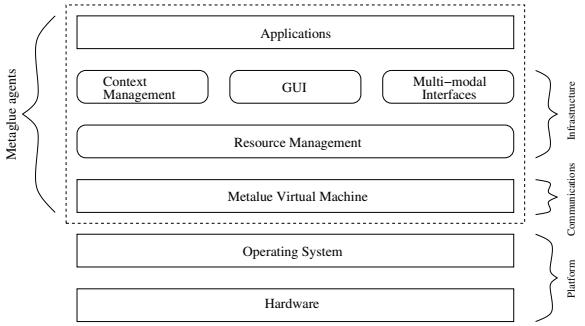

Figure 2.1: Layered Roles for Agents

At the base level, Metaglue agents sit on the Metaglue Virtual Machine (MVM) which in turn sits on the hardware and operating system platform. (see figure 2.1) The MVM provides communication, storage capability and failover should something disrupt the system. The agents communicate by exposing functionality to other agents running within the network. This deliberately under-specifies the capabilities of an agent, such that it can be anything from a software control for a light switch with no user interface to a full-featured email client.

Using these basic agents, we have constructed a *resource management* layer of organization. In addition to an arbitration mechanism, this layer provides a high-level resource discovery service. For example, using just the communication layer, an agent would need to know the exact name of a specific light controller agent to turn on a light. Using the resource management layer an agent can ask for light in a room in a general way or for a list of available lights or groups of lights. This layer is critical to system flexibility, particularly when agents need to move from one environment to another and to recover from system failures transparently.

On top of resource management structures, we have developed three parallel layers: *context management*, *graphical user interfaces*, and *multi-modal user interfaces*. The context management layer contains tools to observe and use contextual information. The graphical user interface (GUI) layer contains infrastructure to control graphical interfaces to applications. In particular, this layer is responsible for allowing applications

to adapt to available display resource. Alongside GUIs, the multi-modal interface layer contains methods for other modes of user input and output, such as gesture and speech recognition, and different audible or visible communication means.

2.2 Inter-Agent Communication

Communication between agents in Metaglue (see figure 2.2) is accomplished in two ways:

- Directly between two agents where one agent knows the identity of the other.
- Messages are broadcast from the sending agent to a notification agent who passes them on to other interested agents.

All agents need to register with the *Catalog* agent when they start up.[1] The Catalog serves as the discovery service for agents only, as opposed to a resource management system which can do agent-name based requests and more.

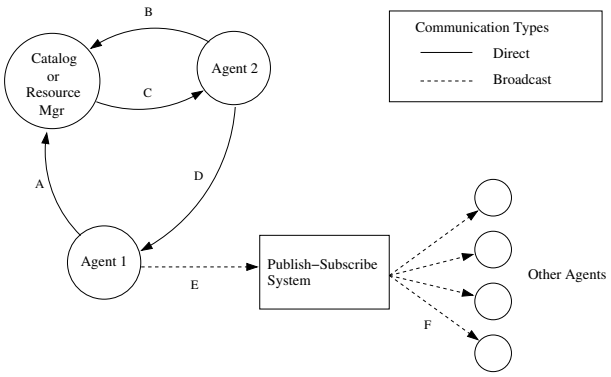

Figure 2.2: Communication in Metaglue

A When Agent 1 starts up it will register its existence with the Catalog.

B Agent 2 knows the full name of Agent 1, so it can request Agent 1 from the Catalog.

C The Catalog returns the contact information of Agent 1 to Agent 2.

D Agent 2 can then contact Agent 1 and request some action.

E The requested action initiates a broadcast message from Agent 1.

F This broadcast message is repeated (by the publish-subscribe system agents) to other agents who are unknown to Agent 1.

[1] The location of the Catalog agent is given to the MVM on initialization.

Chapter 3

Installation

Metaglue can be installed on any platform which is capable of running Sun's Java Platform: Standard Edition. Make sure you have Java version 1.4 or later. If your computer doesn't have Java, go to `http://java.sun.com/products` and get the Standard Edition for your target Operating System.

The support for saving an agent's state requires the use of a database management system (DBMS). MySQL[1] is the preferred DBMS for Metaglue. Both PostgreSQL[2] and Microsoft's Access DB[3] have some support in Metaglue, but it is still in development.

The latest Metaglue packages can be found on our web site's software page.[4] Several operating systems are supported. Follow the instructions in the specific package as they will be tailored to the specific operating system and have shortcuts for setting up the system.

3.1 Generic Installation

Installing Metaglue by hand is not hard. Within each package, there are a set of jar files (Java archives) which contain the classes used by Metaglue. Put these into any directory you like. To run Metaglue, your computer must be able to do two things:

1. Run Java. Make sure the Java executable file is found somewhere in your `PATH` environment variable.
2. Have the jar files mentioned previously be found in the `CLASSPATH` environment variable. It maybe necessary to create the `CLASSPATH` variable if it did not exist before.
3. Create the Metaglue database, after installing a DBMS, by running

   ```
   java edu.mit.aire.util.storage.BuildDatabase
   ```

3.2 Extra Installation

The prepackaged installations of Metaglue come with some extra setup which makes running the system a little easier. There are aliases (shortcuts in Windows) which will simplify what is needed to type to get the system running in the next chapter. "For a full listing, see the Appendix."

---

[1] `http://www.mysql.com`
[2] `http://www.postgresql.org`
[3] `http://www.microsoft.com/Office/Access/`
[4] `http://www.ai.mit.edu/projects/aire/software.shtml`

Chapter 4

Running Metaglue

There is one required piece to run the Metaglue system. Nothing can be done without the *Catalog*, which acts as the directory and registration service for the running agents. The Catalog prevents an agent from starting up in a second place if an agent with an identical name already exists and arbitrates when an agent tries to startup in two places at the same time.

Once the Catalog is running, a running Metaglue system can be created by starting one or more MVMs and any number of agents on any of the MVMs *provided they are all working with the same Catalog.*

4.1 Starting the Catalog

Before running any Metaglue agents, you have to start up the Catalog. This can be done by running the command

```
java edu.mit.aire.metaglue.core.MetaglueAgent -catalog
```

If you're trying to use multiple computers in the agent system, be sure to note the computer name (or IP address) where the catalog is running. When starting up agents, you will need to specify the location of this catalog.

If there are any problems starting up the catalog, most notably error messages that say that the `MetaglueAgent` class cannot be found, be sure that the `CLASSPATH` environment variable (see Installation) is set properly. For any other errors, see the FAQ `http://www.ai.mit.edu/projects/aire/FAQ.shtml`

The Metaglue system maintains information about the state of the agent environment in a database. To remove any old or stale information, you should start up the catalog with the `-purge` option:

```
java edu.mit.aire.metaglue.core.MetaglueAgent -catalog -purge
```

4.2 Starting an Agent

Once the catalog has been started, you can start up an agent by specifying a society, catalog, and agent name when the `MetaglueAgent` starts up:

```
java edu.mit.aire.metaglue.core.MetaglueAgent <society> <catalog host> <agent name>
```

Here, the *society* is a society name. The actual name here doesn't matter much; agents use them to communicate with each other, and they serve as a kind of name-space so that your agents can keep their information separate from a similar agent operating on behalf of someone else. A good society name to use might be your name, or your initials, or something else unique to you.

The *catalog host* is the name of a computer running the catalog (see Starting the Catalog). If all your agents run on the same computer, you can use `localhost` and the agent will look locally for the catalog, but for distributed environments you will need to correctly specify the name (or IP address) of the computer running the catalog.

Finally the *agent name* is the class name specifying an agent. A good test example is `edu.mit.aire.metaglue.debug.PowerTester` (see Debugging Agents), which will show a simple window with a text field in it when it starts up. You can eliminate the `edu.mit.aire.` prefix for the *agent name*; the `MetaglueAgent` will try a small set of prefixes before deciding that a given agent can or cannot be located.

4.3 Starting a Metaglue VM

If you just want to start a Metaglue Virtual Machine on a computer without starting up an agent (for example, if you need to prepare a computer to receive agents), you can do so by starting up the MVM with only the society and catalog host as arguments:

```
java edu.mit.aire.metaglue.core.MetaglueAgent <society> <catalog host>
```

Chapter 5

Configuring Agents

5.1 Specifying Agent Names

Most agents are specified simply through the class name for the agent, also known as the agent's *occupation*. For example, the `PowerTester` agent's occupation is `edu.mit.aire.metaglue.debug.PowerTester`, and this is the name that you use to specify that you want that agent up and running. The occupation is also used inside the agent's Java code to communicate with other agents.

However, if all agents were simply specified using the occupation, that would prevent the possibility of having more than one agent of a given class running at a time. Therefore, there are two other optional parameters that can be used – the *society* and the *designation*.

The society is designed to allow multiple users to share a catalog. Each user runs all his agents under a society name that he creates. To specify the society when referencing an agent, simply give the society name and the occupation, separated by a colon (`:`). For example, the `PowerTester` agent running in the `honcho` society would be specified by

```
honcho:edu.mit.aire.metaglue.debug.PowerTester
```

Once an agent is started with a given society name, that society is usually inherited by all agents that get relied upon subsequently.

In order to allow a single user to run more than one agent of a given occupation within a society (for example, an agent that controls a floor lamp and an agent that controls a desk lamp), the agents can also use a *designation*. This is another simple text string, which can be specified by appending a hyphen (`-`) and the string to the occupation. For example, if you want to start up two different projector controllers in the `honcho` society, they might be specified by

```
honcho:edu.mit.aire.metaglue.device.display.Projector-left
honcho:edu.mit.aire.metaglue.device.display.Projector-right
```

5.2 The Attribute Database

One way that agents can change their behavior is by reading *attributes*. Attributes are simple string or numeric configuration parameters for agents, which can take on different values in different societies, and are kept stored in a persistent database. Common attributes include port identifiers for agents that require serial ports, computer names for agents that need to run on specific hardware, and other configuration parameters.

There are two ways of populating the attribute database. The first is natural – when starting up an agent, if an attribute value can't be found in the database, a window will be displayed asking for the proper value in this society and offering a default value if it knows of one.

The second method is by editing the attribute database and specifying parameters directly by using SQL. The database can be accessed based on your local environment specification. This should only be done by someone experienced with databases and SQL.

Chapter 6

Agent Descriptions

There are several agents that ship with the Metaglue system. The more useful ones to start up from the command line or from the `PowerTester` agent are described in the following sections.

6.1 Debugging Agents

`edu.mit.aire.metaglue.util.log.LogMonitor`
: Once started, this agent monitors the catalog and displays a list of all agents that are running. If you use the mouse to click on an agent in the list, a separate window will appear and maintain a history of all the logging output generated by that agent.

`edu.mit.aire.metaglue.debug.PowerTester`
: Once started, this agent displays a simple text field. Typing the name of any other agent in this field will start up the given agent, and then display a list of functions for the agent. By clicking on one of these functions (and optionally typing in arguments), you receive direct control over the agent's functionality.

  If the `PowerTester` agent does not recognize the name you type into the text field, it will try again, adding a common prefix for the name (for example, `edu.mit.aire`, allowing you to type "metaglue.util.log.LogMonitor" for the `LogMonitor` agent). If that request fails, it will ask you if you want to set up an alias for the agent name. This provides a useful facility, where you can type in "lamp" as a shortcut name for `edu.mit.aire.metaglue.device.light.Light`. These aliased names will be remembered between `PowerTester` sessions.

`mess.Mess`
: The MESS acronym stands for Metaglue Expert System Shell – a version of the CLIPS expert system that has been grafted into the agent framework so that agents can be specified using rule-based code. Full documentation of the MESS language is available elsewhere.[1]

  The `Mess` agent starts up an interactive shell where MESS code can be evaluated and scripts executed. As such, it can serve as an interactive debugging platform for agents.

  For example, you can list all the agents running in the system by starting up the `Mess` agent and running

```
> (bind ?x (reliesOn "edu.mit.aire.metaglue.core.Catalog"))
> (printout t (call (?x enumerateAgents) toString) crlf)
```

[1] Or at least, it will be, as soon as it is written by someone...

6.2 Device Control Agents

It may be most useful to start these agents from the `PowerTester` agent, so that you have easy access to some of their methods. (see Debugging Agents.) Other ways of controlling them might be through speech or through the `Mess` environment.

6.2.1 X10 Controllers

X10 is a company specializing in home automation, and has created numerous devices for controlling lights, blinds, coffee makers, and lots of other neat things. Since one of the specialties of the Metaglue agent system is control of a room's devices, the ability to manipulate X10 devices is paramount. Many different X10 devices are scattered in the appropriate device directories, however the following are generic agents for talking to the X10 controllers themselves:

`edu.mit.aire.metaglue.device.x10.X10`
: This is the common interface for all X10 control agents. All of the agents below implement it.

`edu.mit.aire.metaglue.device.x10.TwoWayX10`
: A low level X10-controlling agent capable of talking to TwoWay X10 interface through the serial port.

`edu.mit.aire.metaglue.device.x10.LynX10`
: A low level X10 controlling agent capable of talking to LynX10 interface through the serial port.

6.2.2 Lights

Naturally, one of the most important devices to control in an automated room is the lights. These agents accomplish that task.

`edu.mit.aire.metaglue.device.light.X10Light`
: The agent for controlling non-dimmable (just on and off) lights through an X10 device. This agent relies on one of the X10 controllers (see X10 Controllers) to do its job.

`edu.mit.aire.metaglue.device.light.X10DimmableLight`
: An extension of `X10Light`, adding extra functionality for controlling the brightness level of a light. Naturally, the lamp must be connected to a special X10 unit that allows dimming.

`edu.mit.aire.metaglue.device.light.LightManager`
: This agent can control collections of `Light` (or `DimmableLight`) agents. Like all device managers, this agent allows other agents to control individual lights by name or all of them at once. It can also keep a control grammar up to date by dynamically updating it with the names of all available lights.

`edu.mit.aire.metaglue.device.light.LightManagerSpeech`
: This agent manages speech control of the light manager agent.

6.2.3 Projector

`edu.mit.aire.metaglue.device.display.Projector`
: Talks to a video projector through a serial port, and can toggle the power or change the video inputs. This one agent handles different projectors by loading different low-level serial controllers depending on the projector. The name of the low level controller class is passed to the agent through an attribute.

6.2.4 Projection Screen

`edu.mit.aire.metaglue.device.display.pscreen.X10ProjectionScreen`
: This agent controls motorized projection screens via X10, and can roll them up or down as requested.

`edu.mit.aire.metaglue.device.display.pscreen.ProjectionScreenSpeech`

: Speech interface to projection screen controlling agents

6.2.5 Drapes

`edu.mit.aire.metaglue.device.drapes.Drapes`
: Talks to drapes or blinds controlled either by an X10 device or an IR controller, and can open or close them using method calls. It can also query the current state o1f the drapes (or at least what the agent *thinks* is the current state).

6.3 Application Control Agents

To be written...

6.4 Meta Control Agents and Scripts

Agents and scripts described in this section are responsible for describing behaviors.

`edu.mit.aire.metaglue.device.display.pscreen.ProjectionScreenMeta`
: This agent listens for notifications from projector agents. In particular, it listens to notifications about change of on/off state of the projector. When a projector turns on, it pulls down the screen. When a projector is turned off, it pulls the screen up.

`edu.mit.aire.metaglue.software.perception.meta.Presentation`
: This script contains a number of rules encoding simple behaviors that a space should exhibit during a presentation. So there are rules for dimming the lights, closing the drapes, etc when the presentation starts... At the moment the rules are very simple. You are very welcome to edit them for yourself.

6.5 System Agents

These agents are typically started up behind the scenes, and don't need to be run from the command line. It might be useful to look at them in `PowerTester` (see Debugging Agents) to gain information about the

system, but for the most part they can be ignored unless you are trying to learn about the details of the system.

`edu.mit.aire.metaglue.core.Catalog`
: This is the catalog agent that all agents need to know about. Agents can ask the catalog to look up information on other agents, get a list of all the agents in the system, and gain other useful information.

`edu.mit.aire.metaglue.core.Notifier`
: A message broadcasting agent, used by several agents within the system. Agents can make requests for updates from the `Notifier` by using the `addSpy` method, and can broadcast information using the `notify` method. Information can be keyed with strings to limit the broadcast to only those interested.

Appendix A

Credits

Metaglue was first deployed as an intelligent infrastructure in the MIT AI Lab in 1997, after much hacking by many people.

A.1 Contributors

There have been many, many contributors to the Metaglue project. In a somewhat random order, here they are:

Michael Coen The first and foremost hacker, researcher, inventor, mentor, and father figure to us all. Instrumental in creating most of the ideas within the system, as well as creator of more systems and subsystems than can possibly be enumerated here.

Brenton Phillips Designer and implementor of the Metaglue core. Really likes the Java exception handling system, occasionally to excess.

Nimrod Warshawsky Creator of the EHA wrappers that allow the system to proceed swiftly without deadlocking while waiting for agents to start up, and then handle errors gracefully.

Krzysztof Gajos Hacker on many pieces of the system, and the original author of the Intelligent CD Player, an early generalized information retrieval system, and the resource management piece. Now the manager of the lab, making sure everyone is doing their job.

Luke Weisman Creator of the drug system, the speech subsystem, and instrumental on giving the system some attitude.

Stephen Peters MapViewer creation, database integration, core-level changes for proxy objects and classloading, and general comic relief. Currently working on expanding the system to handle multiple users and transfer of knowledge between multiple spaces.

Marion Groh Speech system work, and created an Eliza agent so that the room can psychoanalyze you while you lie on the sofa.

Kevin Wilson Initial integration with vision code, implementing nifty door cameras and such. Now working on microphone arrays and other neat toys.

Jared Smith-Mickelson More vision code, making neat systems that use binocular cameras to track people, and controlling cameras to follow you around the room.

Andy Chang Lots of useful lab work, getting equipment, and created a tutorial system so that people can walk into a new space and learn what is possible.

Katherine Koch Useful hacking on systems all over the place, and integrating the room with the START information retrieval system.

Deb Dasgupta Worked on replicating agent state, integrating tablet computers, and creating scheduling systems.

Nicholas Hanssens Restructured logging code, created the LogMonitor agent, and built a system for controlling Windows programs through agent control. Redesigning the speech system to allow for different speech engines to be plugged in at will.

Ruben Brown Helped Stephen disrupt meetings, various modifications and implemented an information management system.

Peter Finin Worked on the first pass of the Metaglue Expert System Shell (MESS), made modifications to Luke's drug system.

Tyler Horton Created the Boggle agent, so we can play games.

Kevin Quigley Our staff programmer, who's done lots of invaluable work trying to clean up our source tree.

Stephanie Chiou

Lin Wu

Dan Aguayo

Frank Bentley

Amit Bhavsar

Ed Tolson

Ajay Kulkarni

Matthew Mishrikey

Alice Oh

Esther Yoo

Gautam Reddy

Fred Choi

Rattapoom "Pipe" Tuchinda

... keep adding people here...

A.2 Release History

0.1 First release designed for the actual public to see, including speech systems and a few simple agents for lamps, drapes, and projectors.

0.2 Deployment of redesigned device agents and resource management. Added PowerPoint agent.

0.3 Added dimmable light agents and speech control over drapes. More stable resource management.

0.4 Major merge, incorporating most of a year's improvements in the infrastructure. Includes remote classloading,

Agent Index

Concept Index