

Mica: Technical Guide

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

Mica: Technical Guide

by James Henry Westendorp, Mohammed Waleed Kadous, Matthew John McGill, and Claude Anthony Sammut
Copyright © 2006, 2007 School of Computer Science and Engineering, UNSW & James Henry Westendorp & Mohammed Waleed Kadous & Matthew McGill & 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
2. Agents	2
Transport Stack	2
Synchronized Communications	3
3. Blackboard	5
Transport Stack	5
Database Storage	5
4. MICA Runner	6
5. XML Transport Protocol	7
Transport Stack	7

List of Figures

2.1. Control flow for agent actions and messages	2
2.2. Control flow for synchronized agent actions and messages	3
4.1. MICA Runner Agent Processes	6

Chapter 1. Introduction

This guide provides a technical description for some of the more complex elements of the MICA framework. It is intended for use by developers and maintainers. It is *NOT* a description of the system or a User's Guide.

Chapter 2. Agents

TODO: ...

Transport Stack

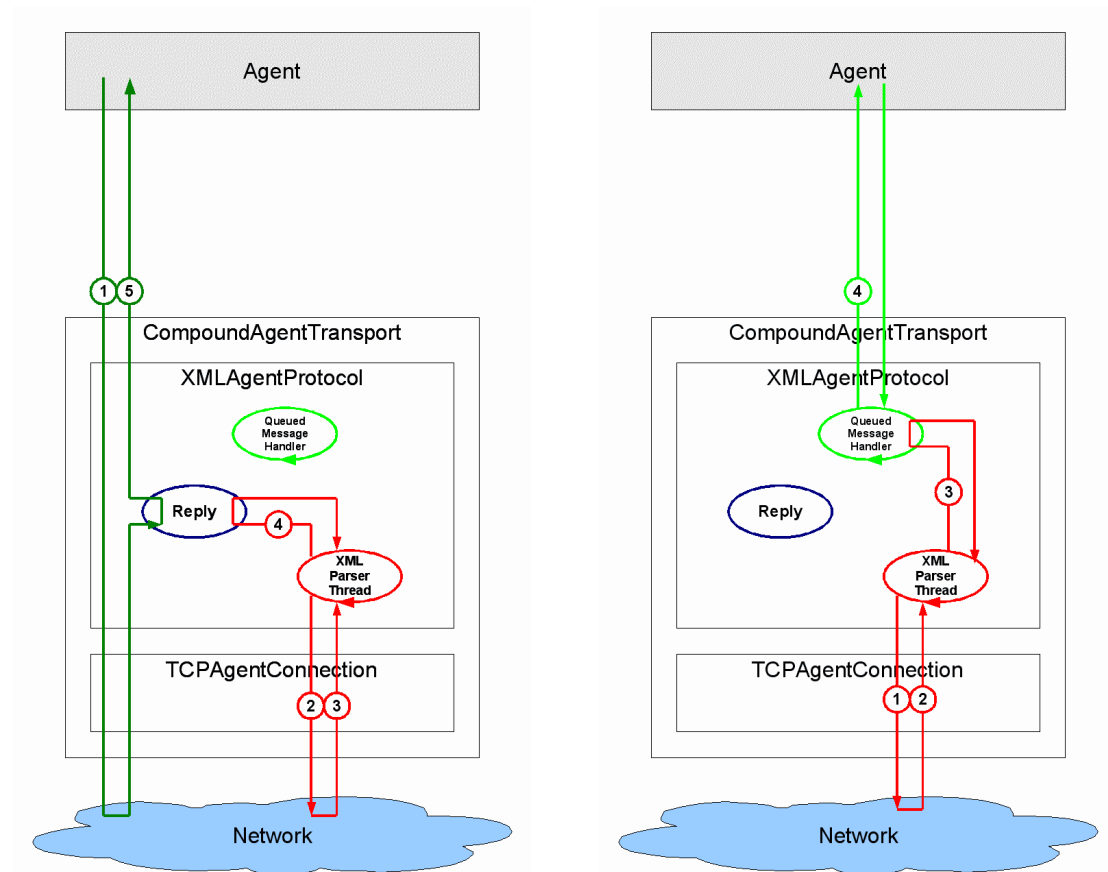


Figure 2.1. Control flow for agent actions and messages

Agent-Blackboard communication is based on two high-level interfaces: `AgentActions` and `AgentMessages`. Individually, these interfaces are relatively simple. However, providing an asynchronous two-way communications channel that implements these interfaces is more complex. Figure 2.1 shows the steps involved in this communication process when (a) executing an action and (b) handling a message from the blackboard, when using the `XMLOverTCPBlackboard`.

Action execution requires the following steps:

1. The process begins when the agents calls an action method on the transport. The calling thread moves down through the protocol, generates a message, and sends that message. Once control returns to the protocol class, the thread blocks and waits for a reply.
2. The protocol class has an internal thread responsible for parsing the incoming XML as it arrives from the network via the TCP connection. This thread is already running but remains block until a message arrives from the network.
3. Once a complete message has arrived, the parser thread processes it.
4. The message is an action reply, so it is stored in a location where the initiating thread can access it. The initiating thread is then woken.

- the initiating thread is resumed. It gets the reply and passes it back to the agent as the return value for the action.

The right-hand diagram shows the sequence of steps for the arrival of a message from the blackboard. These are:

- The parser thread has block and is waiting for new information from the blackboard.
- Once a completed statement arrives, it is parsed and sent to be a message.
- The message is passed to the protocol's other internal thread - a QueuedMessageHandler.
- The message handler thread repeatedly calls the agent's message functions as new messages arrive. If the thread is busy when new messages arrive, they are simply added to the queue and dealt with in turn. This ensures that the agent is only ever handling a single message from the blackboard at any one time.

Synchronized Communications

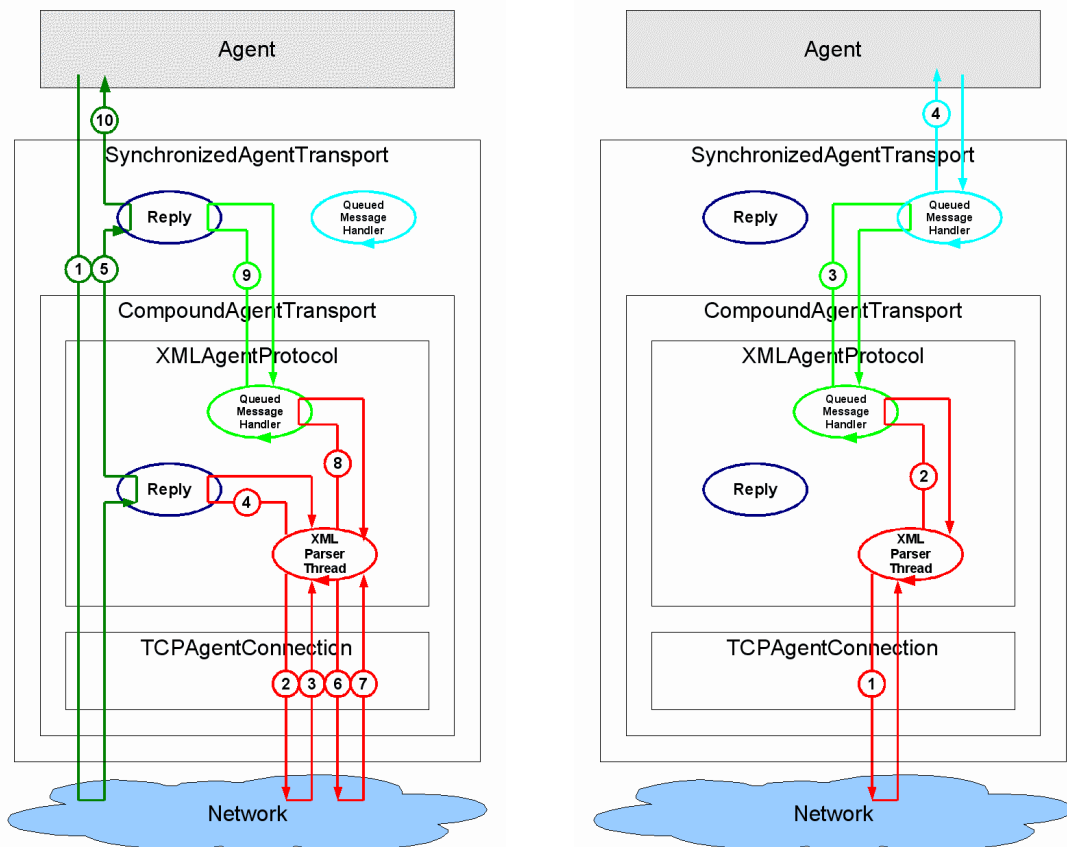


Figure 2.2. Control flow for synchronized agent actions and messages

MICA also provides a capability for synchronous communication between agents. This functionality is implemented as an optional additional layer on top of the asynchronous communications classes. Using this functionality, an agent can send a mob and wait for a reply to that mob. The mobs themselves are no different to any other mob. Rather, the process uses a specific slot to mark the return mob as being a reply to the original mob. This slot is called "replyTo" and its value is the id of the mob (or mobs) to which it is a reply. The synchronized transport layer uses the value of this slot when looking for a reply.

As with the asynchronous transport, the high-level interface `SynchronizedAgentActions` is quite simple. There is just one additional action the agent can call. However, implementing this functionality is more complex as it involves both an action and a message at the asynchronous level.

Figure 2.2 shows the flow of control for (a) synchronized messaging (b) the arrival of a normal message when a synchronized wrapper is in use. The steps involved in a synchronous action are:

1. The initiating thread generates a message inside the protocol and sends it, before block and waiting for a reply.
2. The parse thread is blocked and waiting for input
3. Input arrives from the blackboard
4. The input is parsed and found to be an action reply. The reply is stored and the initiating thread notified.
5. The initiating thread gets the reply and returns to the synchronized layer. It then blocks(again) and waits for a reply mob to arrive.
6. The XML parser thread is again block and awaiting input
7. More input arrives from the blackboard.
8. This input is parsed and found to be a message. It is passed to the queued handler.
9. The queued handler passes the message to the synchronous handler. Here it is determined that it is a reply to the mob originally sent by the initiating thread. The mob is stored as a synchronous reply and the initiating thread notified.
10. The initiating thread grabs the reply mob and passes it back to the agent.

The first five steps are the same as for asynchronous actions, it is only after the action reply is obtained the the process varies.

The synchronous wrapper must also ensure that any normal (non-reply messages) are still handled correctly. Again, the steps involved in this are similar to those for normal message handling:

1. the parser the thread is blocked and awating input.
2. Input arrives from the blackboard.
3. The input is found to be a message and is passed to the queued message handler.
4. The message handler passes the message to the synchronous layer. Here it is found that the message is not a reply to any sent messages, so it must be handled as a normal message. It is passed to a second queued handler inside the synchronous layer.
5. The synchronous queue handler behaves identically to the one in asynchronous layer, processing one message at a time in the order they arrive.

Several points worth noting in this process are:

- Reply mobs are *NOT* handled like normal mobs - the agent's `handleNewMob` functions is never called for them.
- It is possible that other messages will arrive between the initial action and the arrival of the reply mob. These mobs are handled like any other - they are added to the queue inside the synchronous layer and handled in the order they arrive. Of course, if the initiating queue is `queuedMessageHandler` thread (ie. the synchronized call was made from with one of the agents's message handler functions) then none of these mobs will be handled until after that message handling is complete
- It is possible that no reply mob arrives. The synchronous calls have a timeout parameter and it is strongly recommended that be used to avoid having the thread block indefinitely while waiting for a reply that may not arrive.

Chapter 3. Blackboard

TODO: ...

Transport Stack

TODO: ...

Database Storage

TODO: ...

Chapter 4. MICA Runner

MICA Runner is a tool enabling the configuration and execution of multiple agents on a single machine. It can also be configured to provide a blackboard on the current machine. MICA Runner functionality is described in the Users Guide. This section describes the way MICA Runner creates the child processes and interacts with them.

Figure 4.1. MICA Runner Agent Processes

Figure 4.1 shows how Mica Runner uses child processes for each agent and the components used for communication between processes. The components used for each agent within the MicaRunner process are:

AgentHandler	Each agent is represented within MicaRunner by an AgentHandler. This class contains all the functionality necessary to create, observe and terminate an agent.
AgentSettings	The AgentSettings class is a passive class containing the information necessary to create and configure an agent. It is the primary data source for the AgentHandler class.
InputOutputBridge	The I/O bridge class is a thread that gathers the output from the child process and passes it to the agent handlers output window
ProcessHandlerPane	Each agent handler has a processHandlerPane. This is this small controller pane in the list on the left hand side of the runner.
DebugPane	The debug pane is a modified TextArea that acts as a sink for the I/O bridge

Several classes are also used within the child process. These are

AgentHandler	The child process is initiated using the main() method within the AgentHandler class
--------------	--

Chapter 5. XML Transport Protocol

TODO: ...

Transport Stack

TODO: ...