Reactive Action Programs

- BDI-Agents
- Programming in AgentSpeak
- AgentSpeak in Prolog
- Java-Based AgentSpeak (JASON)
- The SPARK Programming Language

Reactive Agents

- Procedural action programs

  - describe complex, long-term strategies

  - are suitable for agents in well-structured environments over which they have sufficient control

- Reactive action programs

  - describe short-term behaviors

  - are suitable for agents in highly dynamic, uncontrollable environments for which long-term strategies are diffcult or even impossible

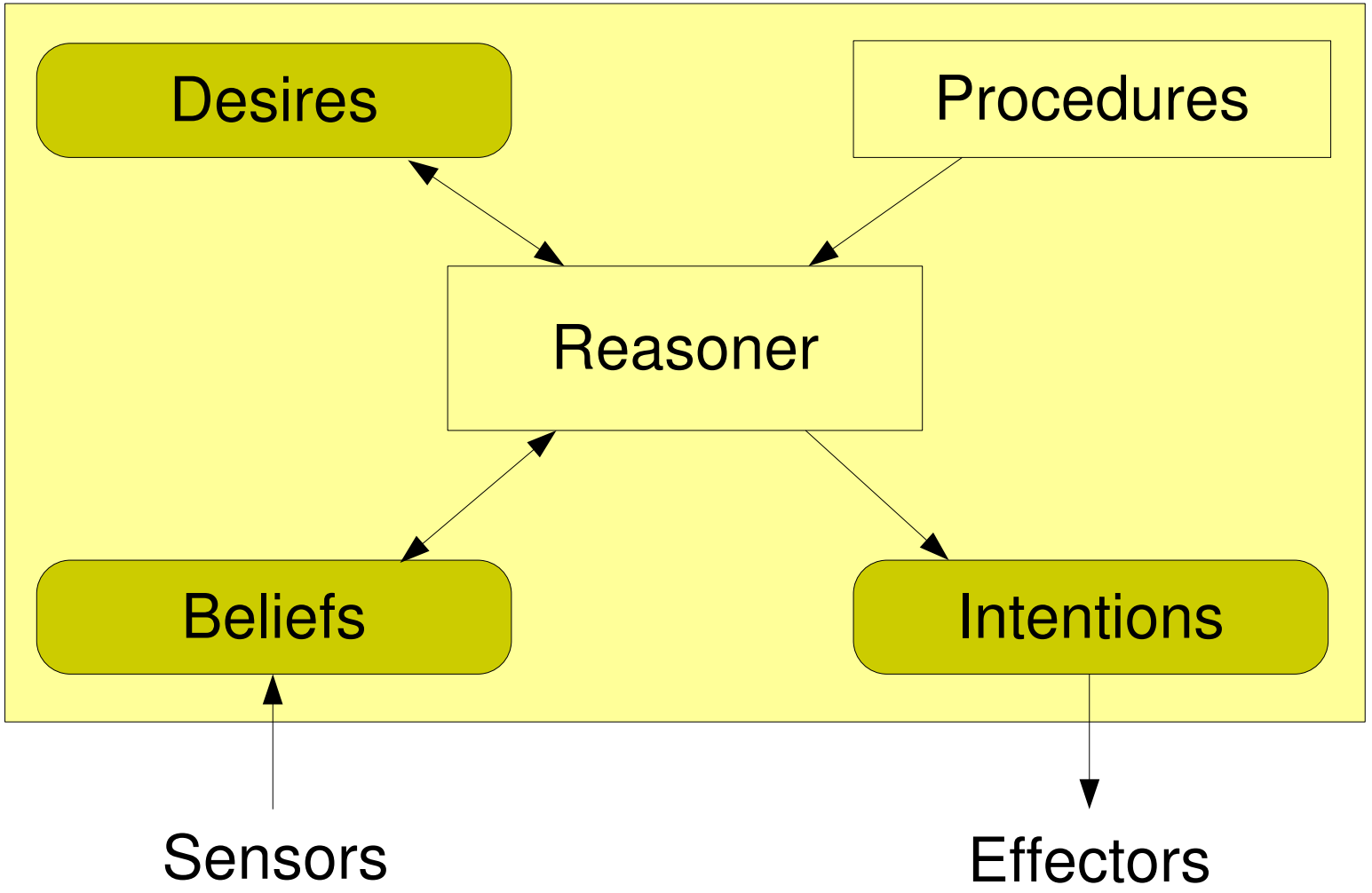
  - Examples: simple agents in an open world
    robots in a largely unknown environment

BDI-Agents

The state of an agent at any time is characterized by its

- Beliefs, which constitute the internal world model. They describe what the agent currently believes about the environment and its own tasks.
- Desires, which are derived from the beliefs and describe what the agent currently tries to achieve.
- Intentions, which are the behaviors (procedures) which the agent has adopted and is currently following in order to meet its desires.

Procedural Reasoning Systems (PRS)



Beliefs

- Beliefs are composed of fluents
- Beliefs are affected by actions and sensor information
- Beliefs may include static knowledge (properties of the environment that do not change)
- Beliefs may also concern the “behavioral stance” of the agent

Desires

- Desires are properties which the agent currently wants to achieve
- Desires are (typically short-term) goals

Procedures

- Procedures constitute the behavioral knowledge of the agent
- Procedures correspond to (typically short) GOLOG procedures or ALP clauses
- Each procedure has as condition a desire for which it is suitable

Intentions

- Intentions are instantiated procedures which the agent has selected to meet its desires
- Intentions can be active or passive
- Actions are selected from active intentions
- Intention ordering: Intentions on top are active, all others remain passive until the preceding intentions have been completed

Recap: Agent Cycle

Reasoner

- Controls the execution of a reactive action program
- Sense-Select-Act cycle
  - Sense the environment
  - Select procedures for desires; select an active intention
  - Act according to the selected intention

AgentSpeak

Syntax

An AgentSpeak domain signature includes a finite set of belief predicates and a finite set of action predicates.

A belief literal is an atom with a belief predicate or its negation.

Example: Gold Mining with Dynamic Obstacles

Symbol	Type
Adjacent	CELL $\times$ CELL
At	{*Agent*, *Gold*, *Obstacle*, *Depot*} $\times$ CELL
Carries	{*Agent*} $\times$ {*Gold*}

Adjacent(*A*, *B*)

Adjacent(*B*, *C*)

At(*Agent*, *A*)

At(*Gold*, *B*)

At(*Depot*, *C*)

Symbol	Type	Meaning
Pick	{*Gold*}	pick up a gold item
Drop	{*Gold*}	drop a gold item
Move	CELL $\times$ CELL	move to an adjacent cell

Triggering Events

If *f* is a belief atom, then !*f* and ?*f* are goals

A triggering event is any of

- +*f*
- –*f*
- +g
- –*g*

where *f* belief atom, *g* goal

Procedures

A procedure is an expression

$$e : b_1, ..., b_m \leftarrow p_1, ..., p_n$$

where

- e triggering event
- context $b_1, ..., b_m$ belief literals ($m \geq 0$)
- body $p_1, ..., p_n$ action atoms or goals ($n \geq 0$)

Empty context or body denoted by *True*

Example

$+At(Gold, x) : At(Agent, x), At(Depot, y) \leftarrow Pick(Gold), !At(Agent, y), Drop(Gold)$

$+!At(Agent, x) : At(Agent, x) \leftarrow True$

$+!At(Agent, x) : At(Agent, y), x \neq y, Adjacent(y, z), \neg At(Obstacle, z)$

$\leftarrow Move(y, z), !At(Agent, x)$

Operational Semantics

State of AgentSpeak program characterized by triple

- B: a set of variable-free belief atoms
- I: a set of intentions of the form
  $[P_1; ...; P_k]$
  where each P_i procedure body (possibly partially instantiated)
- D: a set of desires of the form
  $\langle e ; i \rangle$
  where e triggering event, i intention

  External desire: $\langle e ; [\,] \rangle$

Selection Functions

- S_D selects an element from the current desires
- S_I selects an element from the current intentions
- S_P selects an applicable procedure for a triggering event

Relevance / Applicability

B – set of variable-free belief atoms
e – triggering event
P – procedure $d : b_1, ..., b_m \leftarrow p_1, ..., p_n$

P relevant for $e \stackrel{\text{def}}{=} d\theta = e\theta$ for some θ

$P\theta\eta$ applicable to e wrt. B $\stackrel{\text{def}}{=}$ B $\models (b_1 \wedge ... \wedge b_m)\theta\eta$

Example

$Adjacent(A, B)$

$Adjacent(B, C)$

$At(Agent, A)$

$At(Gold, A)$

$At(Depot, C)$

$+!At(Agent, x) : At(Agent, x) \leftarrow True$

$+!At(Agent, x) : At(Agent, y), x \neq y, Adjacent(y, z), \neg At(Obstacle, z)$

$\leftarrow Move(y, z), !At(Agent, x)$

Both procedures are relevant for $+!At(Agent, C)$.

The second procedure is also applicable with $\theta = \{x/C\}$ and $\eta = \{y/A, z/B\}$

Derivations: States

$\langle B, D, I, \sigma \rangle$ state, where $\sigma \in \{\texttt{Sense}, \texttt{Select}, \texttt{Act}\}$

$\langle B, \{\}, \{\}, \texttt{Sense} \rangle$ initial state

Derivation Rules (1/6)

$$\frac{\langle B, D, I, \mathit{Sense} \rangle}{\langle B', D', I, \mathit{Select} \rangle}$$

$B' \stackrel{\text{def}}{=}$ B updated according to sensing result

$D' \stackrel{\text{def}}{=}$ D plus external desires that have been sensed

Derivation Rules (2/6)

$$\frac{\langle \mathsf{B}, \{\}, \mathsf{I}, \mathit{Select} \rangle}{\langle \mathsf{B}, \{\}, \mathsf{I}, \mathit{Act} \rangle}$$

If S_D (D) = ⟨*e; i* ⟩ and no relevant procedure exists:

$$\frac{\langle \mathsf{B}, \mathsf{D}, \mathsf{I}, \mathit{Select} \rangle}{\langle \mathsf{B}, \mathsf{D} \setminus \{\langle \mathsf{e}; i \rangle\}, \mathsf{I}, \mathit{Select} \rangle}$$

If S_D (D) = ⟨*e; i* ⟩ and relevant but no applicable procedures exist:

$$\frac{\langle \mathsf{B}, \mathsf{D}, \mathsf{I}, \mathit{Select} \rangle}{\langle \mathsf{B}, \mathsf{D}, \mathsf{I}, \mathit{Act} \rangle}$$

Derivation Rules (3/6)

If S_D (D) = $\langle e; i \rangle$ and $S_P(e) = P\theta\eta$:

(a) For external desires

$$\frac{\langle \mathsf{B}, \mathsf{D}, \mathsf{I}, \mathit{Select} \rangle}{\langle \mathsf{B}, \mathsf{D} \setminus \{\langle \mathsf{e}; [\,] \rangle\}, \mathsf{I} \cup \{[P\theta\eta]\}, \mathit{Act} \rangle}$$

(b) For internal desires

$$\frac{\langle \mathsf{B}, \mathsf{D}, \mathsf{I}, \mathit{Select} \rangle}{\langle \mathsf{B}, \mathsf{D} \setminus \{\langle \mathsf{e}; i \rangle\}, \mathsf{I} \cup \{[P\theta\eta; P_1; \ldots; P_k]\}, \mathit{Act} \rangle}$$

where $i = [P_1, \ldots, P_k]$

Derivation Rules (4/6)

$$\frac{\langle \mathsf{B}, \mathsf{D}, \{\}, \mathit{Act} \rangle}{\langle \mathsf{B}, \mathsf{D}, \{\}, \mathit{Sense} \rangle}$$

If $S_I(\mathsf{I}) = [a, P_1; \ldots; P_k]$:

$$\frac{\langle \mathsf{B}, \mathsf{D}, \mathsf{I}, \mathit{Act} \rangle}{\langle \mathsf{B}', \mathsf{D}, \mathsf{I} \setminus \{[a, P_1; \ldots; P_k]\} \cup \{[P_1; \ldots; P_k]\}, \mathit{Sense} \rangle}$$

$\mathsf{B}' \stackrel{\text{def}}{=}$ B updated according to effects of a

Derivation Rules (5/6)

If $S_I(\mathsf{I}) = [!f, P_1; \ldots; P_k]$:

$$\frac{\langle \mathsf{B}, \mathsf{D}, \mathsf{I}, Act \rangle}{\langle \mathsf{B}, \mathsf{D} \cup \{\langle +!f; [P_1; \ldots; P_k] \rangle\}, \mathsf{I} \setminus \{[!f, P_1; \ldots; P_k]\}, Sense \rangle}$$

If $S_I(\mathsf{I}) = [?f, P_1; \ldots; P_k]$ and $\mathsf{B}' \models f\theta$ for some θ:

$$\frac{\langle \mathsf{B}, \mathsf{D}, \mathsf{I}, Act \rangle}{\langle \mathsf{B}, \mathsf{D}, \mathsf{I} \setminus \{[?f, P_1; \ldots; P_k]\} \cup \{[P_1; \ldots; P_k]\theta\}, Sense \rangle}$$

If $S_I(\mathsf{I}) = [?f, P_1; \ldots; P_k]$ and $\mathsf{B}' \not\models f\theta$ for all θ:

$$\frac{\langle \mathsf{B}, \mathsf{D}, \mathsf{I}, Act \rangle}{\langle \mathsf{B}, \mathsf{D} \cup \{\langle +?f; [P_1; \ldots; P_k] \rangle\}, \mathsf{I} \setminus \{[?f, P_1; \ldots; P_k]\}, Sense \rangle}$$

Derivation Rules (6/6)

If $S_I(\mathsf{I}) = [\mathit{True}; P_2; \ldots; P_k]$:

$$\frac{\langle \mathsf{B}, \mathsf{D}, \mathsf{I}, \mathit{Act} \rangle}{\langle \mathsf{B}, \mathsf{D}, \mathsf{I} \setminus \{[\mathit{True}; P_2; \ldots; P_k]\} \cup \{[P_2; \ldots; P_k]\}, \mathit{Sense} \rangle}$$

If $S_I(\mathsf{I}) = [\,]$:

$$\frac{\langle \mathsf{B}, \mathsf{D}, \mathsf{I}, \mathit{Act} \rangle}{\langle \mathsf{B}, \mathsf{D}, \mathsf{I} \setminus \{[\,]\}, \mathit{Sense} \rangle}$$

Example

B = {*At*(*Agent*, *A*), *At*(*Gold*, *A*), *At*(*Depot*, *C*),*Adjacent*(*A*,*B*)}
D = { }
I = { }
σ = `Sense`

B = {*At*(*Agent*, *A*), *At*(*Gold*, *A*), *At*(*Depot*, *C*),*Adjacent*(*A*,*B*)}
D = {⟨+*At*(*Gold*, *A*); [ ]⟩}
I = { }
σ = `Select`

B = {*At*(*Agent*, *A*), *At*(*Gold*, *A*), *At*(*Depot*, *C*),*Adjacent*(*A*,*B*)}
D = { }
I = {[*Pick*(*Gold*), !*At*(*Agent*, *C*), *Drop*(*Gold*)]}
σ = `Act`

B = {*At*(*Agent*, *A*), *Carries*(*Agent*, *Gold*), *At*(*Depot*, *C*),*Adjacent*(*A*,*B*)}
D = { }
I = {[!*At*(*Agent*, *C*), *Drop*(*Gold*)]}
σ = `Sense`

Example (cont'd)

B = {*At*(*Agent*, *A*), *Carries*(*Agent*, *Gold*), *At*(*Depot*, *C*),*Adjacent*(*A*,*B*)}
D = { }
I = {[!*At*(*Agent*, *C*), *Drop*(*Gold*)]}
σ = `Select`

B = {*At*(*Agent*, *A*), *Carries*(*Agent*, *Gold*), *At*(*Depot*, *C*),*Adjacent*(*A*,*B*)}
D = { }
I = {[!*At*(*Agent*, *C*), *Drop*(*Gold*)]}
σ = `Act`

B = {*At*(*Agent*, *A*), *Carries*(*Agent*, *Gold*), *At*(*Depot*, *C*),*Adjacent*(*A*,*B*)}
D = {⟨+!*At*(*Agent*, *C*); [*Drop*(*Gold*)]⟩}
I = { }
σ = `Sense`

Example (cont'd)

B = {*At*(*Agent*, *A*), *Carries*(*Agent*, *Gold*), *At*(*Depot*, *C*),*Adjacent*(*A*,*B*)}
D = {⟨+!*At*(*Agent*, *C*); [*Drop*(*Gold*)]⟩}
I = { }
σ = `Select`

B = {*At*(*Agent*, *A*), *Carries*(*Agent*, *Gold*), *At*(*Depot*, *C*),*Adjacent*(*A*,*B*)}
D = { }
I = {[*Move*(*A, B*),!*At*(*Agent*, *C*); *Drop*(*Gold*)]}
σ = `Act`

B = {*At*(*Agent*, *B*), *Carries*(*Agent*, *Gold*), *At*(*Depot*, *C*),*Adjacent*(*A*,*B*)}
D = { }
I = {[!*At*(*Agent*, *C*); *Drop*(*Gold*)]}
σ = `Sense`

. . .

Towards An ALP Interpreter for AgentSpeak

- ALP clauses describe the sense-select-act cycle
- Beliefs are part of the background theory of the agent
- Special fluent *Goals(e)* to describe the new external desires
- Special action *Sense_Act* to describe sensing
- Intention encoded as list of lists (= list of procedure bodies)
- Desire encoded as pair (*event*,*intention*)
- Procedures encoded as clauses defining `procedure(e,p)`

An ALP Interpreter for AgentSpeak (1/2)

```
sense(D,I)   :- do(sense_act),
                ?(goals(E)),
                append(D,E,F),
                select(F,I).

select(D,I) :- select((X,J),D,E),
               procedure(X,P),
               act(E,[[P|J]|I]).

select(D,I) :- act(D,I).


append(D,[],D).

append(D,[G|H],[(G,[])|E]) :- append(D,H,E).

select(X,[X|Xs],Xs).

select(X,[Y|Xs],[Y|Ys]) :- select(X,Xs,Ys).
```

An ALP Interpreter for AgentSpeak (2/2)

```
act(D,[]) :- sense(D,[]).

act(D,I)  :- select([],    I,J), act(D,J).

act(D,I)  :- select([[]|P],I,J), act(D,[P|J]).


act(D,I)  :- select([[   A|P]|Q],I,J), do(A),
                                       sense(D,[[P|Q]|J]).

act(D,I)  :- select([[?(F)|P]|Q],I,J), ?(F),
                                       sense(D,[[P|Q]|J]).

act(D,I)  :- select([[?(F)|P]|Q],I,J),
                              sense([(+(?(F)),[P|Q])|D],J).

act(D,I)  :- select([[!(F)|P]|Q],I,J),
                              sense([(+(!(F)),[P|Q])|D],J).
```

Encoding Procedures

Procedures $d : b_1, ..., b_m \leftarrow p_1, ..., p_n$ encoded as

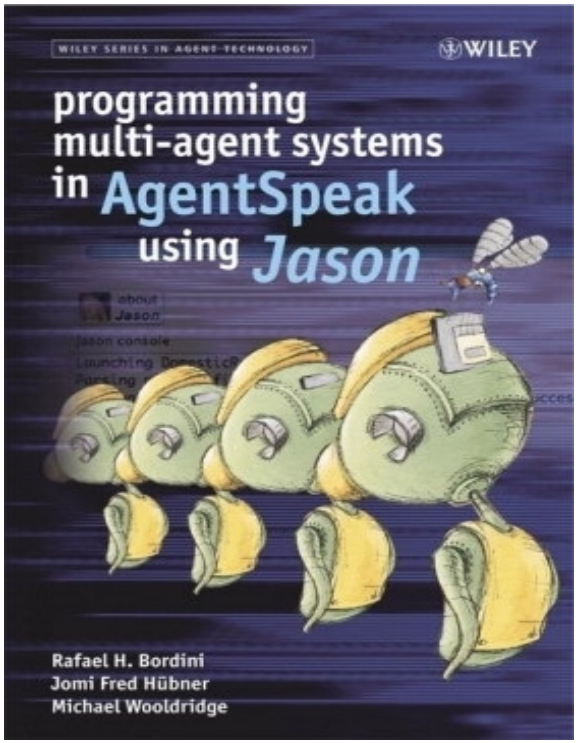
```
procedure(d,P) :- ?(b1), ..., ?(bm),
                  P = [p1,...,pn].
```

```
procedure(+(at(gold,X)),P) :-
     ?(at(agent,X)), ?(at(depot,Y)),
     P = [do(pick(gold)), !(at(agent,Y)), drop(gold)].

procedure(+(!(at(agent,X))),P) :-
     ?(at(agent,X)), P=[].

procedure(+(!(at(agent,X))),P) :-
     ?(at(agent,Y)), ?(not X=Y), ?(adjacent(Y,Z)),
     ?(not at(obstacle,Z)),
     P = [move(Y,Z), !(at(agent,X))].
```

Programming Multi-Agent Systems in Jason

Jason

- Jason implements the operational semantics of a variant of AgentSpeak
- Various extensions, aimed at a more practical programming language
- Platform for developing multi-agent systems
- Available Open Source at `http://jason.sf.net`



Syntax: Beliefs and Goals

- Beliefs represent the information available to an agent (about the environment, other agents, ...)

```
publisher(wiley)
```

- Goals represent (a) states of affairs the agent wants to bring about (i.e. come to believe)

```
!write(book)
```

or (b) attempts to retrieve information from the belief base

```
?publisher(P)
```

Syntax: Plans

An agent reacts to events by executing plans.

- Events happen as a consequence to changes in the agent's beliefs or goals
- Plans are recipes for action, representing the agent's know-how

```
triggering_event : context <- body.
```

where
- the triggering event denotes the event that the plan is meant to handle
- the context represents the circumstances in which the plan can be used
- the body is the course of action to be used at the time a plan is chosen

Syntax: Events and Plans (cont'd)

- Triggering events:

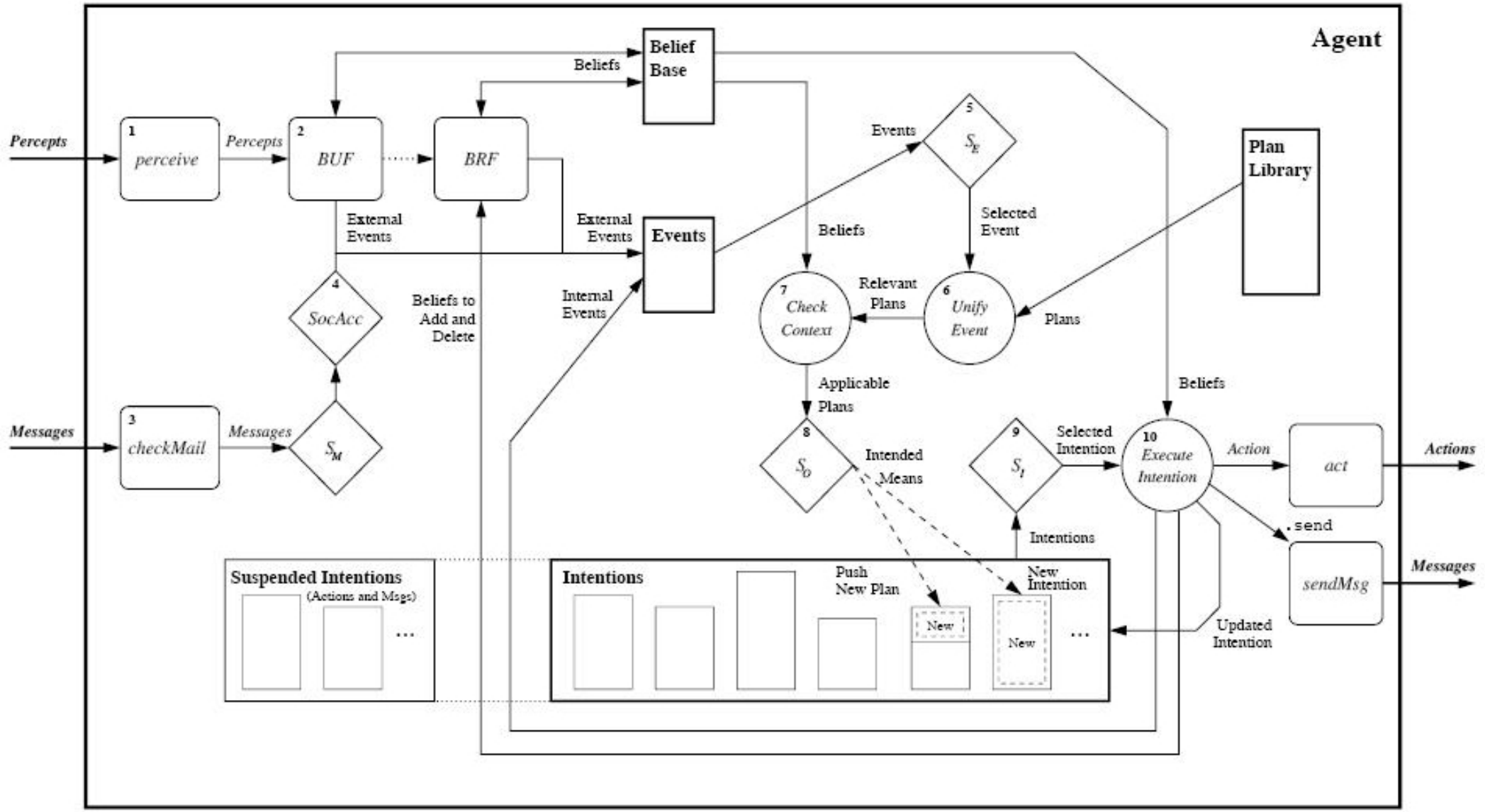
  `+b` (belief addition)
  `-b` (belief deletion)
  `+!g` (achievement-goal addition)
  `-!g` (achievement-goal deletion)
  `+?g` (test-goal addition)
  `-?g` (test-goal deletion)

- The context of a plan is a logical expression to be checked whether it follows from the current belief base
- The body of a plan is a sequence of formulas separated by `;`

Example

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

+!at(Coords)
   :  not at(Coords) & safe_path(Coords)
   <- move_towards(Coords);
      !at(Coords.

+!at(Coords) ...
```

Jason Reasoning Cycle



Jason Reasoning Cycle

1. Perceiving the environment
2. Updating the belief base
3. Receiving communication from other agents
4. Selecting “socially acceptable“ messages
5. Selecting an event
6. Retrieving all relevant plans
7. Determining the applicable plans
8. Selecting one applicable plan
9. Selecting an intention for further execution
10. Executing one step of an intention

Intention Execution

1. Environment actions
2. Achievement goals
3. Test goals
4. Mental notes
5. Internal actions
6. Expressions

Belief Annotations

- Annotated predicate

  `ps(t1,...,tn)[a1,...,am]`

  where `ai` are first-order terms

- All predicates in the belief base have a special annotation `source(si)` where `si` $\in$ {`self`, `percept`} $\cup$ `AgId`

Examples of Annotations

- An agent's belief base with a user-defined `doc` annotation (degree of certainty)

```
blue(box1)[source(ag1)].

red(box1)[source(percept)].

colorblind(ag1)[source(self),doc(0.7)].

liar(ag1)[source(self),doc(0.2)].
```

Plan Annotations

- Plan labels also can have annotations (eg, to specify meta-level information)
- Selection functions (Java) can use such information in plan/intention selection
- Possible to change those annotations dynamically (eg, to update priorities)

- Annotations go in the plan label

```
@aPlan[chance_of_success(0.3),
       usual_payoff(0.9),
       any_other_property]
 +!g(X)
   :  c(t)
   <- a(X).
```

Strong Negation

- The operator ~ is used for strong negation

```
+!leave(home)
   :  not raining & not ~raining
   <- open(curtains) ;
      ...

+!leave(home)
   :  not raining & not ~raining
   <- .send(someAgent,askOne,raining) ;
      ...
```

Belief-Base Rules

- Prolog-like rules in the belief base

```
+likely_color(Obj,C) :- color(Obj,C)[doc(D1)]
                        & not ( color(Obj,_)[doc(D2)]
                                & D2 > D1 )
                        & not ~color(C,B).
```

Handling Plan Failure

- Goal-deletion events syntactically defined in AgentSpeak, but no semantics
- Jason uses them for a plan failure handling mechanism. Handling plan failures is very important as reactive agents are situated in dynamics environments
- A form of “contingency plan“, possibly to “clean up“ before attempting another plan

Example: agent that is blindly committed to goal `g`

```
+!g : g    <- true.
+!g : ...  <- ... ; ?g.
...
-!g : true <- !g.
```

Internal Actions

- Unlike normal actions, internal actions do not change the environment
- Code to be executed as part of the agent reasoning cycle
- AgentSpeak is meant as a high-level language for the agent's practical reasoning
- Internal actions can be used for invoking legacy code elegantly

Internal Actions (cont'd)

- Libraries of user-defined internal actions

  ```
  lib_name.action_name(...)
  ```

- Predefined internal actions have an empty library name

- Internal action for communications

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

  where `ilf` ∈ {`tell,untell,unachieve,askOne,`
  `askAll,askHow,tellHow,untellHow`}

Internal Actions (cont'd)

- Examples of BDI-related internal actions

```
.desire(literal)
.intend(literal)
.drop_desires(literal)
.drop_intentions(literal)
```

- Many others available for: printing, sorting, list/string operations, manipulating the beliefs/annotations/plan library, creating agents, waiting/generating events, etc.

MAS Definition

- Simple way of defining a multi-agent system

```
MAS my_system {
    infrastructure: Jade
    environment: MyEnv
    ExecutionControl: ...
    agents: ag1; ag2; ag3;
}
```

- Infrastructure options: `Centralised, Saci, Jade`
- Easy to define the host where agents and the environment will run
- Multiple instances of an agent `myagents: ag1 #10;`
- Interpreter configuration `agents: ag1 [conf=option];`
- Configuration of event handling, frequency of perception, system messages, user-defined settings, etc.

Customizing Your Agent

Users can customize the `Agent` class to define the selection functions, social relations for communication, and belief update and revision.

- `selectMessage()`
- `selectEvent()`
- `selectOption()`
- `selectIntention()`
- `socAcc()`
- `buf()`
- `brf()`

Overall Agent Architecture

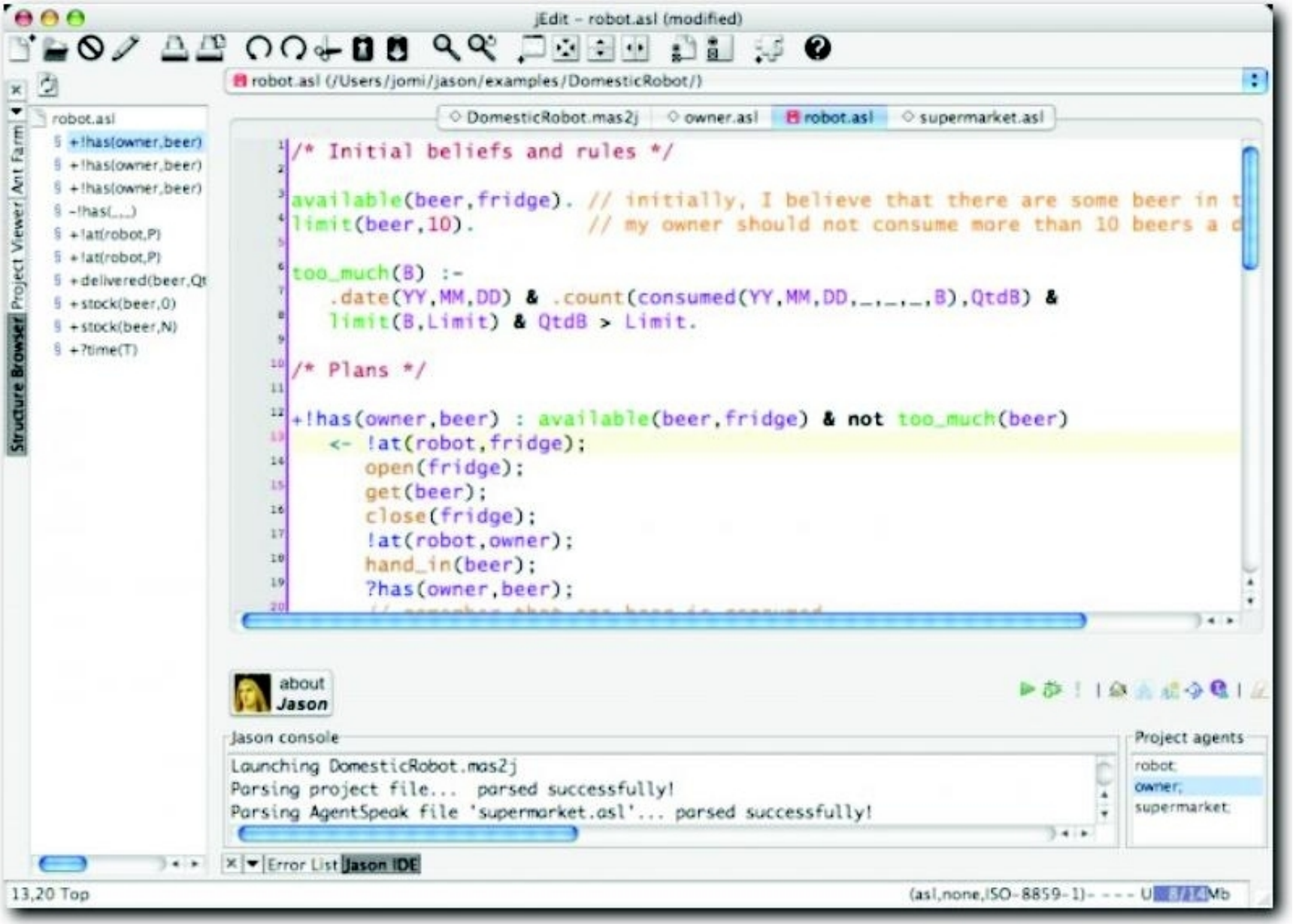
Users can customize the `AgentArch` class to change the way the agent interacts with the infrastructure: perception, action, and communication

- `perceive()`
- `act()`
- `sendMsg()`
- `broadcast()`
- `checkMail()`

Environments

- In actual deployment, there is an environment where the agents are situated
- Need to customize the architecture to get perceptions and actions
- MAS applications are usually tested with simulated environments. This is done in Java by extending Jason's `Environment` class and using methods such as `addPercept(String Agent, Literal Percept)`

Jason for jEdit



Jason's Mind Inspector