

# Mica: Users Guide

**Matthew John McGill** <mmcgill@cse.unsw.edu.au>  
**James Henry Westendorp** <jhw@cse.unsw.edu.au>  
**Mohammed Waleed Kadous** <waleed@cse.unsw.edu.au>  
**Claude Anthony Sammut** <claudio@cse.unsw.edu.au>

---

## **Mica: Users Guide**

by Matthew John McGill, James Henry Westendorp, Mohammed Waleed Kadous, and Claude Anthony Sammut  
Copyright © 2003-2008 School of Computer Science and Engineering, UNSW & Matthew McGill & James Henry Westendorp & Mohammed Waleed Kadous & Claude Sammut

### **Licensing**

Mica is released under the GNU Lesser General Public License(LGPL) version 3 or later. You should have received a copy of the license with Mica. For more information regarding the LGPL see <http://www.gnu.org/licenses/lgpl.html>.

---

# Table of Contents

1. Introduction .....	1
Example Interaction .....	1
Mica Design .....	2
2. Running Mica .....	3
3. Mobs .....	4
Mob Types .....	4
Mob Persistence .....	4
4. Agents .....	5
Agent Functions .....	5
handleNewMob(Mob) .....	5
handleDeletedMob(Mob) .....	5
handleReplacedMob(Mob, Mob) .....	5
handleTypeManagerChanged() .....	5
setTransport(AgentTransport) .....	5
getTransport() .....	5
init(MicaProperties) .....	5
terminate() .....	6
Agent Transport .....	6
connect(String) .....	6
disconnect() .....	6
register(String) .....	6
unregister(String) .....	6
getTypeManager() .....	6
writeMob(Mob) .....	6
readMob(String) .....	6
deleteMob(String) .....	6
replaceMob(Mob) .....	6
mobSearch(String) .....	7
isConnected() .....	8
getAgentName() .....	8
setMessageHandler(MessageHandler) .....	8
synchronizedWriteMob(Mob, long) .....	8
5. Blackboards .....	9
Declaring Mob Types .....	9
6. MicaRunner .....	10
Interface .....	10
Configuration .....	11
7. Mica Security .....	12
SSL Connections .....	12
Secure XML Protocol .....	12
Verifying Agents .....	13
Restricting Agent Actions .....	13
8. Tools .....	15
DefaultAgents .....	15
SimpleAgentFrame .....	15
GUIAgent .....	15
LogDebugger .....	16
MobMaker .....	17
PDARunner .....	18
PDARunner Configuration .....	18
ProxyAgent .....	19
Services .....	19
Bibliography .....	20

---

## List of Figures

1.1. Example Weather Interaction .....	1
1.2. Agent Connections to a Blackboard .....	2
3.1. Example Mob Type Hierarchy .....	4
5.1. Example Type Definitions .....	9
6.1. MicaRunner User Interface .....	10
6.2. Example MicaRunner Configuration File .....	11
7.1. Example Agent Verification File .....	13
7.2. Example Agent Rights File .....	14
8.1. LogDebugger Interface .....	16
8.2. MobMaker Interface .....	17
8.3. PDARunner Interface .....	18
8.4. Example PDARunner Configuration File .....	18

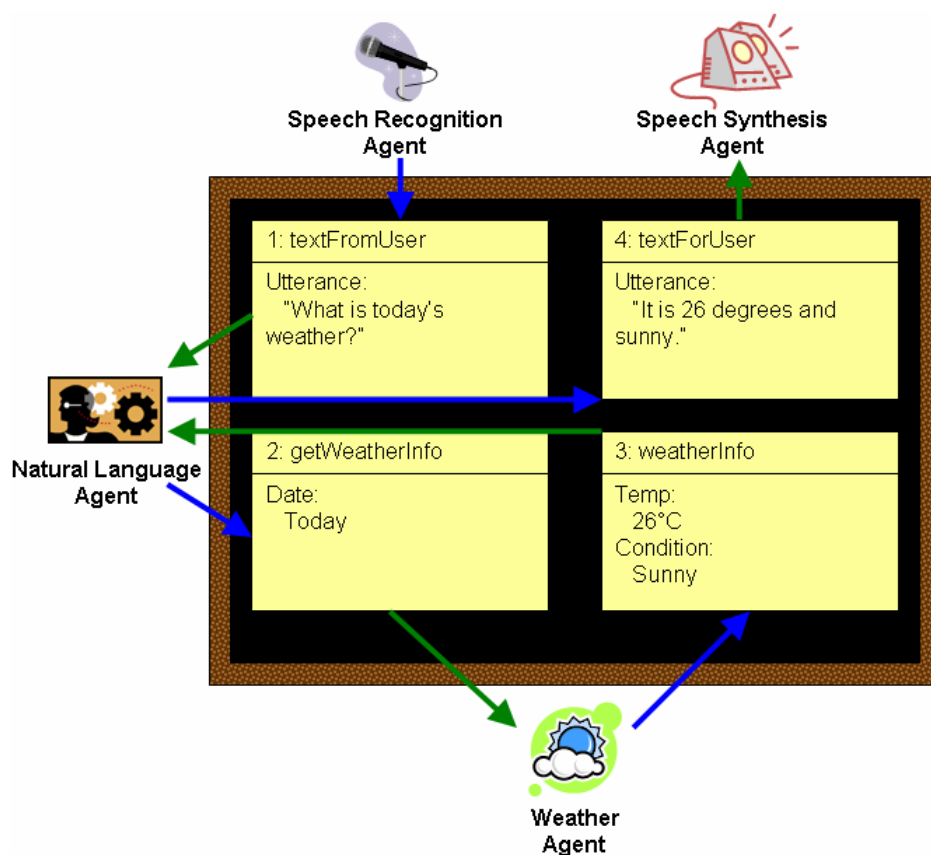
# Chapter 1. Introduction

MICA is a toolkit for use in the development of applications that involve different modes of interaction and diverse autonomous agents. MICA stands for Multimodal Interagent Communication Architecture <sup>1</sup>.

MICA is built around a blackboard paradigm. Agents connect to the blackboard and register to be notified when certain type of information are posted. Agents can then post their own messages to the blackboard and any agents registered for the message will be notified. When an agent is notified of a message it can process the information as it sees fit and if needed post its own message in response.

This manual details how to construct Mica agents for use in a variety of applications.

## Example Interaction



**Figure 1.1. Example Weather Interaction**

Shown in Figure 1.1 is how a simple spoken interaction with a system could take place over Mica. This interaction involves four independent agents communicating using four messages posted to the blackboard. The sequence of the interaction is:

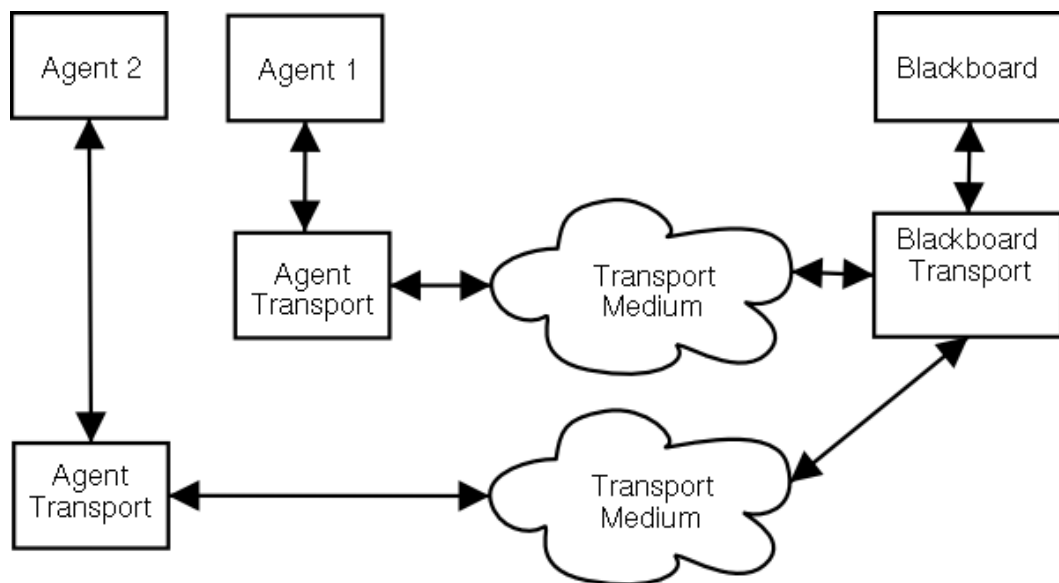
1. Upon recognition of a segment of speech a speech recognition agent posts the recognized speech.
2. When notified of the new speech text the natural language agent decodes the text into a command to get today's weather information.
3. In response to the command for weather information a weather agent posts today's temperature and condition.

<sup>1</sup>Actually, this is the latest version of the acronym. Previously it used to stand for "Multimodal Internet Conversation Architecture", but then it was realised (a) it wasn't restricted to the internet (b) it wasn't restricted to conversation either. But the name has stuck, in any case.

4. The natural language agent receives the weather information for today and transforms it into a standard English sentence which it posts in response.
5. When notified of text to be returned to the user a speech synthesis agent synthesizes the text to audio data and plays it to the user.

One of Mica's significant benefits is its ability to separate the functional components of an application from the interface components. In the example in Figure 1.1 the weather agent responds to a command to get today's weather generated by the natural language agent. This command could just as easily come from a graphical interface and the weather agent would respond precisely the same.

## Mica Design



**Figure 1.2. Agent Connections to a Blackboard**

Mica is designed so that the blackboard and the agents that connect to it can be kept as separate as possible. This is done by funnelling all communications between the blackboard and agents through a transport layer.

As long as the blackboard and agents use the same transport layer they can communicate. The most common one used and provided with Mica distribution is an XML over TCP transport layer.

---

## Chapter 2. Running Mica

In the Mica distribution you will find the file `mica.jar`. If you wish to use Mica in an application you simple need to add this file to the application's classpath. If the application needs to use the `SQLBlackboard` then you will also need to add a compatible SQL database will also need to be available. This can be accomplished by adding the file `hsqldb.jar`<sup>1</sup> provided with the distribution to the classpath as well.

There are C libraries available for creating agents using XML over TCP agent transports that may be provided upon request. Send a message to `mmcgill@cse.unsw.edu.au` if you would like to try them.

---

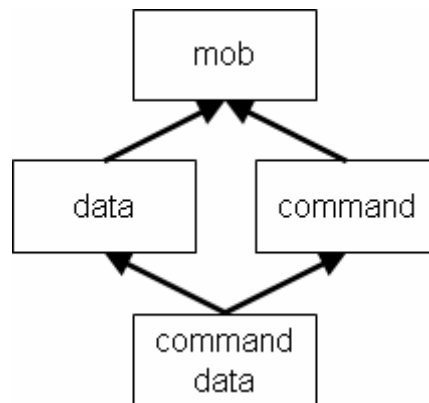
<sup>1</sup>This is a binary copy of the HSQLDB database available from <http://hsqldb.org/>.

---

# Chapter 3. Mobs

Mica objects (mobs) are the messages that agents post to the blackboard. Mobs use a frame paradigm [1] to formalize their information. In this paradigm each mob has a set of slots that hold values. As yet Mica does not restrict the slots that can be added to a mob or the values that can be placed in a slot.

## Mob Types



**Figure 3.1. Example Mob Type Hierarchy**

Mobs when created are given a type. A mob's type is used to determine which agents to notify when it is written to the blackboard. Mica allows the multiple inheritance of types but doesn't accept cyclic inheritances. The type 'mob' is the top most mob type and all mobs will inherit from it either directly or indirectly. When a mob is created it is set to its given type.

Figure 3.1 shows what a simple mob type hierarchy may look like.

The mob types are managed by the blackboard. For information on how to define mob types see the section called "Declaring Mob Types".

## Mob Persistence

When a mob is written to the blackboard it is possible to tell the blackboard how long the mob is to be kept. This is done by setting the mob's persistence. There are three values that a mob's persistence can take: *permanent*, *transient* and *connected*. The meaning of these persistences are:

- |           |   |
|-----------|---|
| permanent | The mob is to be kept until it is explicitly deleted.   |
| transient | The mob is not to be kept. Just notify registered agents and forget about it.   |
| connected | The mob is to be kept as long as the agent that wrote the mob remains connected to the blackboard. This persistence can be used by mob declaring services that they offer when they are being used in a service oriented environment. |



---

# Chapter 4. Agents

In any system built utilising Mica the agents of the system are the most. Virtually all functionality performed by the system and all interaction with users will be performed or mediated by one agent or another.

## Agent Functions

For an agent to successfully work with Mica it needs to implement a number of specific functions. These functions are described in the following sections. So as not to restrict the agents capabilities how these agents implement the functions is left up to the agents developers.

The functions required for agents can be divided into two groups. The first four functions are used to receive mobs and information from the blackboard. The rest are used to standardise the manner in which agents are created and destroyed.

Some basic default implementations of these functions have been implemented in DefaultAgent, see the section called “DefaultAgents” for information about the implementation.

### **handleNewMob(Mob)**

If any new mob is written to the blackboard and this agent is registered to receive it then this method will be called. For most agents this is the most important function as it determines how the agent is to respond to any received mobs.

### **handleDeletedMob(Mob)**

When a mob is deleted from the blackboard and this agent is registered for its type then this method is called. This allows an agent to stay up to date with information on the blackboard.

### **handleReplacedMob(Mob, Mob)**

When a mob is replaced and this agent is registered for its type then this method will be called.

### **handleTypeManagerChanged()**

This method is called when the blackboard detects that its mob hierarchy has changed. Typically this only occurs when agents post mobs of unknown types which are then set to inherit from 'mob'.

### **setTransport(AgentTransport)**

This method tells the agent what transport it is to use to connect to the blackboard.

### **getTransport()**

This method is used to get the transport the agent is using to connect to the blackboard.

### **init(MicaProperties)**

The standard process for creating and starting an agent is:

1. Create the agent.
2. Set the agent's transport.

3. Initialise the agent.

This method is used to implement the third step in this process. Typically in this method an agent will connect to the blackboard and register for any mob type it is interested in.

Passed as an argument to the method is a set of parameters that may be used to configure the agent.

## **terminate()**

This method tells the agent to shut itself down. Commonly this method will disconnect the agent from the blackboard.

# **Agent Transport**

An agent's transport is its connection to the blackboard. As such the transport provides all of the actions that an agent can initiate with the blackboard. The function provided by the agent transport are described in the following sections.

## **connect(String)**

This method attempts to connect to the blackboard using the given string as the agent's name. If successful the method will return the actual name used to identify the agent.

## **disconnect()**

Disconnects the agent from the blackboard.

## **register(String)**

Tells the blackboard that the agent is interested in all mobs of the given type.

## **unregister(String)**

Tells the blackboard the agent is no longer interested in the given type.

## **getTypeManager()**

Requests a type manager that knows the current type hierarchy.

## **writeMob(Mob)**

Writes the given mob to the blackboard.

## **readMob(String)**

Reads the named mob off of the blackboard.

## **deleteMob(String)**

Deletes the named mob from the blackboard.

## **replaceMob(Mob)**

Overwrites the mob with the new values.

## mobSearch(String)

This method is used to search the blackboard for all stored mobs that fit a given condition. The parser for the search string is dependant on the actual blackboard being used.

## Search Query Language

The default implementation in the Mica distribution includes a blackboard with an SQL back end. Searching this blackboard is done using standard SQL statements with some functional additions applicable to mobs. These functions can be used to restrict the output from the search or to order the returned mobs.

### SQL Mob Functions

<code>typeof(mob, 'type')</code>	allows you to check the type of a mob. This method returns a boolean.
<code>hasslot(mob, 'slotName')</code>	allows you to check whether a mob has a particular slot or not.
<code>getslot1(mob, 'slot-Name')</code>	gets the first value of a slot. It returns a string.
<code>getslot1asdbl(mob, 'slotName')</code>	gets the first value of a slot. It returns a double.
<code>getslot1asint(mob, 'slotName')</code>	gets the first value of a slot. It returns an int.
<code>getslotn(mob, 'slot-Name', pos)</code>	is a useful method for getting arbitrary information from a multi-valued slot.
<code>contains(mob, 'slot-Name', 'value')</code>	allows you to check whether a mob has a particular value stored somewhere in a multi-valued slot.

### SQL Examples

Here are some example queries to help you on your way.

#### Getting all mobs on the blackboard

To get all of the mobs currently on the blackboard you can use:

```
select * from mobs
```

#### Getting all mobs of a particular type

To find all mobs of a particular type you could try:

```
select * from mobs where typeof(mob, 'text')
```

#### Ordering mobs

To order the mobs you can try the `order by` command. This will look like:

```
select * from mobs where typeof(mob, 'text')
order by getslot1(mob, 'creationTime')
```

Appending `asc` or `desc` will place the mobs in ascending and descending order respectively.

#### Getting the first of a list of mobs

To get the most recent mob you could try:

```
select top 1 * from mobs
order by getslot1(mob, 'creationTime') desc
```

### Finding mobs with particular properties

To find a mob you have written you could use:

```
select * from mobs
where contains(mob, 'creator', 'myAgentName')
```

As long as only one value is in the creator slot this could also be done by:

```
select * from mobs
where getslot1(mob, 'creator') = 'myAgentName'
```

## isConnected()

Checks that the transport is connected to the blackboard.

## getAgentName()

Asks the transport for the name the agent is currently connected to the blackboard with.

## setMessageHandler(MessageHandler)

This method is used to tell the transport who is to respond to the messages coming from the blackboard. Typically an agent transport's message handler is the agent and is set when the transport is created. So this method is often not required.

## synchronizedWriteMob(Mob, long)

An extension of the agent transport is a synchronized transport <sup>1</sup>. This transport allow mobs to be used to perform remote procedure calls(RPC) over Mica.

This method writes the given mob to the blackboard and waits for a mob to be sent in reply. A reply mob is any mob that the agent is registered for that names the original mob in its 'replyTo' slot.

The method returns an object containing the name of the written mob and the received reply. If no reply mob is received in the given time period then the returned object will have no reply.

---

<sup>1</sup>The implementation is `unsw.cse.mica.sync.SynchronizedTransport` which is used by wrapping it around a standard agent transport.

---

# Chapter 5. Blackboards

The blackboard is the central hub in the Mica architecture. All agents connect to the blackboard and all mobs are written to it. The blackboard is responsible for deciding which agents need to be notified when mobs are written and modified.

The standard Mica distribution includes 2 simple blackboards. The `SimpleBlackboard` stores mobs in a `HashMap`. The `SQLBlackboard` stores the mobs in a table in an SQL database and provides an SQL style language for searching for mobs. To use the `SQLBlackboard` a compatible SQL database needs to be available. Included in the Mica distribution is `hsqldb.jar` a binary distribution of the HSQLDB database. To use this with the `SQLBlackboard` add `hsqldb.jar` to the classpath.

## Declaring Mob Types

```
<typedesc>
  <mobdecl name="text" >
    <slot name="utterance" />
  </mobdecl>

  <mobdecl name="textFromUser" persistence="transient" >
    <parent name="text" />
  </mobdecl>

  <mobdecl name="textForUser" persistence="transient" >
    <parent name="text" />
  </mobdecl>

  <include file="myTypes.xml" />
</typedesc>
```

**Figure 5.1. Example Type Definitions**

As the blackboard is responsible for maintaining the mob type hierarchy the blackboard needs a means by which it knows what the type hierarchy is. With the standard Mica blackboard a developer can define the type hierarchy using a simple XML file. An example of such a file is shown in Figure 5.1.

When constructing these configuration files each mob type is declared in its own `mobdecl` element. This element gives the name of the type and its default persistence. Inside the type declaration you list the parent types and optionally the slots that the type expects. As yet the slot information is not used except in documenting the types for developmental purposes.

Inside the type definition files it is possible to use an `include` element to load types from a given file or directory.

When loading the types the blackboards' default action is to look in the `config/type` directory of the `Mica.Home` directory which is by default the current working directory. All `.xml` files in this directory are considered types definition files and are loaded.

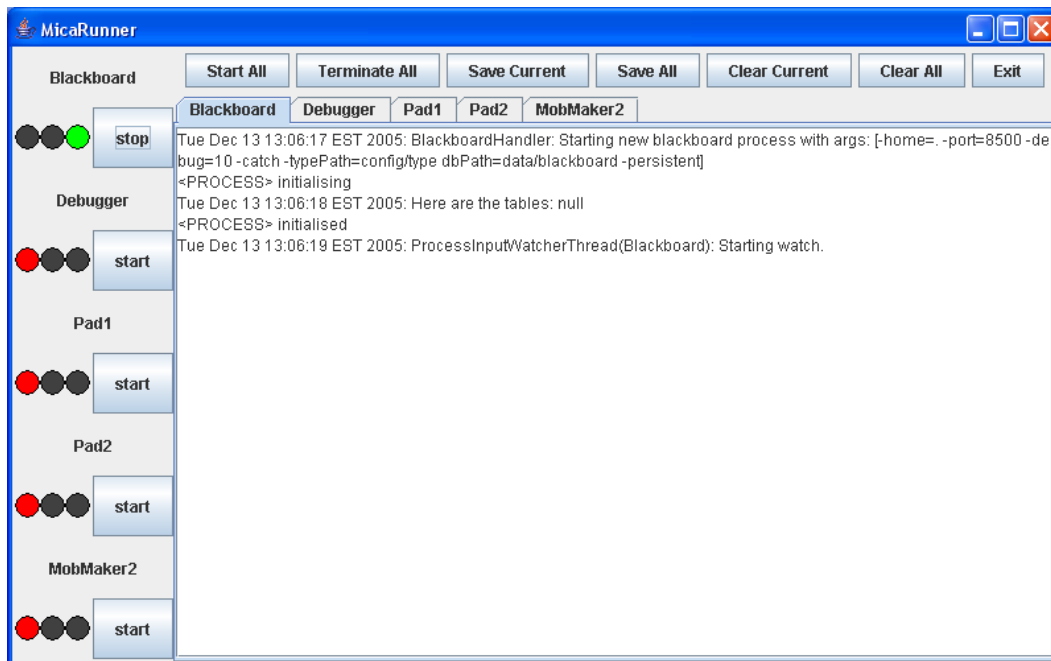
For more details on the XML format of type definition files look in `types.dtd`.

---

# Chapter 6. MicaRunner

MicaRunner is a tool to simplify and automate the task of starting up a blackboard and series of agents. Its main role is for starting and stopping a mica network during development and testing.

## Interface



**Figure 6.1. MicaRunner User Interface**

By default MicaRunner opens a graphical interface though it can be configured to run without it.

This interface is divided into roughly 3 sections. On the left of the interface you will find the name of each agent/blackboard and below it a coloured indicator and a button. The button is used to start and stop the agent/blackboard and the indicator shows the agent's status. When the agent is running the indicator will be green and when it is stopped the indicator will show red. When the agent is started the indicator will turn amber until the agent is completely initialised. If the indicator is amber and flashing it means the agent is waiting for an agent it depends on to finish initialising before it can start.

At the top of the interface is a line of buttons that can be used to start and stop all of the agents in MicaRunner.

Taking up most of the interface is a series of text panes in a set of tabs. Each agent has its own tab and in the text pane will be found the output of the process in which the agent is running.

### Note

As yet these text panes cannot be used to send input to the processes/agents.

# Configuration

```
<runner>
  <blackboard />

  <agent class="unsw.cse.mica.tools.LogDebugger" name="AnAgent" />

  <host name="www.myhost.com" />
  <port number="8500" />

  <agent class="unsw.cse.mica.tools.LogDebugger" name="AnotherAgent" >
    <depends name="AnAgent" />
    <arg param="x" value="600" />
  </agent>
</runner>
```

**Figure 6.2. Example MicaRunner Configuration File**

MicaRunner needs to be given a configuration file for it to run. A very simple configuration file is shown in Figure 6.2. This configuration file tells MicaRunner what blackboards and agents are needed.

When defining an agent it is possible to add a `depends` element to indicate that the agent requires another agent to be running before it can be successfully be started. By default if the runner has to start a blackboard all agents will be dependent upon it running.

When declaring an agent the class which implements the agent needs to be specified. Optionally the name the agent is to use can also be given.

It is possible to define parameters using `arg` tags that will be passed to an agent/blackboard in its `init` method.

By default agents will attempt to connect to a blackboard running on the local machine and both blackboards and agents will attempt to connect using port 8500. It is possible to use `host` and `port` tags to override this default behaviour.

If all agents are connecting to a remote blackboard MicaRunner is not required to start a local one.

MicaRunner processes tags in the order found in the configuration file so settings will only apply to agents/blackboards declared after the setting and not those declared before it.

There are many more configuration options available to MicaRunner. For details look at `runner.dtd`.

---

# Chapter 7. Mica Security

The common transport used with Mica and used by default with MicaRunner is an XML protocol sent over TCP. This provides no security at all for either agents or mobs. Alternatively it is possible to use a secure XML protocol and an SSL connection layer.

When using MicaRunner the transport protocol and connection layers can be selected using a transport tag. Available protocols are 'xml' and 'secure', while connections can be either 'tcp' or 'ssl'. Such a tag will look like:

```
<transport protocol="secure" connection="ssl" />
```

## SSL Connections

The SSL connection for Mica allows all communication between an agent and the blackboard to be encrypted.

The SSL connection layer for Mica uses standard Java SSL sockets. To use these sockets Java needs to know where to look for valid SSL certificates. The easiest way to do this is to give Java access to a key store and the password for the store. For a blackboard this can be done by setting the Java parameters `javax.net.ssl.keyStore` and `javax.net.ssl.keyStorePassword`. For an agent the parameters `javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword` need to be set.

Key stores can be created using Java's `keytool` tool.

If using MicaRunner the key store can be set using a `keystore` tag attached to the `transport` tag. This will look something like:

```
<transport protocol="secure" connection="ssl" >
  <keystore location="myKeyStore" password="123456" />
</transport>
```

## Secure XML Protocol

The secure XML protocol is an extension of the XML that can be used to restrict the mobs that an agent can see or write. It can also be used to prevent unauthorised agents from connecting to the blackboard.

On the agent side of the secure XML protocol there is little change to the XML protocol. The only change is that the secure protocol allows the optional setting of a password that will be sent along with the agent's name when connecting to the blackboard.

Virtually all security in the secure XML protocol is handled in the blackboard's transport layer. This is done by adding a security manager to the blackboard's transport protocol. This security manager is responsible for deciding whether an agent will be allowed to connect and what it can and cannot do with the mobs on the blackboard.

A very simple static security manager has been implemented in `unsw.cse.mica.blackboard.secure.SimpleBlackboardSecurityManager`. When using MicaRunner this is the security manager used. This security manager will look in the `config/security` directory of `Mica.Home` for its configuration files.



## Verifying Agents

```
<agents>
  <group name="Writer" />
  <group name="Reader" />
  <group name="Guest" groups="Reader" />
  <agent name="AnAgent" password="123456" groups="Writer Reader"/>
  <agent name="AnotherAgent" groups="Guest"/>
</agents>
```

**Figure 7.1. Example Agent Verification File**

The `SimpleBlackboardSecurityManager` will look in the file `agents.xml` for information on allowed agents and the groups to which they belong. Groups can be defined in a hierarchy if desired. Figure 7.1 shows an example configuration file with 3 groups and 2 agents.

When an agent is declared with a password all agents attempting to connect using that name must provide the required password. If no password is given then any agent trying to use that name with a password will be refused connection.

Agents can belong too multiple groups and groups can inherit from multiple groups. To do so add a `groups` attribute to the tag and a space(' ') separated list of groups.

The format for the agent verification file is given in `agentauthorisationrules.dtd`.

When using `MicaRunner` the password for an agent to use can be set by adding a `transport` tag to the agent and setting the `password` attribute. This will look like this:

```
<agent class="MyAgent" >
  <transport password="123456" />
</agent>
```

## Restricting Agent Actions

`SimpleBlackboardSecurityManager` will look in the `rules.xml` configuration file for the rules it uses to determine what an agent can and cannot do. An example rules file is shown in Figure 7.2.

In the `rules` it is possible to set whether the default is to accept or reject an action. If no default is given the default is to accept.

The rules are a list of `if` elements that say whether to accept or reject the action. Each `if` element has a condition followed by a `then` tag and an action. It may also have an `else` tag and alternate action for when the condition is false. The action can be to accept or reject or another conditional `if` statement.

```
<rules default="reject">
  <if>
    <and>
      <action type="write"/>
      <group name="Writer" />
    </and>
    <then />
    <accept/>
  </if>
  <if>
    <and>
      <action type="read" />
      <group name="Reader" />
    </and>
    <then />
    <if>
      <and>
        <group name="Guest" />
        <or>
          <type name="classified" />
          <slot name="classified" />
        </or>
      </and>
      <then />
      <reject />
    <else />
      <accept />
    </if>
  </if>
</rules>
```

**Figure 7.2. Example Agent Rights File**

There are a number of conditions that can be used to determine whether or not to accept an action. Available conditions include:

action	test if the agent is trying to read or write the mob
group	test if the agent belongs to the given group
type	test if the mob is of a given type
slot	test if the mob has a slot, can also be used to check if a slot has a specific value
and	groups a set of conditions that must all hold true
or	groups a set of conditions of which at least one must be true
not	tests that a condition is false

Each rule will be processed in order until a designation to accept or reject is found. If all rules are tried and no designation is made the default action to accept or reject will be used.

The XML format for the rule configuration file is given in `agentauthorisationrules.dtd`.

If an agent is allowed to read a mob it will receive all `handleNewMob`, `handleDeletedMob`, `handleReplacedMob` events associated with it. It will also be able to see the mob using the `readMob` and `mobSearch` functions. The ability to write a mob includes writing the mob, deleting the mob and replacing it.

---

# Chapter 8. Tools

To make developing an application with Mica easier a number of tools have been put together and included in the standard Mica distribution. These tools serve a variety of purposes from making agents easier to implement to enabling state of the blackboard to be examined and manipulated.

## DefaultAgents

To make constructing agents a little easier two abstract implementations of the `Agent` interface are available.

The first, `DefaultAgent` is practically empty adding empty method bodies to many of the required methods. Any agent extending this class will have to implement its own `init` and `handleNewMob` methods.

The second abstract `Agent` implementation is `DefaultAgent2`. This class provides generic implementations of the interface functions.

`DefaultAgent2`'s major differences to `DefaultAgent` are that it includes a constructor that can be used to ensure the agent has an `AgentTransport` that can be used for synchronous communications over the blackboard (see the section called “`synchronizedWriteMob(Mob, long)`”) and it maintains a `TypeManager` that is kept up to date with the blackboard.

The `init` method of `DefaultAgent2` will connect the agent to the blackboard using a name supplied as an argument or else using the agent's class name.

## SimpleAgentFrame

`SimpleAgentFrame` is an extension of `JFrame` that can be used to provide an agent with a simple GUI.

The main benefits of the `SimpleAgentFrame` is that it can configure its size, position and title according to parameters given to the agent and that when the window is closed it will call `terminate` on the agent.

## GUIAgent

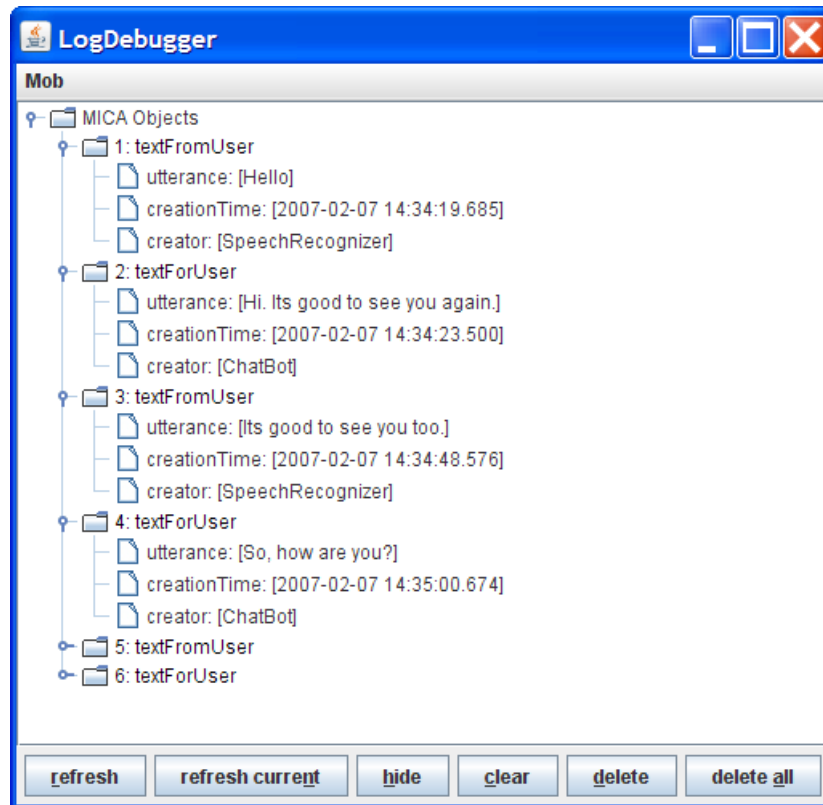
`GUIAgent` is an extension of `DefaultAgent2` that adds a `SimpleAgentFrame` to the agent.

Any agent that extends `GUIAgent` will have to implement `createComponents(MicaProperties)` in which it will create its GUI components and add to the `SimpleAgentFrame` frame.

`GUIAgent`'s `terminate` method will close and discard the frame when called.

Some examples of the agents that can be built using `GUIAgent` are `LogDebugger` and `MobMaker`.

# LogDebugger



**Figure 8.1. LogDebugger Interface**

The LogDebugger is a fairly simple agent that keeps track of all of the mobs on the blackboard. It is registered for all mobs so when any mob is written, replaced or deleted it is notified and logs the change.

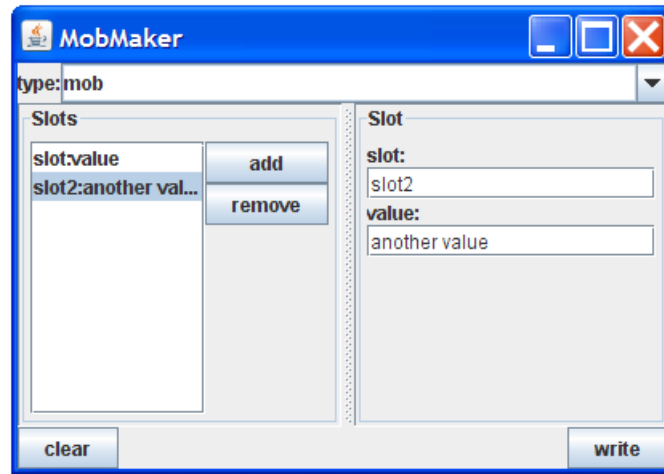
The interface to LogDebugger is shown in Figure 8.1.

Colour is used to show a mob's state. If the mob is black then it resides on the blackboard. If the mob is green then it is transient. If it is blue then it has been replaced. If it is red it has been deleted. Red mobs that have a line through them show when the mob was deleted.

The buttons along the bottom of the interface allow the user to refresh the display with all mob changes since the agent was initiated, refresh only the mobs currently on the blackboard, hide selected mobs, hide all mobs, delete selected mobs and delete all mobs.

The LogDebugger also has a simple search mechanism for displaying those mobs that match a small number of criteria.

# MobMaker



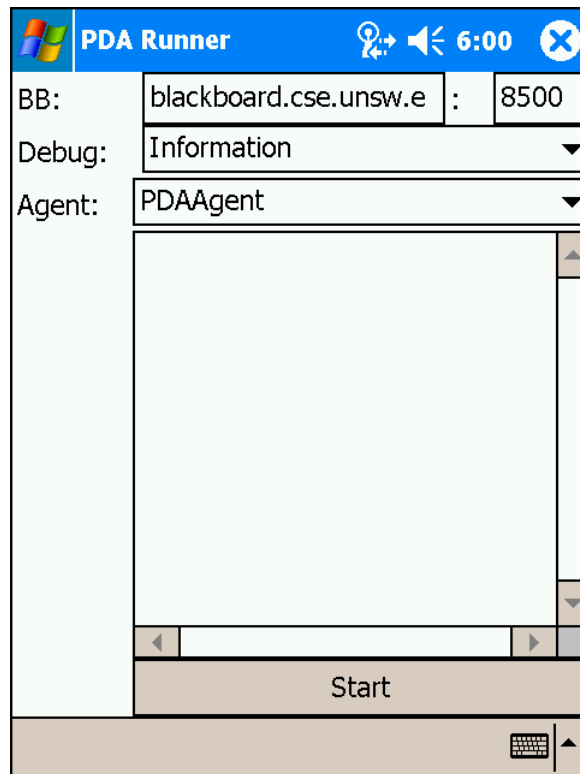
**Figure 8.2. MobMaker Interface**

The MobMaker agent is a simple agent used in testing how agents respond to mobs. A MobMaker can be used to create any type of mob it can find from a TypeManager it loads when it is initiated. The MobMaker interface is shown in Figure 8.2.

To create a mob using the MobMaker select the mob type from the pull down menu at the top of the interface then add each slot and its value. If a list of values is required for a slot. The add each value separately, the MobMaker will add the values in the order that they are found in the slot list. If needed slots can be removed from the mob.

When the mob is complete press the 'write' button to write it to the blackboard. Once th mob is written you can edit slots and type and write another mob or you can select 'clear' which will clear all the values for the mob.

# PDARunner



**Figure 8.3. PDARunner Interface**

The PDARunner is a small tool for starting Mica agents on a PDA. The interface for PDARunner can be seen in Figure 8.3.

The top line of the interface allows the user to select the blackboard the agent is to connect to. The second line allows the user to select the amount of debugging information they want to see. The third line is a pull down menu of currently available agents. Below this is a text area which will display all debugging output the agent writes using `unsw.cse.mica.util.Debug`. At the very bottom of the interface is the button to start the agent.

## Note

As yet the PDARunner only supports XML over TCP transports as it is not yet known whether SSL sockets are available on the PDA.

## PDARunner Configuration

```
<agent class="unsw.cse.mica.demo.PDAAgent" >
  <classpath >
    <pathelement url="pdaagent.jar" />
  </classpath>
  <arg param="context" value="example" />
</agent>
```

**Figure 8.4. Example PDARunner Configuration File**

An example configuration file for PDARunner is shown in Figure 8.4. The PDARunner looks in the `\My Documents\Jars\` folder of the PDA for its agent configurations. Any file with a name like `'agent-*.xml'` in the folder is considered to be an agent configuration file.

The root element of the file is an `agent` tag which gives the class for the agent and optionally a name for the agent.

When the PDARunner is started it uses a minimal classpath so the agent can be given a `classpath` tag to add required classes to the classpath. Each new jar file or package location is added using a separate `path` element tag.

If needed, the configuration file can list a series of parameters using `arg` tags that will be passed to the agent in its `init` method.

The precise format for the PDARunner configuration files can be found in `pdarunner.dtd`.

## ProxyAgent

*TODO: ...*

## Services

*TODO: ...*

---

# Bibliography

[1] Minsky, Marvin. *A Framework for Representing Knowledge*. Massachusetts Institute of Technology. . 1974.