Using *Jason* to Implement a Team of Gold Miners

Rafael H. Bordini[1], Jomi F. Hübner[2], and Daniel M. Tralamazza

[1]University of Durham (UK)
R.Bordini@durham.ac.uk

[2]University of Blumenau (Brazil)
jomi@inf.furb.br

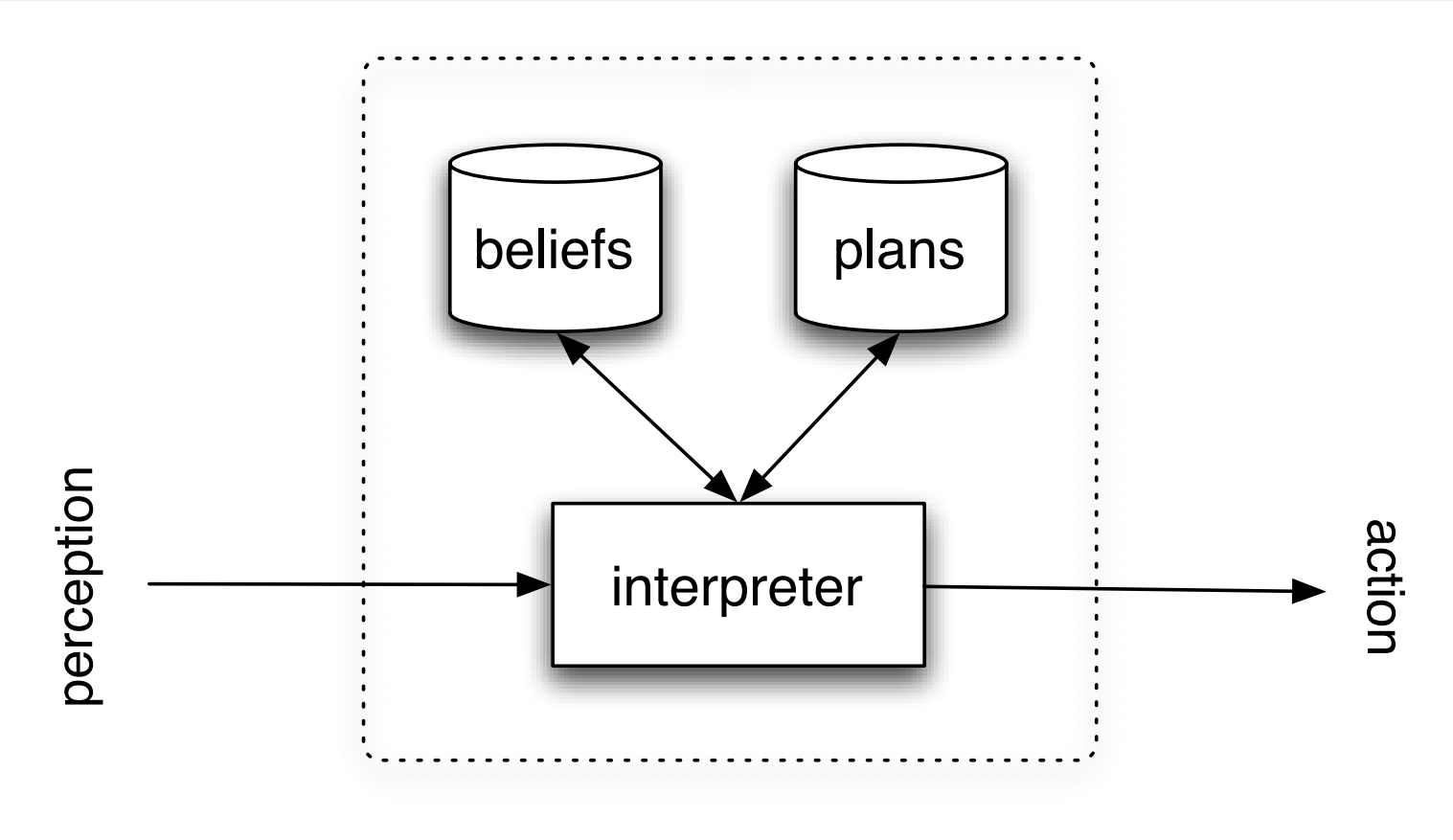
[3]EPFL (Switzerland)
tralamazza@gmail.com

CLIMA Contest 2006 – Hakodate, Japan

AgentSpeak(L)

- Originally proposed by Rao (1996)
- Programming language for BDI agents
- Natural use of logic programming for reactive planning systems
- Influential in the design of other languages
- However, originally only an abstract language; various extensions were needed to make it a practical language

Basic Architecture of an AgentSpeak agent



Basic AgentSpeak Syntax

- Beliefs: predicateSymbol(ground,terms). (a literal)
- Plans: event : context <- body.
- Event:
  - +belief / -belief
  - +!goal / -!goal
  - +?testgoal / -?testgoal
- Context: literal & ... & literal
- Body:
  - action: action(ground,terms).
  - achievement goal: !newGoal
  - test goal: ?belief
  - belief addition: +literal
  - belief deletion: -literal

Example

```
+green_patch(Rock) :
   not battery_charge(low) <-
      ?location(Rock,Coordinates);
      !traverse(Coordinates);
      !examine(Rock).

+!traverse(Coords) :
   safe_path(Coords) <-
      move_towards(Coords).

+!traverse(Coords) :
   not safe_path(Coords) <-
      ...
```

Jason features (I)

- An interpreter for an extended version of AgentSpeak
- Distribution over the net using SACI
- Implements the operational semantics of AgentSpeak
- Some of its features are:
  - strong negation, so both closed-world assumption and open-world are available
  - speech-act based inter-agent communication (and annotation of beliefs with information sources)
  - handling of plan failures

Jason features (II)

- annotations on plan labels, which can be used by elaborate (e.g., decision-theoretic) selection functions
- support for developing Environments (in Java)
- fully customisable (in Java) selection functions, trust functions, and overall agent architecture (perception, belief-revision, inter-agent communication, and acting)
- a library of essential “internal actions”
- straightforward extensibility by user-defined internal actions, programmed in Java

Language Extensions: Internal Actions

- Internal actions can be defined by the user in Java

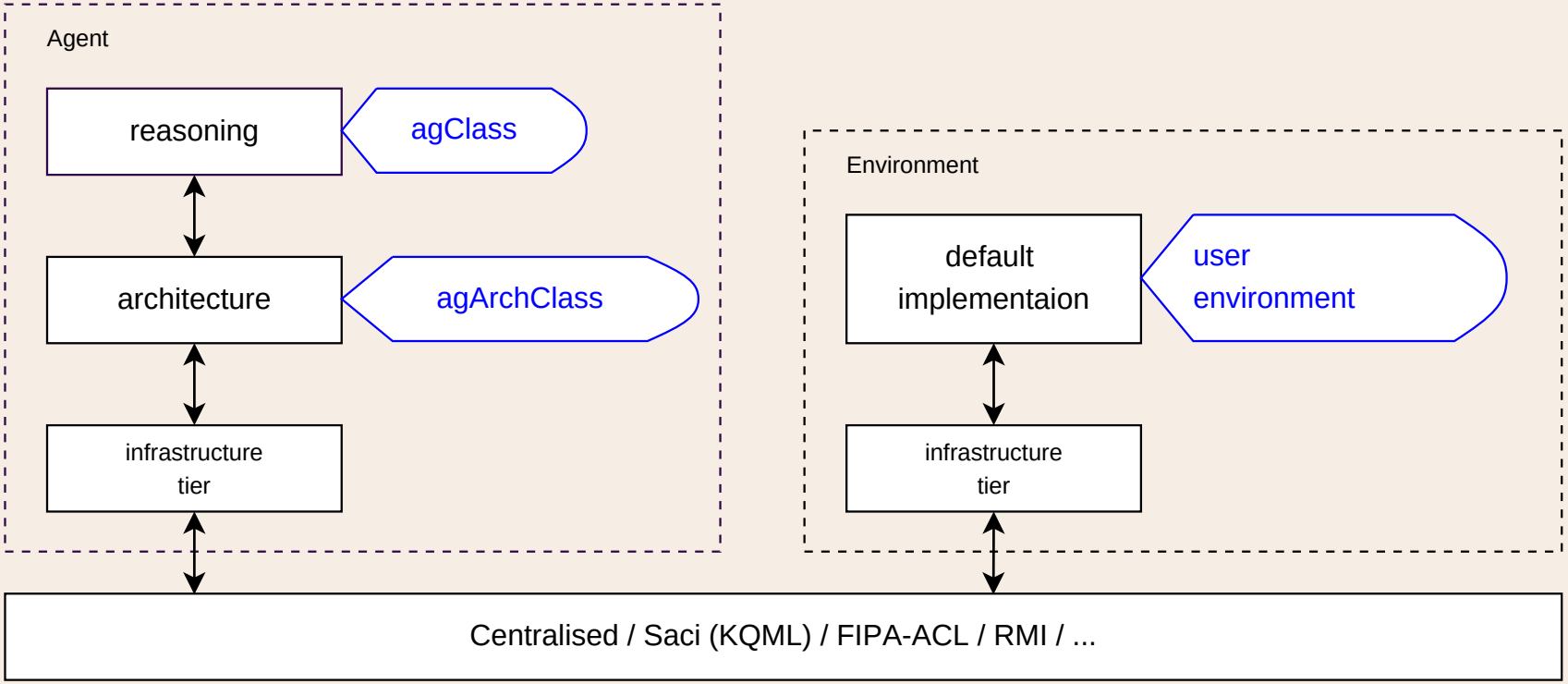
  `libName.actionName(...)`

- Standard (pre-defined) internal actions have an empty library name
  - `.print(`$term_1, term_2, \ldots$`)`
  - `.myName(`*var*`)`
- Internal action for communication:

  `.send(`*r*`,`*ilf*`,`*pc*`)`

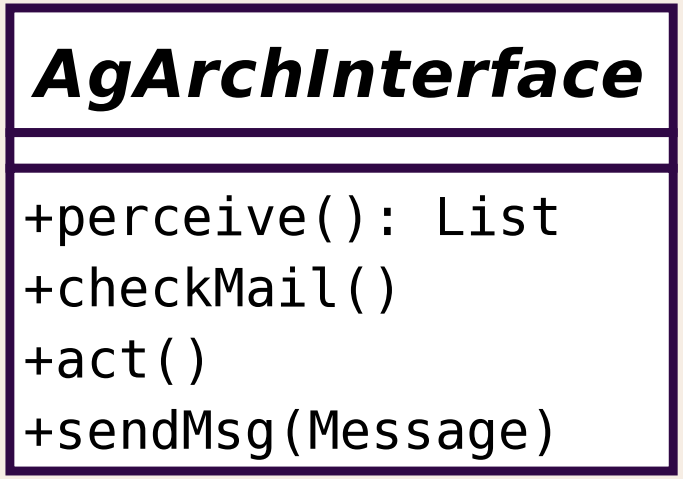
  where *ilf* $\in$ {`tell`, `untell`, `achieve`, `unachieve`, `tellHow`, `untellHow`, `askIf`, `askOne`, `askAll`, `askHow`}

Infrastructure



Customising an Overall Agent Architecture

- Users can define a specific overall (rather than reasoning) architecture for an agent
- This is used to customise the way the agent does perception of the environment, receives communication massages, does belief revision, and acts in the environment
- Customised to connect to the CLIMA server



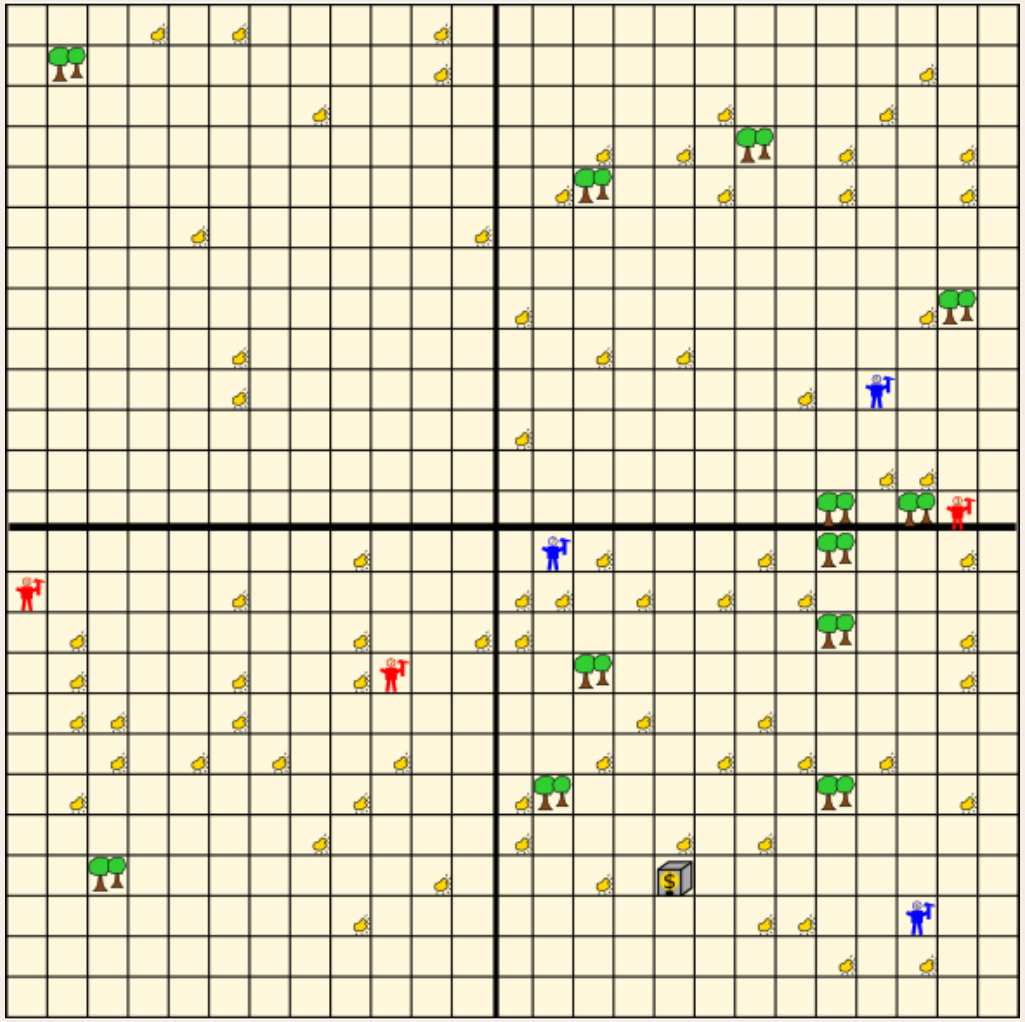
Jason is available
Open Source
under GNU LGPL at:

`jason.sourceforge.net`

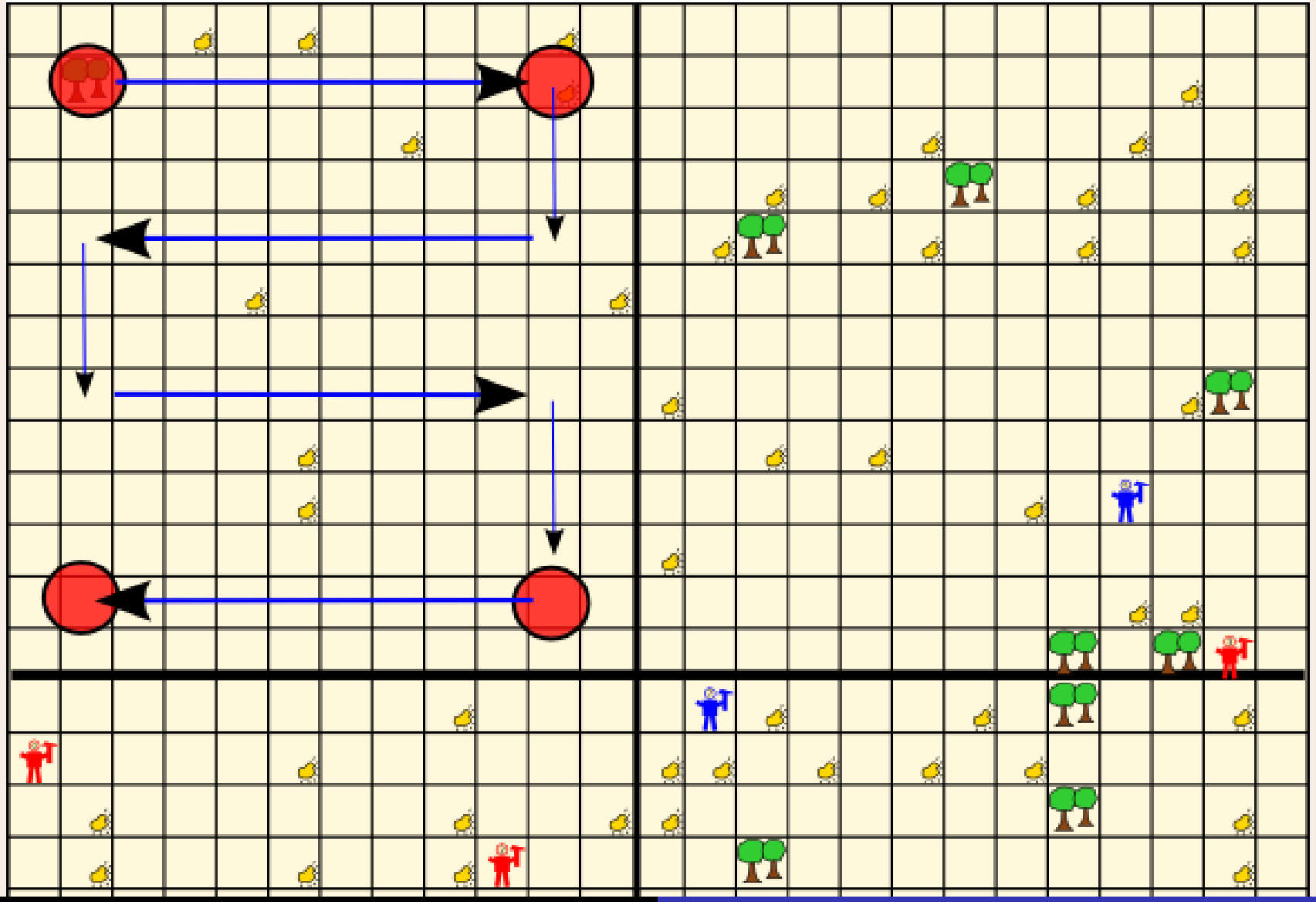
(kindly hosted by SourceForge)

Jason
by *Gustave Moreau* (1865)
Oil on canvas, 204 x 115.5 cm.
Musée d'Orsay, Paris.
© Photo RMN. Photograph by
Hervé Lewandowski.

Quadrant allocation



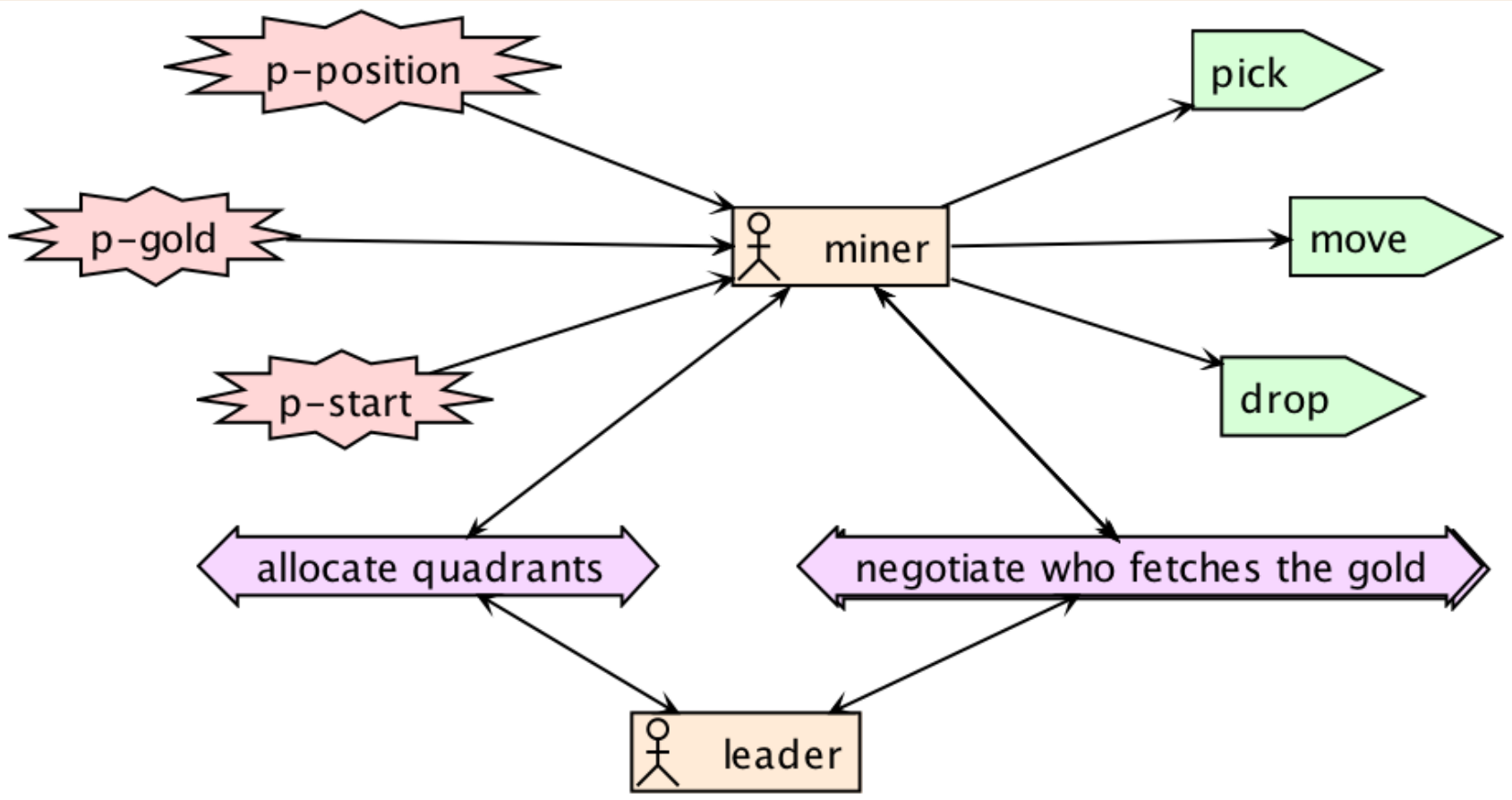
Wandering



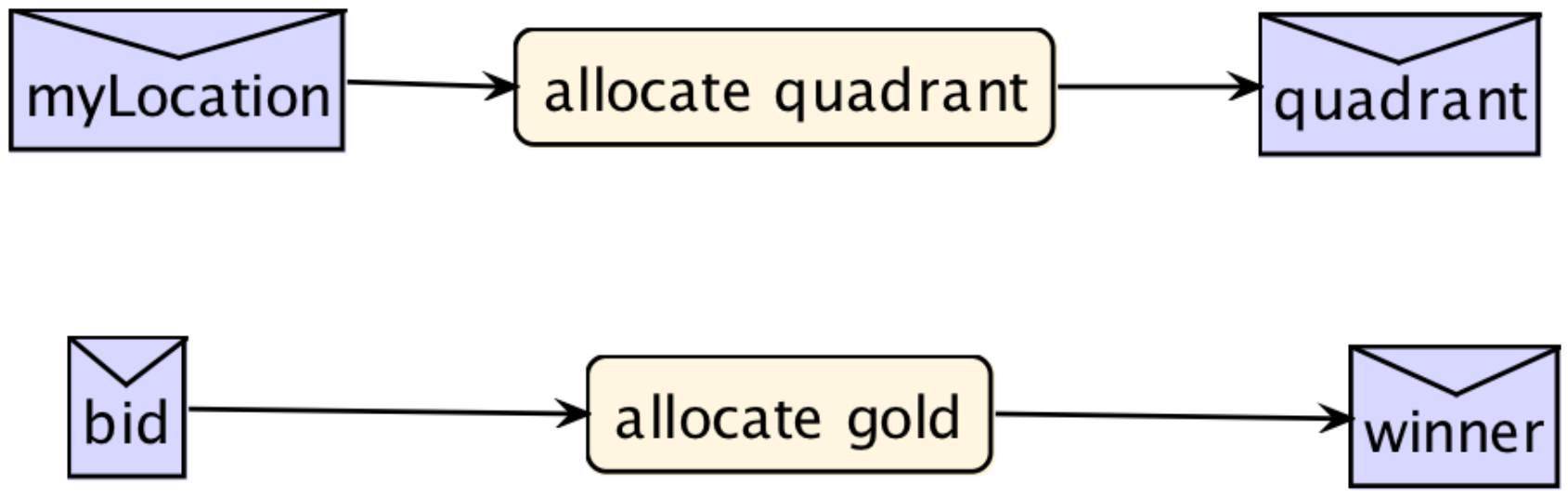
Gold handling

- When a miner sees a piece of gold:
  - if free (not committed to another gold): pick it up and carry to depot
  - if not free and not carrying gold (committed to another gold, but has not collected it yet): give up gold last committed and pick that one up
  - if already carrying gold: announce to others
- When another agent announces more gold has been found:
  - if free: bid based on Manhattan distance
  - if allocated by leader: go to gold, pick it up, and carry to depot

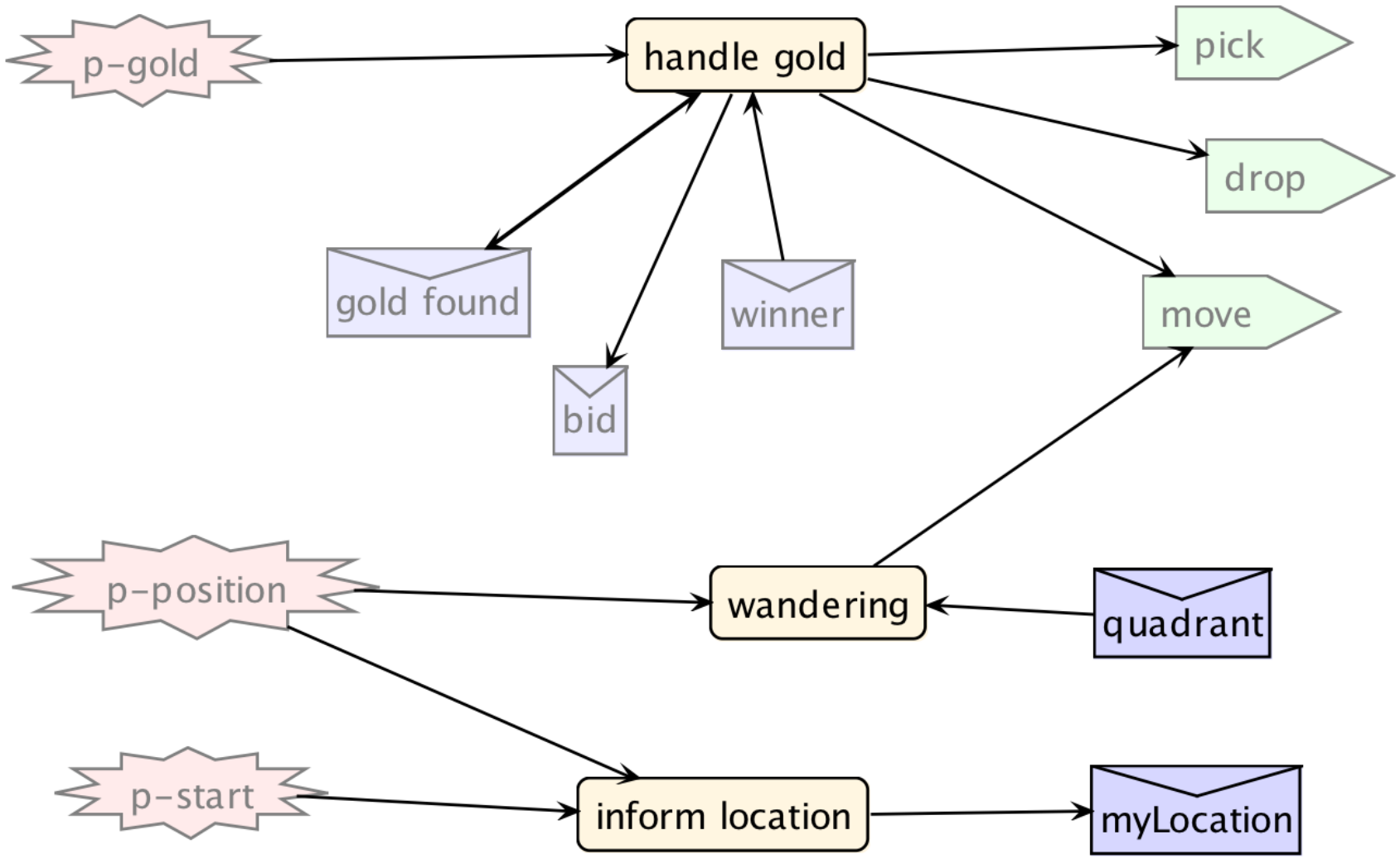
Prometheus Design: Overview



Prometheus Design: Leader



Prometheus Design: Miner



Miner: gold handling I

```
+cell(X,Y,gold) : not carrying_gold & free
  <- -free;
     +gold(X,Y);
     !init_handle(gold(X,Y)).
+cell(X,Y,gold)
  :  not gold(X,Y) & not carrying_gold & not free &
     .desire(handle(gold(OldX,OldY))) &
  <- +gold(X,Y);
     .dropIntention(handle(gold(_,_)));
     .broadcast(tell,gold(OldX,OldY));
     !init_handle(gold(X,Y)).
+cell(X,Y,gold)
  :  not gold(X,Y) & not committed(gold(X,Y))
  <- +gold(X,Y);
     .broadcast(tell,gold(X,Y)).
```

Miner: gold handling II

```
+!handle(gold(X,Y)) : true
  <- .broadcast(tell, committedTo(gold(X,Y)));
     !pos(X,Y);
     !ensure(pick);
     .broadcast(tell,picked(gold(X,Y)));
     ?depot(_,DX,DY);
     !pos(DX,DY);
     !ensure(drop);
     -gold(X,Y);
     !!choose_gold.

-!handle(G) : G
  <- -G;
     !!choose_gold.
```

Miner: gold handling III

```
+!choose_gold : not gold(_,_) <- +free.
+!choose_gold : gold(_,_)
  <- .findall(gold(X,Y),gold(X,Y),LG);
     !calcGoldDistance(LG,LD);
     .sort(LD,[d(Distance,NewG)|_]);
     !!handle(NewG).

+!calcGoldDistance([],[]) : true <- true.
+!calcGoldDistance([gold(GX,GY)|R],
                   [d(D,gold(GX,GY))|RD])
  :  pos(IX,IY) & not committedTo(gold(GX,GY))
  <- jia.dist(IX,IY,GX,GY,D);
     !calcGoldDistance(R,RD).
+!calcGoldDistance([_|R],RD) : true
  <- !calcGoldDistance(R,RD).
```

Miner: moving (using A*)

```
+!pos(X,Y) : pos(X,Y) <- true.
+!pos(X,Y) : not pos(X,Y)
  <- !next_step(X,Y);
     !pos(X,Y).
+!next_step(X,Y)
  :  pos(AgX,AgY)
  <- jia.getDirection(AgX, AgY, X, Y, D);
     do(D).
```

Leader: gold allocation

```
+bidFor(Gold,Distance)[source(M1)]
  :  bidFor(Gold,_)[source(M2)] &
     bidFor(Gold,_)[source(M3)] &
     M1 \== M2 & M1 \== M3 & M2 \== M3
  <- !allocateMinerFor(Gold).

+!allocateMinerFor(Gold) : true
  <- .findall(op(D,Ag),bidFor(Gold,D)[source(Ag)],LD);
     .sort(LD,[op(_,CloserAg)|_]);
     .broadcast(tell,allocatedTo(Gold,CloserAg)).

-!allocateMinerFor(Gold) : true
  <- .print("could not allocate gold ",Gold).
```

Conclusions

- AgentSpeak is suitable for the problem:
  - elegant declarative solution
  - reactiveness to dynamic environment
- ***Jason*** implementation provided good support for:
  - high-level communication
  - integration with the contest simulator
  - using external Java code (e.g., A*)
- Difficulties:
  - new paradigm
  - some bugs in ***Jason*** (now mostly fixed!)
  - some difficulties with concurrent intentions
  - we did not do well in the scenarios with too much uncertainty (but possibly lack of time/experience)