JASON

Rahul Arora

Research lab – Summer term 2014

University Koblenz-Landau

Features

1) Implement KQML based inter-agent communication

    .send(+receiver, +illocutionary_force, +prop-content)

3) Fully customisable selection functions, trust functions(in JAVA)

4) Extensibility by user defined internal actions.

Interpretation Cycle

1) At every interpretation cycle interpreter updates a list of events
2) Believe revision function updates both external and internal event(changes in agent belief)
3) Event is selected with the help of selection function(Se)
4) Selection function(So) chooses the single applicable plan
5) Si function selects one the agent's intention

Scenario

```
MAS heathrow{
                infrastructure: centralised
                environment: Heathrow
                agents:
                      mds agentClass mds.MDSAgent
                           #5;
                       cph agentArchClass cph.CPHagArch
                              agent Class cph.CPHAgent
                                #10;

}
```

Creating Environment

```
public class HeathrowEnv extends Environment {
public List getPercepts(String agName) {
if ( ... unattended luggage has been found ... ) {
// all agents will perceive the fact that there is unattendedLuggage
getPercepts().add(Term.parse("unattendedLuggage"));
}
if (agName.startsWith("mds")) {
// mds robots will also perceive their location and code to implement that
}
else {
return getPercepts();
}}
```

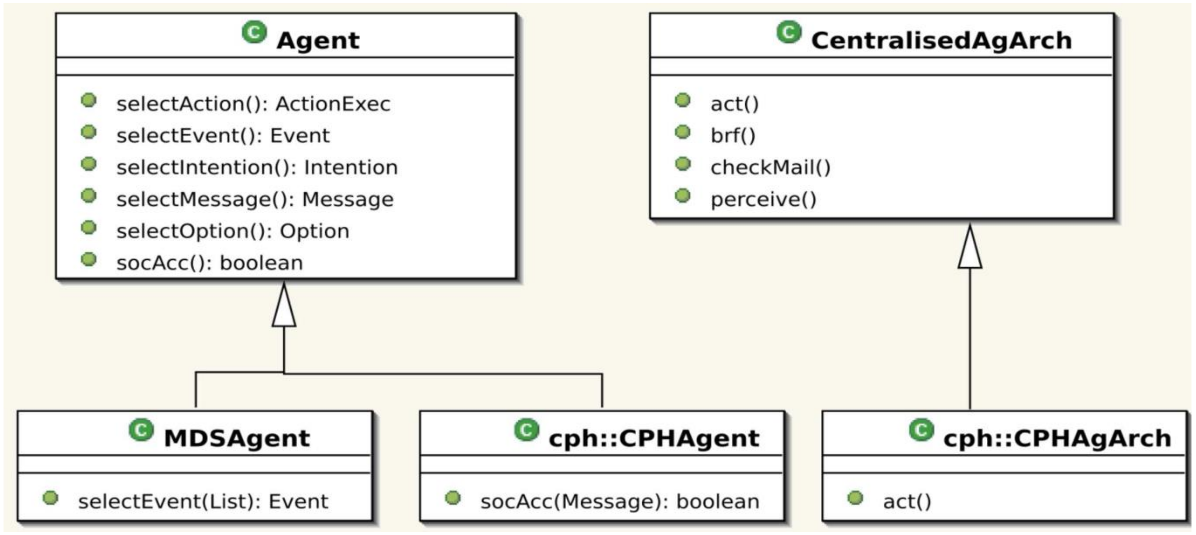
Creating Environment(1/2)

```
public boolean executeAction(String ag, Term action) {
if (action.hasFunctor("disarm")) {
... the code that implements the disarm action
... on the environment goes here
} else if (action.hasFunctor("move")) {
... the code for changing the agents’ location and
... updating the agsLocation map goes here
}
return true;
}
}
```

Customising Agents

1) Certain aspects of agents can be customised by overriding methods of **Agent** class such as Social acceptance(soc Acc) function for the trust and power social relations.

<u>Example</u>:- a CPH903 robot only does what an MDS79 robot asks:-



```
package cph;
import jason.asSemantics.Agent;
public class CPHAgent extends Agent {
public boolean socAcc(Message m) {
if (m.getSender().startsWith("mds") && m.getIlForce().equals("achieve")) {
return true;
} else {
return false;
}}}
```

Customising Agents (1/2)

User can also customize the functions defining the overall agent architecture in the following ways:-

a) The way the agent will perceive the environment

b) the way it will update its belief base given the current perception of the environment, i.e., belief revision function (BRF)

c) how the agent gets messages sent from other agents

Internal Actions

1) Internal actions that start with ‘.’ are part of a standard library of internal actions

2) In the AgentSpeak program, the action is accessed by the name of the library, followed by ‘.’, followed by the name of the action

   Example:- mds.calculateMyBid(Bid):

3) Libraries are defined as Java packages and each action in the user library should be a Java class

Internal Actions (1/2)

Example:-

```
package mds;
import ...
public class calculateMyBid implements InternalAction {
public boolean execute(TransitionSystem ts, Unifier un,
Term[] args) throws Exception {
int bid = ... a complex formula ...;
... plus complex algorithm and calculations for adjusting the agent’s bid ...
un.unifies(args[0], Term.parse(""+bid));
return true;
}
```

Available Tools

1) JASON provides IDE (Jedit) which provides a GUI for editing a MAS configuration file as well as Agent-Speak code for the individual agents

2) There are three execution modes.

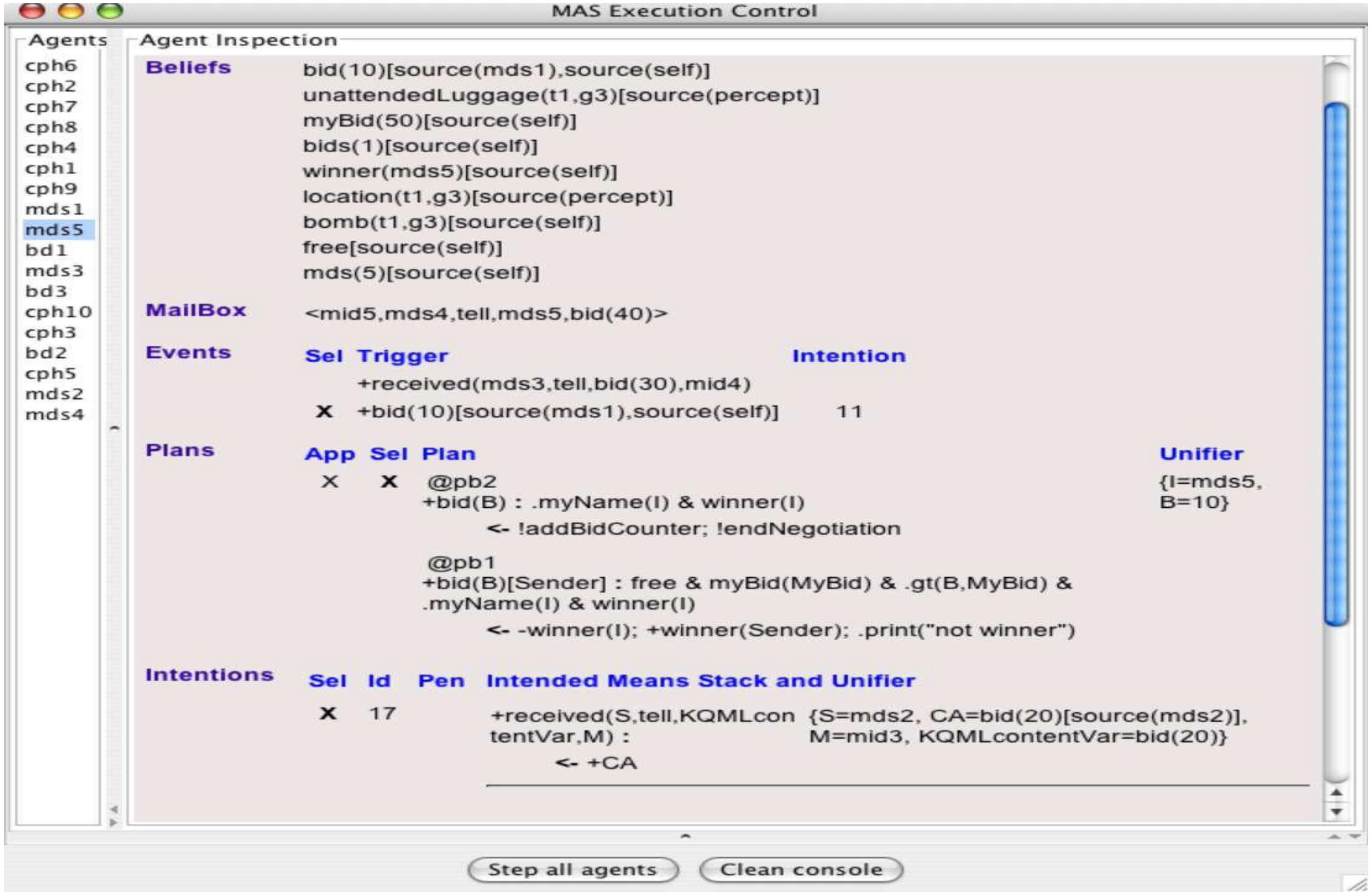
   a) Asynchronous

   b) Synchronous

   c) Debugging

3) There is another tool provided as part of the IDE which allows the userto inspect agents' internal states when the system is running in debugging mode. The tool is called "mind inspector"

Mind inspector Screenshot



```
cph6
cph2
cph7
cph8
cph4
cph1
cph9
mds1
mds5
bd1
mds3
bd3
cph10
cph3
bd2
cph5
mds2
mds4
```

```
Beliefs      bid(10)[source(mds1),source(self)]
             unattendedLuggage(t1,g3)[source(percept)]
             myBid(50)[source(self)]
             bids(1)[source(self)]
             winner(mds5)[source(self)]
             location(t1,g3)[source(percept)]
             bomb(t1,g3)[source(self)]
             free[source(self)]
             mds(5)[source(self)]

MailBox      <mid5,mds4,tell,mds5,bid(40)>

Events       Sel  Trigger                                          Intention
                  +received(mds3,tell,bid(30),mid4)
              X   +bid(10)[source(mds1),source(self)]              11

Plans        App  Sel  Plan                                                       Unifier
              X    X   @pb2                                                       {I=mds5,
                       +bid(B) : .myName(I) & winner(I)                           B=10}
                            <- !addBidCounter; !endNegotiation

                       @pb1
                       +bid(B)[Sender] : free & myBid(MyBid) & .gt(B,MyBid) &
                       .myName(I) & winner(I)
                            <- -winner(I); +winner(Sender); .print("not winner")

Intentions   Sel  Id   Pen  Intended Means Stack and Unifier
              X   17        +received(S,tell,KQMLcon  {S=mds2, CA=bid(20)[source(mds2)],
                            tentVar,M) :               M=mid3, KQMLcontentVar=bid(20)}
                                 <- +CA
```