MIT PROJECT OXYGEN
PERVASIVE, HUMAN-CENTERED COMPUTING

Programming with Metaglue

How to create basic agents

LCS

Metaglue Overview – Basic Capabilities

- On-demand agent startup
- Automatic restarting of agents
- Direct call or publish-subscribe communication
- Service mapping
- Customization (Attributes)
- Persistent storage (Persistent Map, Icebox)
- Interfaces: speech, GUI, web

Intelligent Room

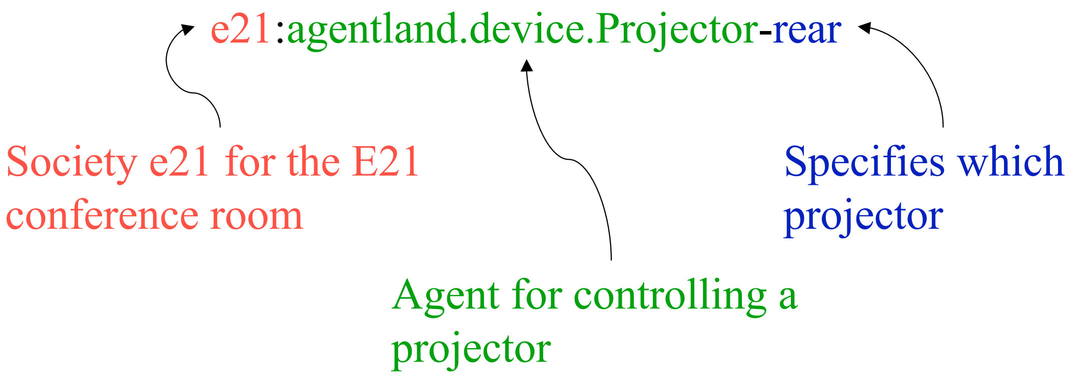
Agent Naming

- **Society – specific to people, spaces and groups**
- **Occupation – agent's function as Java interface name**
  - agentland.device.Projector,
  - agentland.software.StartInterface
- **Designation – to differentiate among various instances of the same agent within a society**

society : occupation - designation

Intelligent Room

Agent Naming – Example


e21:agentland.device.Projector-rear
Society e21 for the E21 conference room
Specifies which projector
Agent for controlling a projector

Intelligent Room

MIT PROJECT OXYGEN
PERVASIVE, HUMAN-CENTERED COMPUTING

Writing a Basic Agent

L C S

lab @ MIT

MIT PROJECT OXYGEN
PERVASIVE, HUMAN-CENTERED COMPUTING

Writing Metaglue Agents
File Naming Conventions

- Two files: the agent + the interface

For agent `agentland.device.display.Projector`:

- Interface:
  `agentland/device/display/Projector.java`
- Agent:
  `agentland/device/display/ProjectorAgent.java`

Intelligent Room

Why separate files?

- **The name of an object is not the object itself in RMI**
- **The Interface declares the *`name`* of the agent and what methods are available to other agents**
  - Some methods available through inheritance
- **The Agent is the fully implemented *`class`* object**

Intelligent Room

Writing Metaglue Agents

The most basic agent interface

```
package newbie.tutorial;

import metaglue.*;
import java.rmi.*;
import agentland.resource.*;

public interface Basic extends Managed {

} // Basic
```

The interface is of type interface

The most basic agent

```
package newbie.tutorial;

import metaglue.*;
import java.rmi.*;
import agentland.resource.*;

public class BasicAgent extends ManagedAgent implements Basic {

    public BasicAgent() throws RemoteException {

    }

} // BasicAgent
```

Intelligent Room

Writing Metaglue Agents

The most basic agent interface

```
package newbie.tutorial;

import metaglue.*;
import java.rmi.*;
import agentland.resource.*;

public interface Basic extends Managed {

} // Basic
```

The most basic agent

```
package newbie.tutorial;

import metaglue.*;
import java.rmi.*;
import agentland.resource.*;

public class BasicAgent extends ManagedAgent implements Basic {

    public BasicAgent() throws RemoteException {

    }

} // BasicAgent
```

The agent is of type class and will always implement the interface for which it is named

Intelligent Room

Writing Metaglue Agents

The most basic agent interface

```
package newbie.tutorial;

import metaglue.*;
import java.rmi.*;
import agentland.resource.*;

public interface Basic extends Managed {

} // Basic
```

The most basic agent

```
package newbie.tutorial;

import metaglue.*;
import java.rmi.*;
import agentland.resource.*;

public class BasicAgent extends ManagedAgent implements Basic {

    public BasicAgent() throws RemoteException {

    }

} // BasicAgent
```

The basic packages you always have to import

Intelligent Room

Writing Metaglue Agents

The most basic agent interface

```
package newbie.tutorial;

import metaglue.*;
import java.rmi.*;
import agentland.resource.*;


public interface Basic extends Managed {

} // Basic
```

This is where the **ManagedAgent** lives

The most basic agent

```
package newbie.tutorial;

import metaglue.*;
import java.rmi.*;
import agentland.resource.*;

public class BasicAgent extends ManagedAgent implements Basic {

    public BasicAgent() throws RemoteException {

    }

} // BasicAgent
```

ManagedAgent is a superclass of all agents capable of communicating with resource managers. Most of our agents now extend ManagedAgent.

Intelligent Room

Writing Metaglue Agents

The most basic agent interface

```
package newbie.tutorial;

import metaglue.*;
import java.rmi.*;
import agentland.resource.*;


public interface Basic extends Managed {

} // Basic
```

The most basic agent

```
package newbie.tutorial;

import metaglue.*;
import java.rmi.*;
import agentland.resource.*;

public class BasicAgent extends ManagedAgent implements Basic {

    public BasicAgent() throws RemoteException {

    }

} // BasicAgent
```

The constructor, as well as all exported methods (i.e. the ones specified in the interface) have to either throw **`RemoteException`**, or this exception has to be caught inside the method. It's an RMI thing.

Intelligent Room

Writing Metaglue Agents

The second most basic agent interface

```
package newbie.tutorial;

import metaglue.*;
import java.rmi.*;
import agentland.resource.*;


public interface Basic extends Managed {

         public void tellMe() throws RemoteException;

} // Basic
```

An exported method is thus declared in an interface…

Intelligent Room

Writing Metaglue Agents

The second most basic agent

```
package newbie.tutorial;

import metaglue.*;
import java.rmi.*;
import agentland.resource.*;

public class BasicAgent extends ManagedAgent implements Basic {

    public BasicAgent() throws RemoteException {

    }

    public void tellMe() throws RemoteException {
       log("I am " + getAgentID());
       log("My society is " + getSociety());
       log("My designation is " + getDesignation();

       log("I am running on " + whereAreYou());
    }

} // BasicAgent
```

An exported method is thus declared inside an agent itself…

Intelligent Room

Writing Metaglue Agents

The second most basic agent

```
package newbie.tutorial;

import metaglue.*;
import java.rmi.*;
import agentland.resource.*;

public class BasicAgent extends ManagedAgent implements Basic {

    public BasicAgent() throws RemoteException {

    }

    public void tellMe() throws RemoteException {
       log("I am " + getAgentID());
       log("My society is " + getSociety());
       log("My designation is " + getDesignation());

       log("I am running on " + whereAreYou());
    }

} // BasicAgent
```

Primitives that allow the agent to find out about its own identity

We will talk about logs later…

Fundamental Metaglue Primitives

`reliesOn()` returns a pointer to proxy representing an instance of the agent with specified `AgentID`; if necessary, the agent is first started.

- **Agent** `reliesOn`**(AgentID aid)**
- **Agent** `reliesOn`**(String occupation)**
- **Agent** `reliesOn`**(String occupation, Object designation)**

`reliesOn` is for direct communication

- **void** `tiedTo`**(String hostName)**
- **void** `tiedTo`**(AgentID anotherAgent)**
- **void** `tieToDesignation`**()**

`tiedTo()` should only be called in the constructor! It ensures that the agent runs on a particular machine or on the same VM as another agent.

Fundamental Metaglue Primitives

`reliesOn()` returns a pointer to proxy representing an instance of the agent with specified `AgentID`; if necessary, the agent is first started.

- **Agent** `reliesOn`**(AgentID aid)**
- **Agent** `reliesOn`**(String occupation)**
- **Agent** `reliesOn`**(String occupation, Object designation)**

These two methods take the society from the current agent

- **void** `tiedTo`**(String hostName)**
- **void** `tiedTo`**(AgentID anotherAgent)**
- **void** `tieToDesignation`**()**

`tiedTo()` should only be called in the constructor! It ensures that the agent runs on a particular machine or on the same VM as another agent.

Writing Metaglue Agents – reliesOn()

The not-so basic agent

```
package newbie.tutorial;

import metaglue.*;
import agentland.resource.*;
import java.rmi.*;

public class NotSoBasicAgent extends ManagedAgent implements NotSoBasic {

    Basic basic;

    public NotSoBasicAgent() throws RemoteException {
          basic = (Basic) reliesOn( Basic.class );
    }

    public void test() throws RemoteException {
          log( "calling tellMe() from the basic agent:" );
          basic.tellMe();
    }

} // BasicAgent
```

Writing Metaglue Agents – reliesOn()

The not-so basic agent

Note that the whole `reliesOn` process happens in terms of interfaces and not actual agents. What you get back from `reliesOn` is an object that implements the same interface as the agent but you do not get the agent itself!

```
package newbie.tutorial;

import metaglue.*;
import agentland.resource.*;
import java.rmi.*;

public class NotSoBasicAgent extends ManagedAgent implements NotSoBasic {

    Basic basic;

    public NotSoBasicAgent() throws RemoteException {
          basic = (Basic) reliesOn( Basic.class );
    }

    public void test() throws RemoteException {
          log( “calling tellMe() from the basic agent:” );
          basic.tellMe();
    }

} // BasicAgent
```

Intelligent Room

Writing Metaglue Agents – reliesOn()

The not-so basic agent

```
package newbie.tutorial;

import metaglue.*;
import agentland.resource.*;
import java.rmi.*;

public class NotSoBasicAgent extends ManagedAgent implements NotSoBasic {

    Basic basic;

    public NotSoBasicAgent() throws RemoteException {
          basic = (Basic) reliesOn( Basic.class );
    }

    public void test() throws RemoteException {
          log( “calling tellMe() from the basic agent:” );
          basic.tellMe();
    }

} // BasicAgent
```

But you talk to agents as if they were local objects…

Intelligent Room

Logging Messages in Metaglue

- `Better than System.out.println()`

- `void log(int logLevel, String message)`
- `void log(String logLevel, String message)`
- `void log(String message)`
  (defaults to `log("INFO", message)`)

- **Log levels:**

As ints:	*String shortcuts:*
`LogStream.DEBUG`	`"DEBUG"`
`LogStream.INFO`	`"INFO"`
`LogStream.WARNING`	`"WARNING"`
`LogStream.ERROR`	`"ERROR"`
`LogStream.CRITICAL`	`"CRITICAL"`

More on Logging

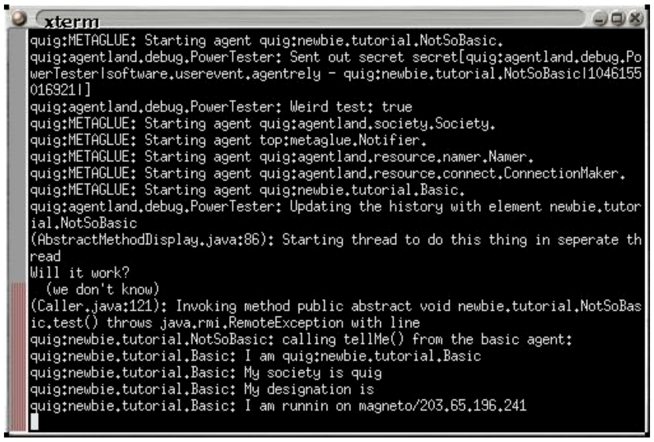
- **You can specify in your agent what kind of messages from a given agent should appear on the console window:**

  void `setLogLevel(int logLevel)`

  `Example:`

```
public BasicAgent() throws RemoteException {
      setLogLevel(LogStream.DEBUG);
 }
```

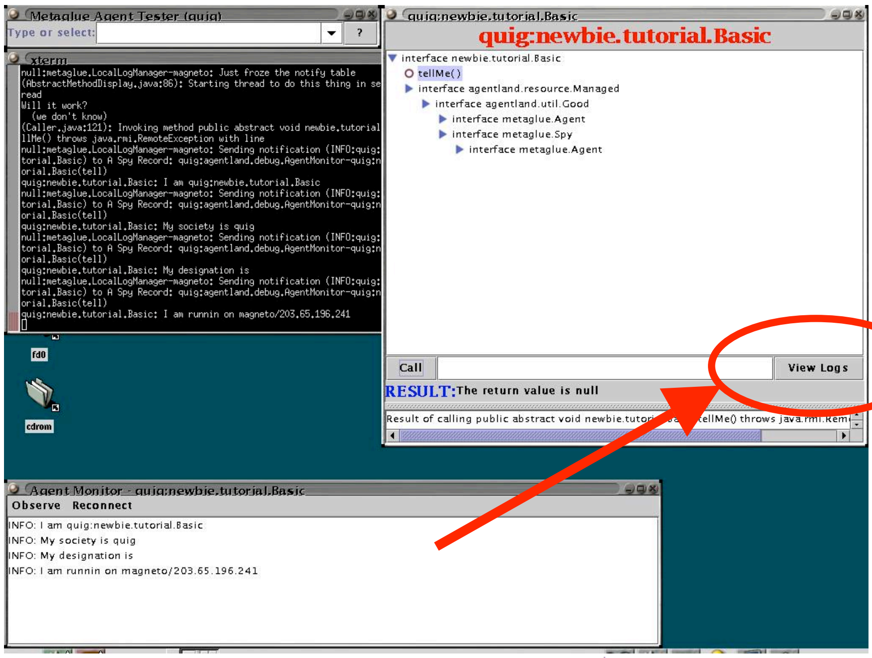
Why bother with Logging?



```
quig:METAGLUE: Starting agent quig:newbie.tutorial.NotSoBasic.
quig:agentland.debug.PowerTester: Sent out secret secret[quig:agentland.debug.Po
werTester|software.userevent.agentrely - quig:newbie.tutorial.NotSoBasic|1046155
016921|]
quig:agentland.debug.PowerTester: Weird test: true
quig:METAGLUE: Starting agent quig:agentland.society.Society.
quig:METAGLUE: Starting agent top:metaglue.Notifier.
quig:METAGLUE: Starting agent quig:agentland.resource.namer.Namer.
quig:METAGLUE: Starting agent quig:agentland.resource.connect.ConnectionMaker.
quig:METAGLUE: Starting agent quig:newbie.tutorial.Basic.
quig:agentland.debug.PowerTester: Updating the history with element newbie.tutor
ial.NotSoBasic
(AbstractMethodDisplay.java:86): Starting thread to do this thing in seperate th
read
Will it work?
  (we don't know)
(Caller.java:121): Invoking method public abstract void newbie.tutorial.NotSoBas
ic.test() throws java.rmi.RemoteException with line
quig:newbie.tutorial.NotSoBasic: calling tellMe() from the basic agent:
quig:newbie.tutorial.Basic: I am quig:newbie.tutorial.Basic
quig:newbie.tutorial.Basic: My society is quig
quig:newbie.tutorial.Basic: My designation is
quig:newbie.tutorial.Basic: I am runnin on magneto/203.65.196.241
```

- In a distributed system, the console/launcher window can be the standard out <stdout> for many agents
- These logs will be very confusing to use if you want to track the progress of a particular agent

Viewing Logs – *agentland.debug.PowerTester*



Metaglue Agent Tester (quig)
Type or select:
?
xterm
fd0
cdrom
quig:newbie.tutorial.Basic
Call
View Logs
Agent Monitor - quig:newbie.tutorial.Basic
Observe
Reconnect

```
null:metaglue.LocalLogManager-magneto: Just froze the notify table
(AbstractMethodDisplay.java:86): Starting thread to do this thing in se
read
Will it work?
  (we don't know)
(Caller.java:121): Invoking method public abstract void newbie.tutorial
llMe() throws java.rmi.RemoteException with line
null:metaglue.LocalLogManager-magneto: Sending notification (INFO:quig:
torial.Basic) to A Spy Record: quig:agentland.debug.AgentMonitor-quig:n
orial.Basic(tell)
quig:newbie.tutorial.Basic: I am quig:newbie.tutorial.Basic
null:metaglue.LocalLogManager-magneto: Sending notification (INFO:quig:
torial.Basic) to A Spy Record: quig:agentland.debug.AgentMonitor-quig:n
orial.Basic(tell)
quig:newbie.tutorial.Basic: My society is quig
null:metaglue.LocalLogManager-magneto: Sending notification (INFO:quig:
torial.Basic) to A Spy Record: quig:agentland.debug.AgentMonitor-quig:n
orial.Basic(tell)
quig:newbie.tutorial.Basic: My designation is
null:metaglue.LocalLogManager-magneto: Sending notification (INFO:quig:
torial.Basic) to A Spy Record: quig:agentland.debug.AgentMonitor-quig:n
orial.Basic(tell)
quig:newbie.tutorial.Basic: I am runnin on magneto/203.65.196.241
```

quig:newbie.tutorial.Basic

```
interface newbie.tutorial.Basic
  tellMe()
  interface agentland.resource.Managed
    interface agentland.util.Good
      interface metaglue.Agent
      interface metaglue.Spy
        interface metaglue.Agent
```

RESULT: The return value is null

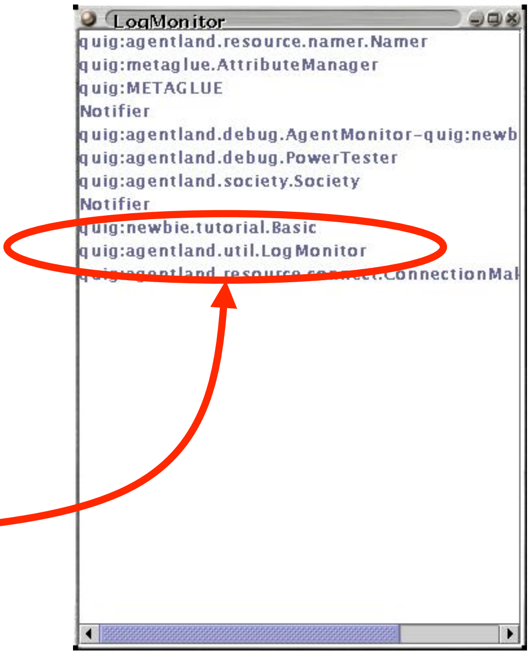
Result of calling public abstract void newbie.tutori[illegible]tellMe() throws java.rmi.Rem

```
INFO: I am quig:newbie.tutorial.Basic
INFO: My society is quig
INFO: My designation is
INFO: I am runnin on magneto/203.65.196.241
```

Viewing Logs – *agentland.util.LogMonitor*

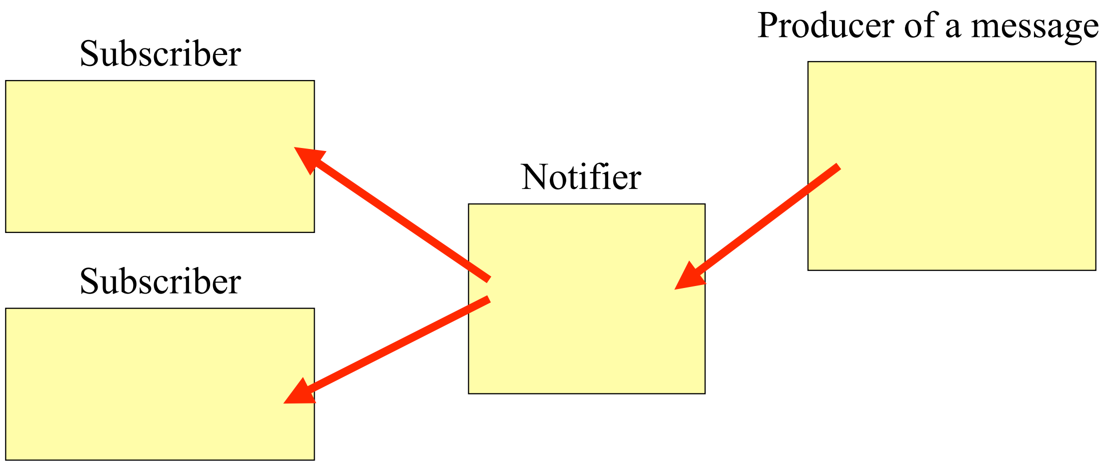
- The *LogMonitor* agent will bring up the same logging window as in the previous slide, but it does not need to use the *PowerTester* agent
- *LogMonitor* will list all agents which are currently running on the catalog currently in use

Even itself !



```
LogMonitor
quig:agentland.resource.namer.Namer
quig:metaglue.AttributeManager
quig:METAGLUE
Notifier
quig:agentland.debug.AgentMonitor-quig:newb
quig:agentland.debug.PowerTester
quig:agentland.society.Society
Notifier
quig:newbie.tutorial.Basic
quig:agentland.util.LogMonitor
quig:agentland.resource.connect.ConnectionMa
```

Sending and Receiving Messages – *metaglue.Notifier*



Anatomy of a Message

- **Messages are represented by instances of the object "Secret"**
  - Name
    device.light.stateUpdate.on
  - Details – any `Serializable` object
  - Source – `AgentID` of the sender
  - Time stamp – the time when the secret was first created
    * **based on the clock of the machine where the sender is located**

Naming of Messages

- **Names based on the Agent's *full heirarchical name***
  - For the agent named ***`device.Light`***
    ***`device.Light.stateUpdate.on`**
    ***`device.Light.stateUpdate.off`**

- **When you subscribe to `device.Light` you will receive `device.Light.stateUpdate` messages as well**
  - The same as subscribing to **`device.Light.*`**

- **When you subscribe to `device.*.stateUpdate`, you will receive state updates from all devices**

- **Subscribing to notifications should happen in the `startup()` method**

Subscribing to Notifications

```
package newbie.tutorial;

import metaglue.*;
import agentland.resource.*;
import java.rmi.*;

public class BasicAgent extends ManagedAgent implements Basic {

    public BasicAgent() throws RemoteException {
          addSpy( "tutorial.basic.StateUpdate" );
    }

    public void tell( Secret s ) throws RemoteException {
          if ( s.isA( "tutorial.basic.StateUpdate" ) )
             log( "Received new state " + s.details() );
    }


} // BasicAgent
```

Processing notifications
tell() is the default method for processing notifications

Subscribing to Notifications

```
package newbie.tutorial;

import metaglue.*;
import agentland.resource.*;
import java.rmi.*;

public class BasicAgent extends ManagedAgent implements Basic {

    public BasicAgent() throws RemoteException {
    }

    public void tell( Secret s ) throws RemoteException
        if ( s.isA( "tutorial.basic.StateUpdate" ) )
        log( "Received new state " + s.details() );
    }

    public void startup () {
        addSpy( "tutorial.basic.StateUpdate" );
    }

} // BasicAgent
```

Check what kind of message has been received before working with it

Subscribing to a family of notifications

Subscribing to Notifications – cont.

```
package newbie.tutorial;

import metaglue.*;
import agentland.resource.*;
import java.rmi.*;

public class BasicAgent extends ManagedAgent implements Basic {

    public BasicAgent() throws RemoteException {
    }

    public void action( Secret s ) throws RemoteException {
        if ( s.isA( "tutorial.basic.Action" ) )
        log( "Received an action notification " + s.details() );
    }

    public void startup () {
        addSpy( "tutorial.basic.Action", "action" );
    }


} // BasicAgent
```

Processing notifications through a custom method

Specifying the method to process notifications

Intelligent Room

Sending Notifications

```
package newbie.tutorial;

import metaglue.*;
import agentland.resource.*;
import java.rmi.*;

public class BasicAgent extends ManagedAgent implements Basic {

    public BasicAgent() throws RemoteException {
    }

    public void doMyThing() throws RemoteException {
      // do something
      Object stateObject = getState();

      notify( "tutorial.basic.StateUpdate", stateObject );

    }


} // BasicAgent
```

Sending a notification

Intelligent Room

Building Agents

- These can be run from `~/metaglue` or the source code area in `~/metaglue/newbie/tutorial/`
- Compile all of the java source files
  - **`make javac`**

    `*Remember, Java is NOT Python.  You must recompile after making changes!`
- `<edit to fix errors>`
- Compile the implementation files
  - **`make rmic`**

Running Agents

- **First start the catalog:**

  `mg_catalog [-purge]`

  `purge` will remove any previous maps and registered agents from the database when it starts the catalog. Only one of these is allowed on a computer.

- **Then start a Metaglue platform:**

  `agent` *`society catalogHost`* `[agent name]`

  Any agent can be started by providing the agent's name (the package interface. This will never end with "Agent")

  Not including an agent namewill start an empty Metaglue platform ready to receive agents.

- **or the agent tester:**

  `mg_agent` *`society catalogHost`* `agentland.debug.PowerTester`

Dialog boxes

- **Many messages pop up asking for values. These are the part of the customization of Metaglue through remembered attributes**
- **The defaults for most of them are fine.**
- **Those that don't have defaults:**
  - `username` for `agentland.society.Society`
    - **None needed for the class, but enter your name if you like.**
  - Others will be particular to the agents you are running. See the class material for information on those.

Intelligent Room

Statistics on Metaglue

- **10 Tons of fun:**
  - There are over 450 agents that exist within Metaglue
  - Between 50 and 80 agents are running the intelligent room
  - You are using more than 10 agents just while running the X10BasicLightControl
    - **Test it! Use `agentland.util.LogMonitor`**
- **Metaglue has been in development since 1998**
- **The system is used in several offices and homes including the office of the AI lab director, Rodney Brooks**
- **There are 2 full spaces at MIT (a 3rd is coming soon!) and one space in Australia running Metaglue**
  - Why not get your own?

Intelligent Room