Metalevel Engineering in AmbientTalk

REME-D: a Reflective Epidemic Message-Oriented Debugger

Elisa Gonzalez Boix
egonzale@vub.ac.be

Vrije
Universiteit
Brussel

Well-known Meta Programs

Debugger

Compiler

Class Browser

Object inspector

Interpreter

Profilers

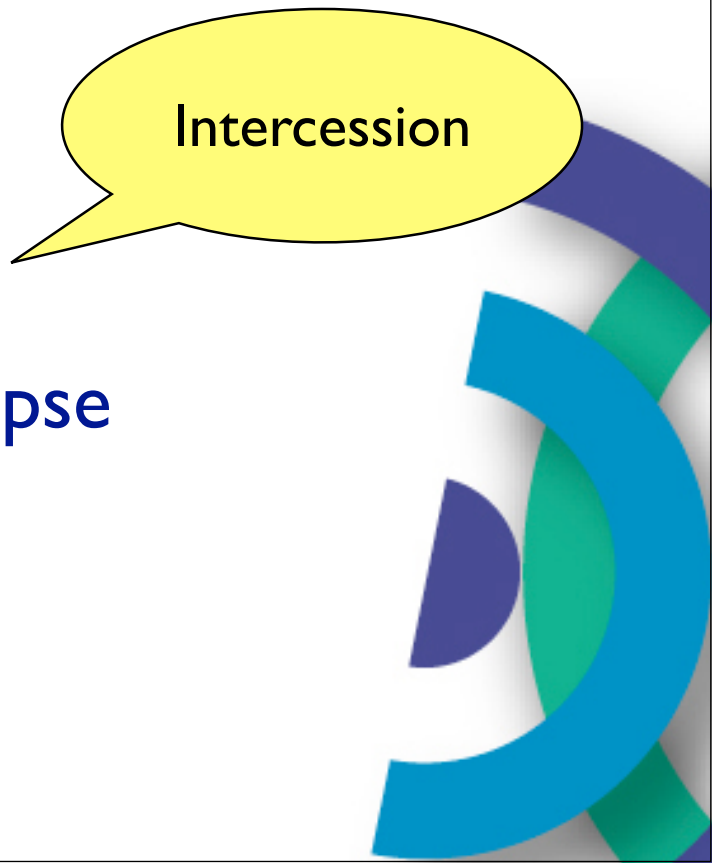
Monitors

Language Extensions

REME-D
from a user perspective

REME-D

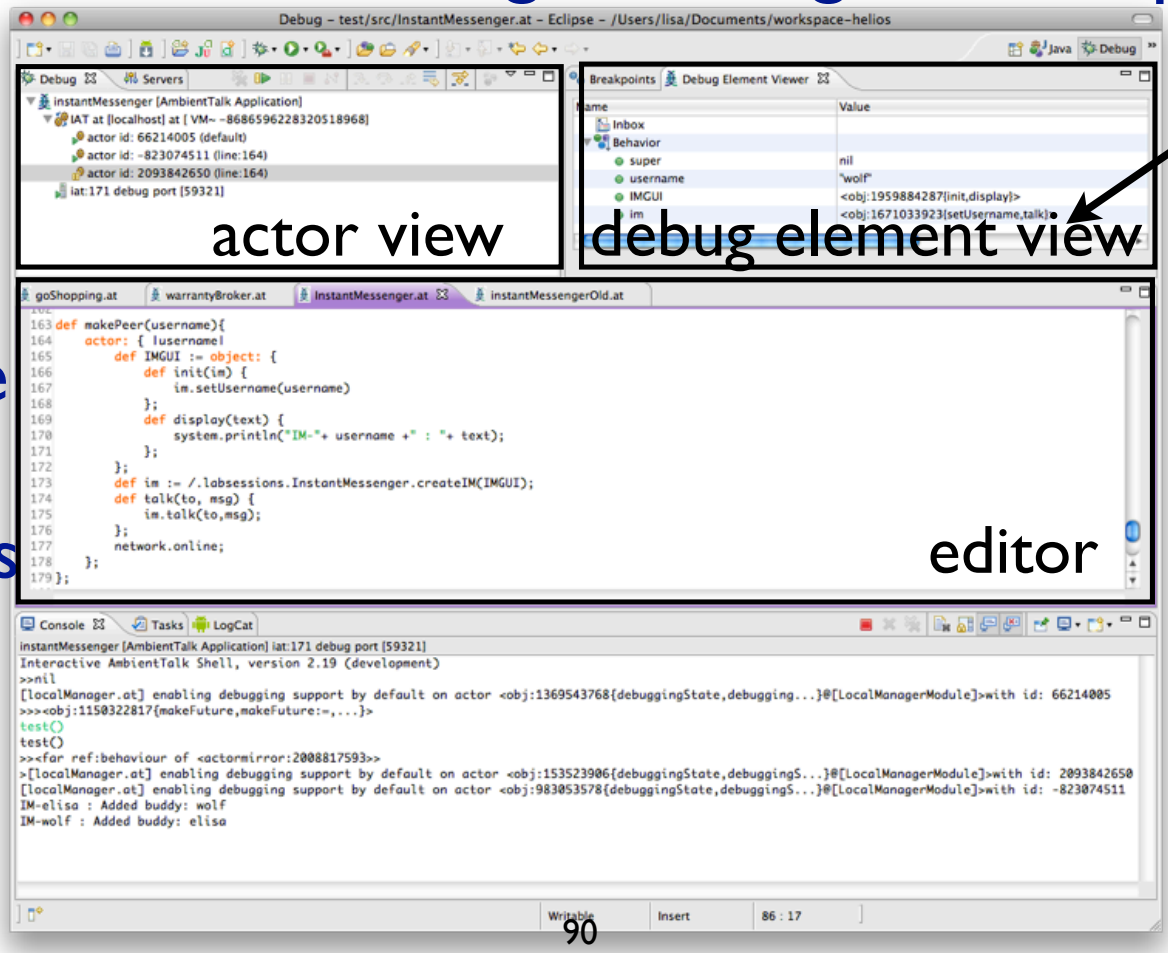
- Distributed debugger designed for AmOP programs.
- It is written in AmbientTalk itself reflectively.
- UI implemented on top of the Eclipse Debug Framework.



Message-Oriented Debugging

- Online distributed debugger:
  - asynchronous message sending and reception.

Message-Oriented

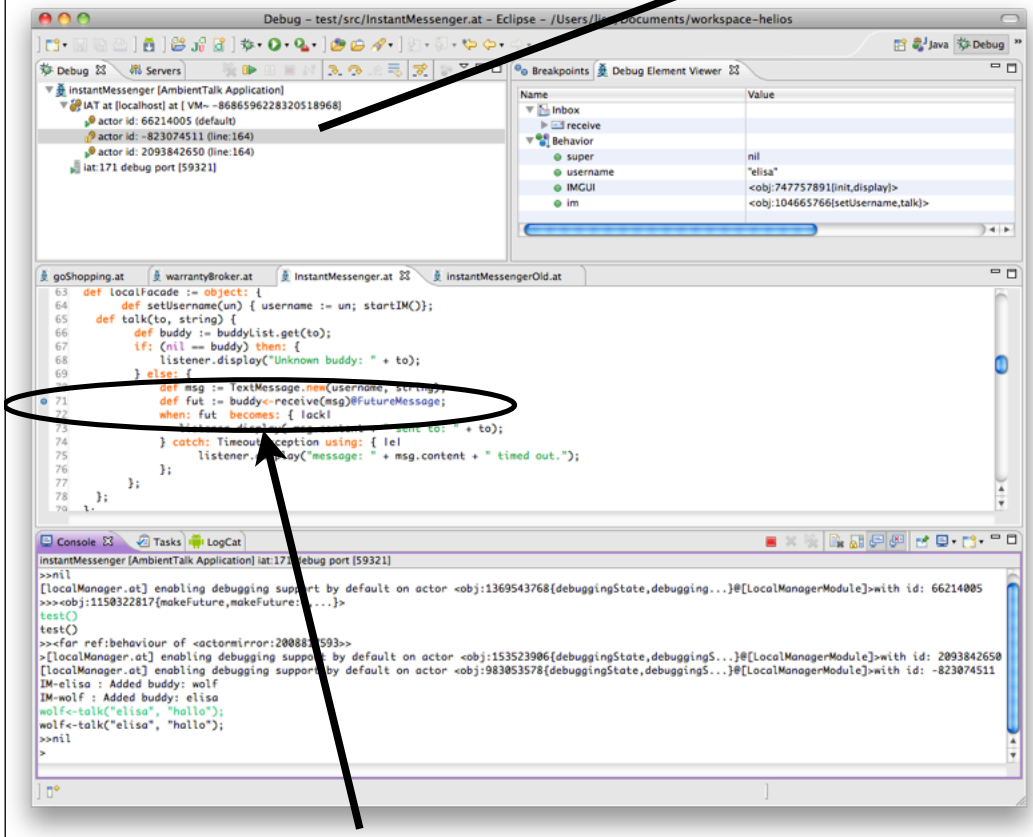


pause/resume actors between turns

inspect mailbox and behaviour

Message Breakpoints

Actor suspended when the message reaches the head of the mailbox

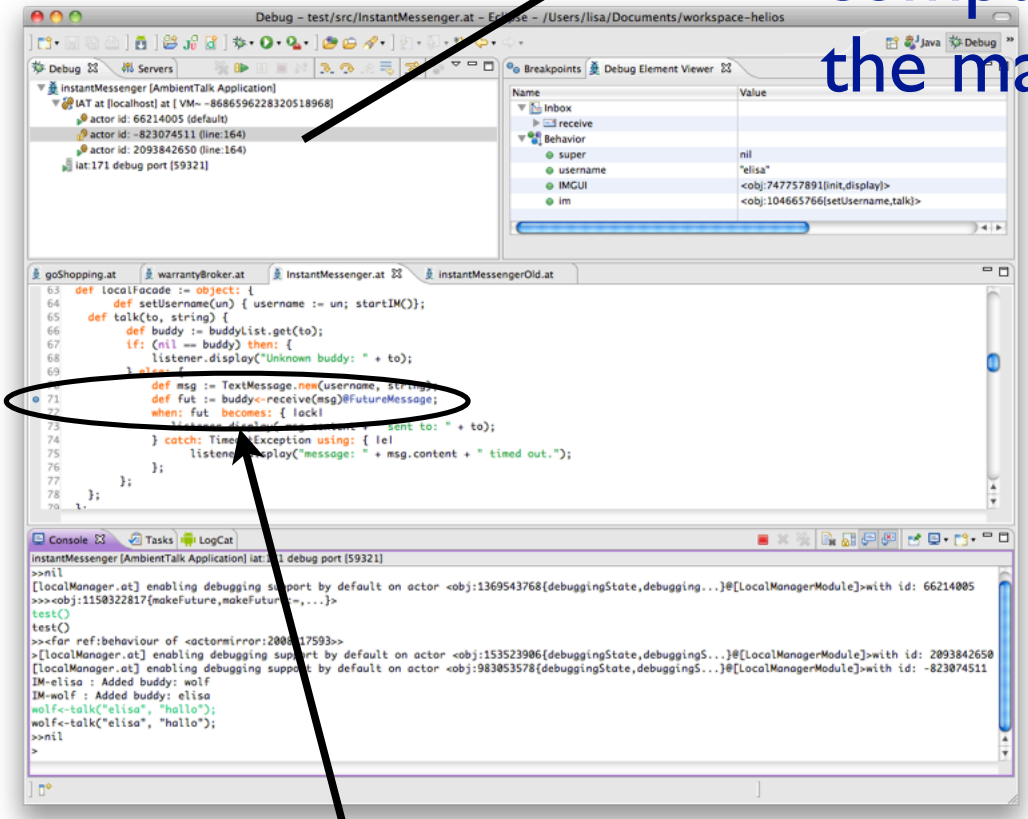


Breakpoints are configurable per VM.

Breakpoints on loc of asynchronous message sends (the <- operator)

Message Resolved Breakpoints

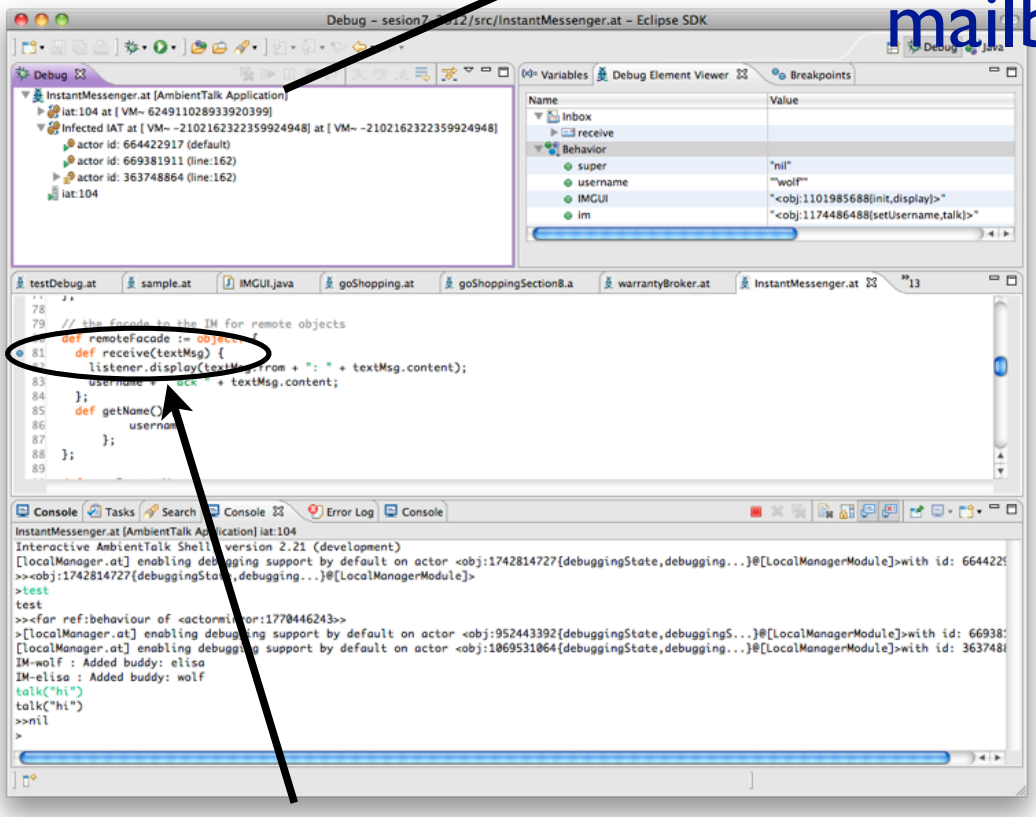
Actor suspended when the message with the return value of the computation reaches the head of the mailbox



Breakpoints on loc of asynchronous message sends (the <- operator)

Method Breakpoints

Actor suspended when any message defined on the given line of code reaches the head of the mailbox



Breakpoints on loc of a method definition

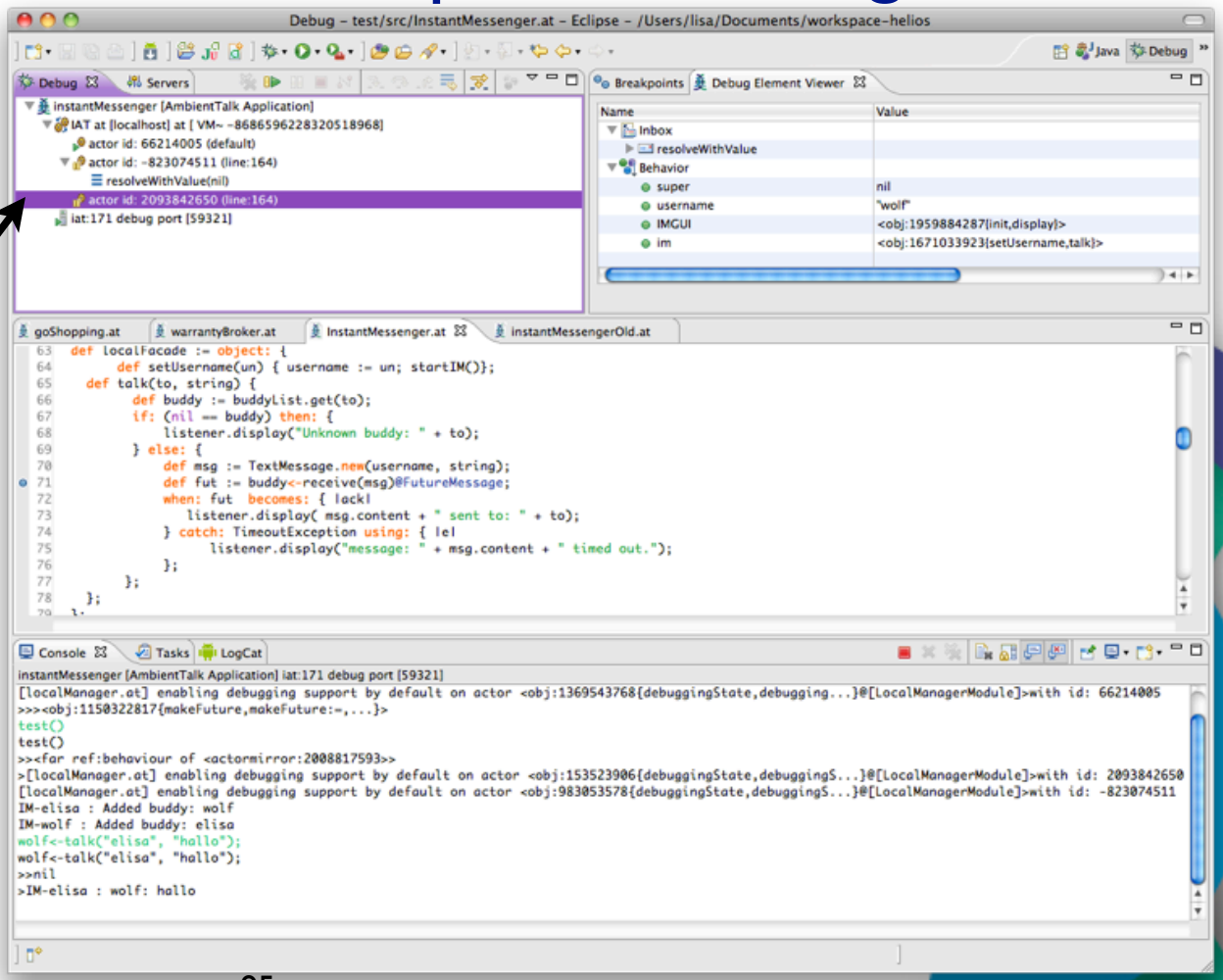
Step-by-Step Execution

- Step Over a message
- Step Into a message
- Step Return a message

Step Into

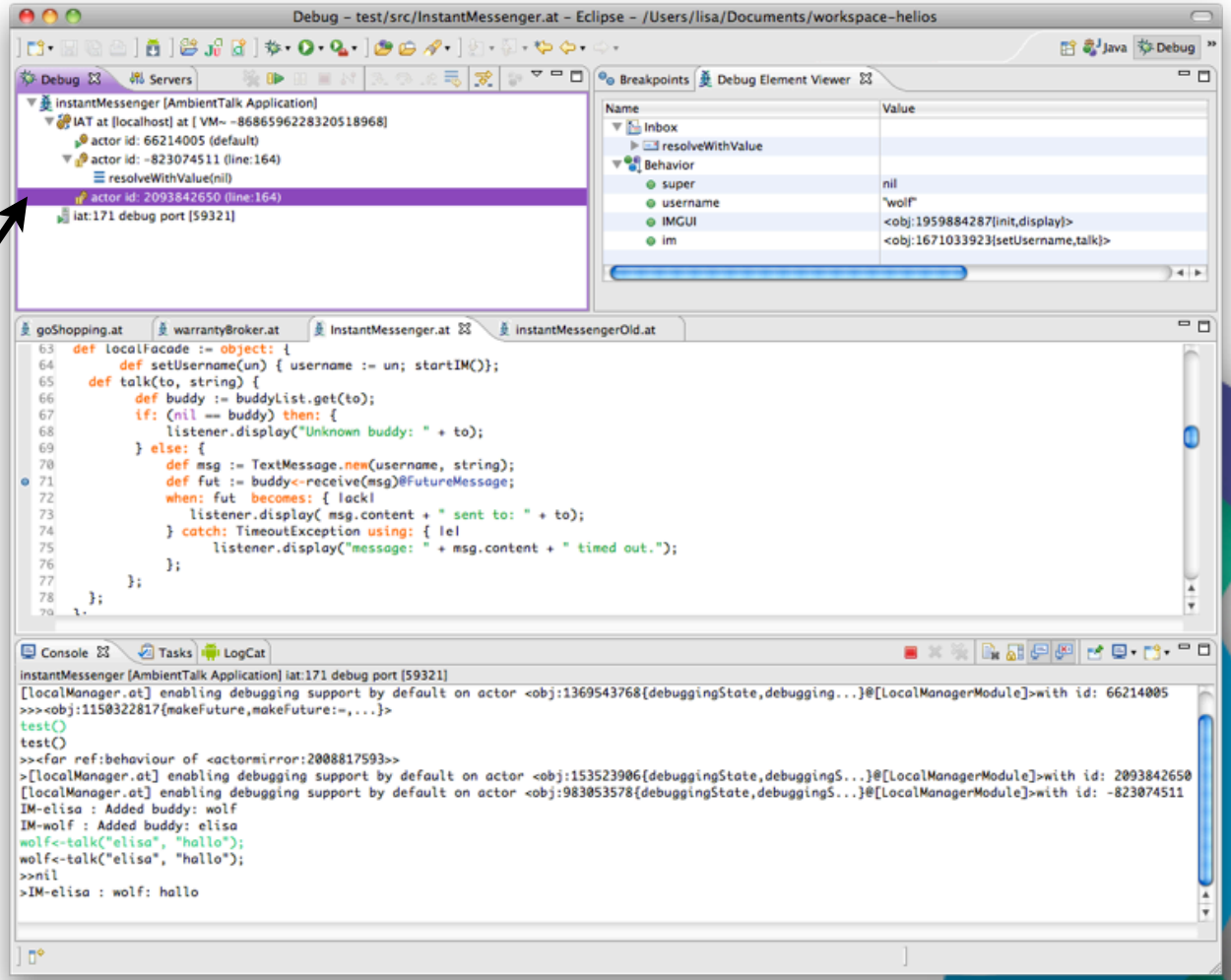
- Step-into to navigate on the messages sends as a result of the execution of the paused message.

current actor and receiver actors of messages sent within a turn are paused.



Step Return

- Return from a message is paused.



the receiver of the return value of a futurized messages sent is paused.

Open Debugging Sessions

Epidemic

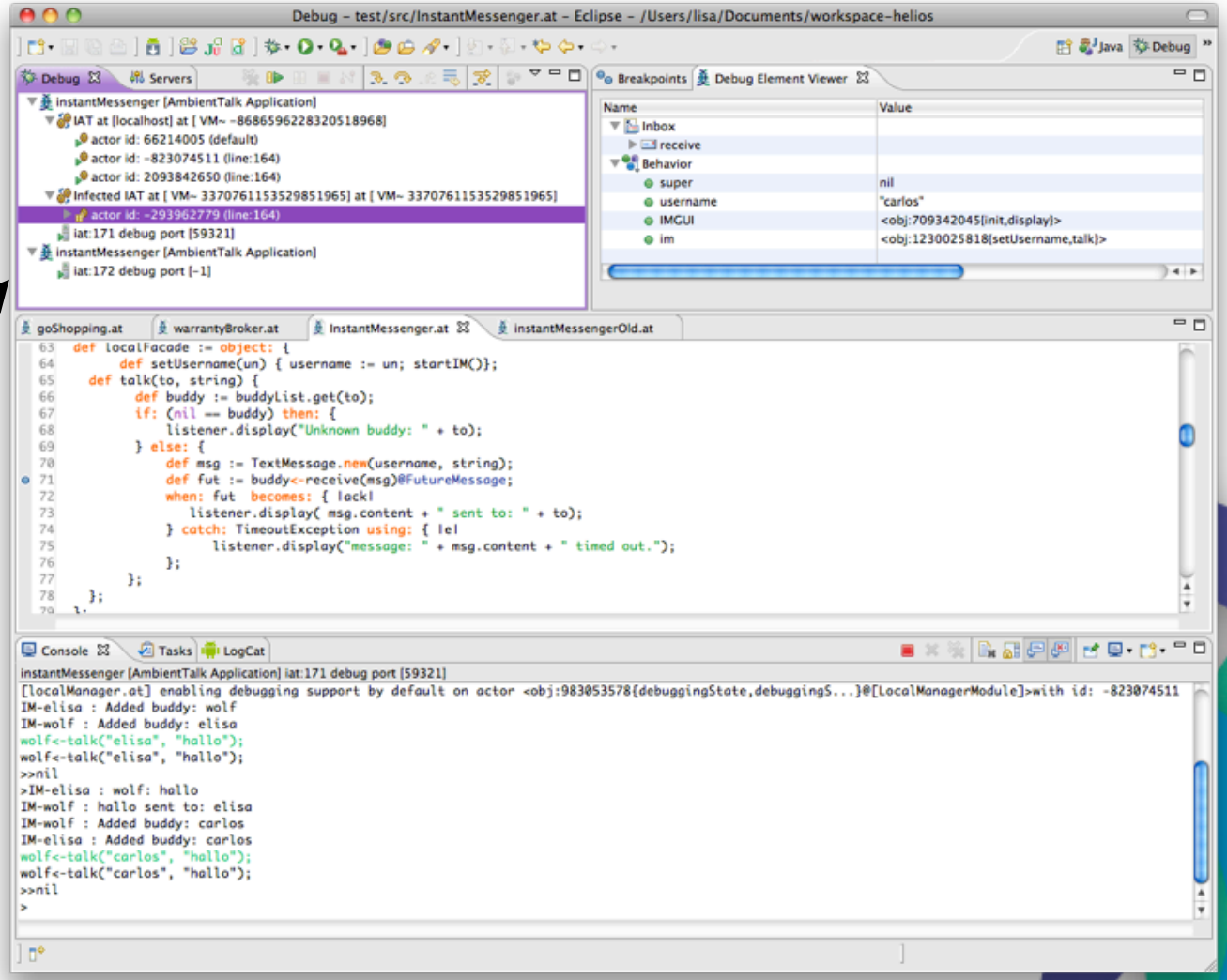
- It does not require to define a priori the devices participating in a debugging session:

Actors added dynamically to a debugging session upon receiving a message with breakpoint (*).

Network failures ≠ exceptions
actor removed from the debugging session, and resume if necessary.

(*) Actor needs to be declared debuggeable: lauched with -Xdebug option.

Open Debugging Sessions



infected actors are added to the debug view

infected actors are not stopped when the session ends!

REME-D

- More info at:

  http://code.google.com/p/ambienttalk/wiki/DebuggingInEclipse

- For the session:
  - Don't forget to open the Debug Element View.
  - Always debug your app with -n option.
  - Be patient with the Eclipse GUI :)

REME-D

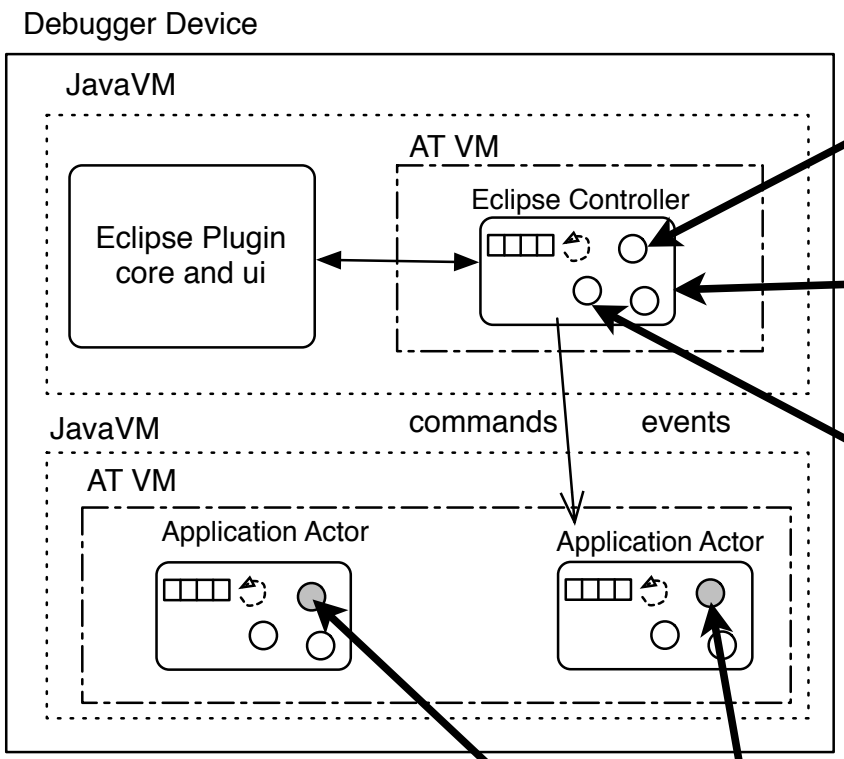
- More info at:

  http://code.google.com/p/ambienttalk/wiki/DebuggingInEclipse

- For the session:
  - Don't forget to open the Debug Element View.
  - Be patient with the Eclipse GUI :)

REME-D
from a programmer perspective

REME-D Architecture



debugger manager

listener for commands from eclipse GUI

listener for events from debugger manager

Each actor being debug carries a local manager

Debugger Manager

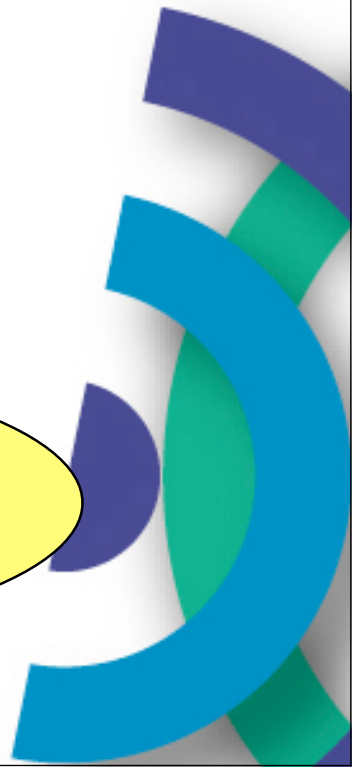
- Manages the debugging session:
  - Keeps list of actors in debug.
  - Enables communication with GUI and local managers:
    - Local Interface
    - Remote interface

Debugger Manager

methods called from
`EclipseController.commandBehaviour`

```
def localInterface := object: {
    def breakpointActiveOn(filename, lineNumber, listVMIds);
    def clearBreakpoint(filename, lineNumber);
    def setBreakpoint(filename, lineNumber, messageResolved := false);
    def stepReturn(actorId);
    def stepOver(actorId);
    def stepInto(actorId);
    def resumeActor(actorId);
    def pauseActor(actorId);
    def loadMainCode(actorId);
    def setupDebugSession(debugEventList);
    def getLocalManagerById(localManagerId);
    def listLocalManagers();
};
```

accessible from Eclipse
Plugin in AT Debugger Manager
Console



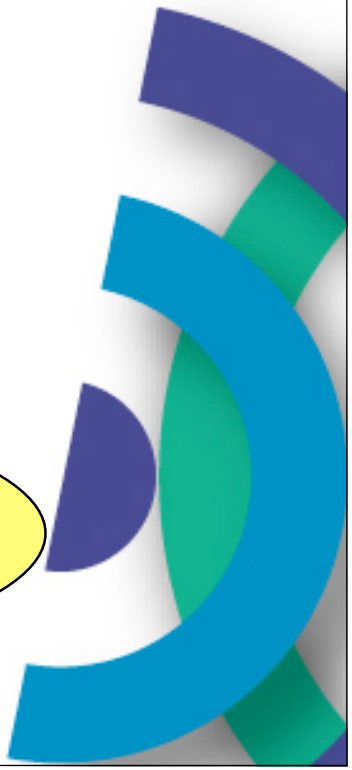
```
eclipseController.debuggerSessionBhv.stepOver(-1253);
```

Debugger Manager

methods called
from
localManager

```
def remoteInterface := object: {
    def actorStarted(actorId, sourceLocation, frLocalManager);
    def actorPaused(actorId,actorState);
    def actorResumed(actorId);
    def updateInbox(actorId, msg, addition := true);
    def updateMessageSent(actorId, msg);
};
```

methods
in turn call
EclipseController.debuggerEventListener



Local Manager

```
def interfaceDebuggerManager := object: {
 def evaluateCode();
 def startInDebugMode(tableCodeBreakpoints);
 def pause();
 def resume();
 def stepInto();
 def stepReturn();
 def stepOver();
 def stepReturn();
 def stepInto();
 def addBreakpoint(breakpoint);
 def removeBreakpoint(breakpoint);
};
```

methods called from **debuggerManager**

Local Manager

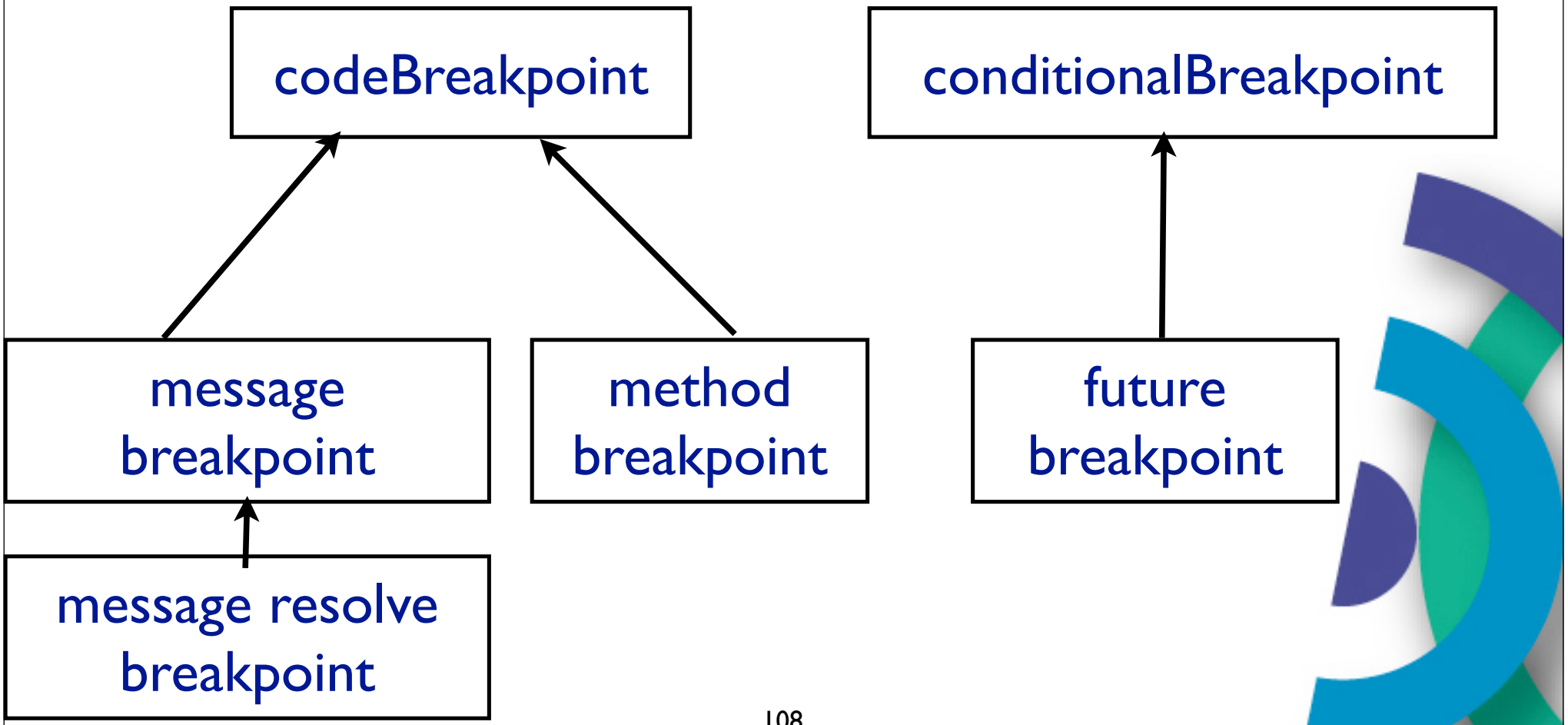
- Actor protocol overriding 3 meta-level methods:

```
extend: actor with: {
   def actorId := debuggerUtilModule.generateRandomId();

   def debuggingState := INITIAL;
   def pausedState := INITIAL;

   def debuggerManager := nil;

   def inbox := [];

   def senderBreakpoints := atHashMap.new();
   def receiverBreakpoints := atHashMap.new();

   ...

   def send(rcv, msg);
   def schedule(rcv, msg);
   def serve();

   def interfaceDebuggerManager := object: { ...}

   };
};
```

debugging
related messages annotated with
@Debug

Breakpoints

must import
the breakpoint trait



Breakpoints

```
def TBreakpoint := isolate:{
        def condition;
        def breakpointId;
        def breakpointTypes;

        def init(cond, types, bId := /.at.support.debugger.util.generateRandomId()){
            self.breakpointId := bId;
            self.condition := cond;
            self.breakpointTypes := types;
        };
        def getBreakpointId() {self.breakpointId};
        def onEntry() {true};
        def ==(otherBreakpoint) {
            self.getBreakpointId == otherBreakpoint.getBreakpointId
        };
        def matches(rcv,msg) {
            self.condition(rcv,msg);
        };
        def print() {
            "<" + self.toString() + ":taggedAs:" + self.breakpointTypes + ">";
        };
        def getBreakpointTypeTags() { self.breakpointTypes };
        def isTaggedAs(typeTag) {
            { |return|
                self.breakpointTypes.each: { |t|
                    if: t.isSubtypeOf(typeTag) then: { return
                };
                false;
            }.escape();
        };
    };
```

closure takes as parameter message and receiver

SenderBreakpoint and/or ReceiverBreakpoint

Breakpoints

```
def methodBreakpoint := extendIsolate: codeBreakpoint with:{
      def init(name, number) {
            def cond := script: { |rcv, msg|
                  def res := false;
                  if: ((reflect: rcv).respondsTo(msg.selector)) then: {
                        def method :=  (reflect: rcv).grabMethod(msg.selector);
                        def sourceLocation := /.at.support.util.getSourceLocation(method);
                        if: (nil != sourceLocation) then: {
                              if: ((name == sourceLocation.fileName).and:{number == sourceLocation.line}) then: {
                                    res := true;
                              };
                        };
                  };
                  res;
            } carrying: `[name, number];
            super^init(name, number, cond, [/.at.support.debugger.util.ReceiverBreakpoint]);
      };
      def toString() {
            "method-breakpoint:"+ super^toString();
      };
};
```

REME-D

- For the session:
  - Don’t forget to make the Eclipse plugin point to the atlib in your project.