Multi-agent and Semantic Web Systems: Agent Reasoning

Fiona McNeill

School of Informatics

18th March 2013

BDI Model

- Dominant model for defining practical agent-based reasoning.
- Addresses question of how to reason about complex distributed systems.
- Behaviour is determined by three elements of mental states:

  Beliefs: These define the partial knowledge that the agent has about the world.

  Desires: These represent the states of affairs that the agent would ideally like to bring about.

  Intentions: The desires that agent has committed to achieving.

- Agent may not be able to achieve all its desires; and they may be inconsistent.
- Intentions $\subseteq$ Desires

Practical Reasoning, I

- Practical reasoning: directed towards deciding what to do.
- Bratman (1990):
  - evaluate competing options;
  - trade-offs between different desires / goals;
  - conditioned by beliefs.
- Foundation for Belief-Desire-Intention (BDI) model of agents.

Practical Reasoning, 2

Deliberation: What to do

- selecting goals, weighing up different 'desires'
- generates intentions

Means-End Reasoning: How to achieve goals

- assess suitable actions, consider available resources
- generates plans, which then turn into action

Intentions, I

Properties of Intentions:

• Once an intention has been adopted, agent will try to carry it out.

• Once an intention has been adopted, agent will persist with it until (i) fulfilled or (ii) considered infeasible.

• Current intentions can exclude otherwise available options / intentions.

•An agent should only adopt an intention if it believes it is achievable.

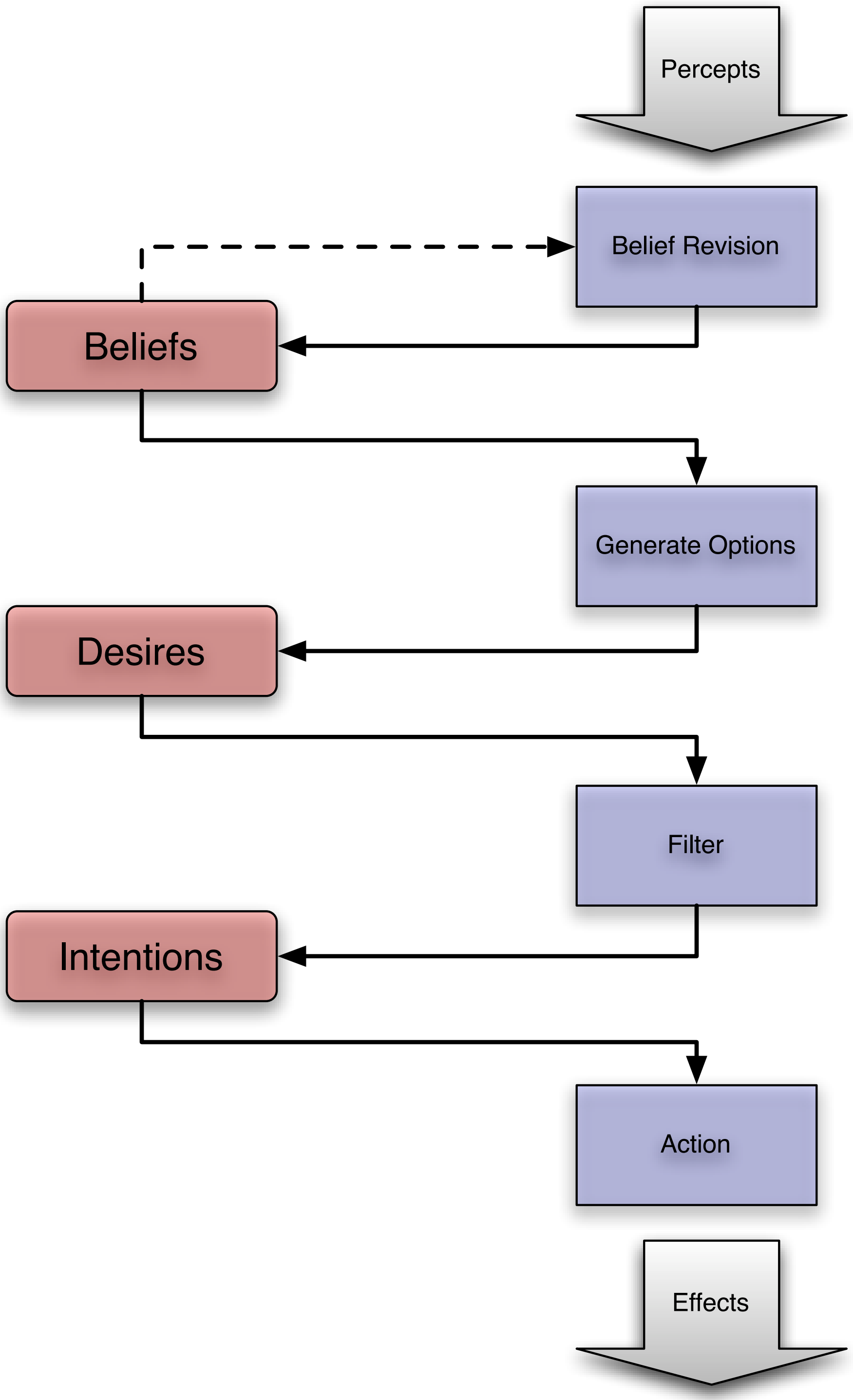
Intentions, 2

Persistent Goal: ϕ is a *persistent goal* if:

- A believes ϕ is not true now, and has a goal that ϕ becomes true in the future; and
- before dropping ϕ, A believes either that ϕ is true or will never become true.

Intention: A has *intention* to carry out action α iff A has persistent goal to bring about a state where it believes that it will do α and then does α.

Simplified BDI Architecture



Simplified BDI Algorithm

```
Reason(B, D, I)
    while true do
        p <- next percept
        B <- revise(B, p)
        D <- options(B, I)
        I <- deliberate(B, D, I)
        P <- plan(B, I)
        execute(P)
```

perceive external events

revise beliefs based on percepts

compute new desires based on revised beliefs

consider competing options and make new intentions

perform means-end analysis on intentions to determine next actions

carry out the action

AgentSpeak

- Originally proposed by Rao
- Programming language for BDI agents
- Based on logic programming (e.g., Prolog)
- Inspired by PRS (Georgeff & Lansky), dMARS (Kinny), and BDI Logics (Rao & Georgeff)
- Abstract programming language, intended to bridge between BDI theory and practical systems like PRS

Syntax of AgentSpeak

The main language constructs of AgentSpeak are:

- Beliefs
- Goals
- Plans

Architecture of an AgentSpeak agent has four main components:

- Belief Base
- Plan Library
- Set of Events
- Set of Intentions

Beliefs and Goals

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

Belief

```
hotel(sheraton)
```

Goals represent states of affairs the agent wants to bring about (or come to believe, when goals are used declaratively)

Achievement goals

```
!book_rooms(sheraton)
```

Or attempts to retrieve information from the belief base:

Test goals

```
?hotel(P)
```

Events and 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

AgentSpeak Plan

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

- `triggering_event` denotes the events that the plan is meant to handle;
- the `context` represent the circumstances in which the plan can be used;
- if the context is believed true at the time a plan is being chosen, then:
  - the `body` is the course of action to be used to handle the event

AgentSpeak 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` is logical expression
  - typically a conjunction of literals;
  - need to check whether they follow from the current state of the belief base
- The `body` is a sequence of actions and (sub) goals to be achieved.

AgentSpeak: Hello World

Hello World

```
started.

+started <- .print("Hello World!").
```

AgentSpeak Plans, I

Mars Rover

```
+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) ...
```

AgentSpeak Plans, 2

Mars Rover

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

- The belief that `Rock` has a green patch has been added (e.g. through perception)
- Whenever agent has this belief, and its batteries are not too low, then:
  - check belief base for coordinates of `Rock` (i.e. a test-goal);
  - achieve goal of reaching those coordinates and examining `Rock`.

AgentSpeak Plans, 3

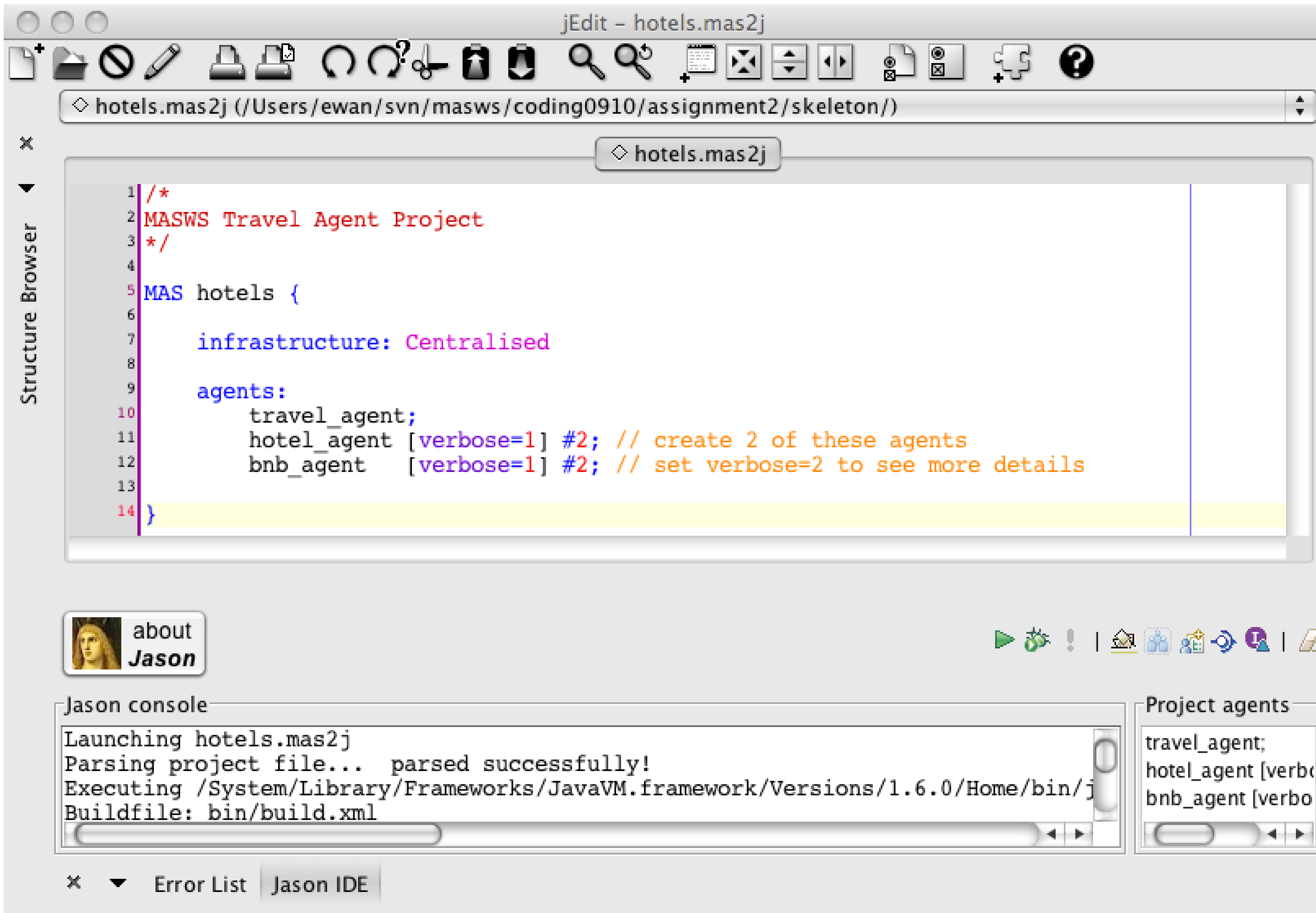
Mars Rover

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

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

- Two alternative courses of action for achieving the goal of reaching the coordinates.
- Choice of action depends on what agent believes to be true of the environment.
- `move_towards(Coords)` is a basic action for changing the environment.
- Alternative plan should deal with situation in which `safe_path(Coords)` fails to be true.

Jason Configuration File



Communication in *Jason*

- At start of each reasoning cycle, agents check for messages from other agents.
- These have following structure: ⟨sender, illoc_force, prop_content⟩
- Messages are sent using a pre-defined internal action: `.send`
- Internal actions are ones which do not affect environment; by convention, names always start with `.` (full-stop).
- General form:

```
.send(receiver, illoc_force, prop_content)
```

Communication in *Jason*: `receiver`

- Uses name for agents given in configuration file.
- If multiple instances (cf. `hotel_agent`), numbers starting from 1 are appended; e.g. `hotel_agent1`, `hotel_agent2`, ...
- `receiver` can also be a list of agent names, for multicasting.
- Alternatively, use the iaction `.broadcast`, which sends to all agents.

Communication in *Jason*: `illoc_force` and `prop_content`

- Uses KQML performatives.
- Two of 10 available performatives:
  - tell — *s* intends *r* to believe the literal in the message's content
  - achieve — *s* requests *r* to try to achieve state of affairs where literal in the message's content is true (goal delegation)
- Propositional content is a term that can e.g. be a literal or represent a triggering event or a plan, or else a list of events, plans, etc.

Travel example

```
+!find_rooms(1) : true
        <- .broadcast(tell, require_rooms(1));
           !wait;
           !show_result.
```

Communication Example

Travel example

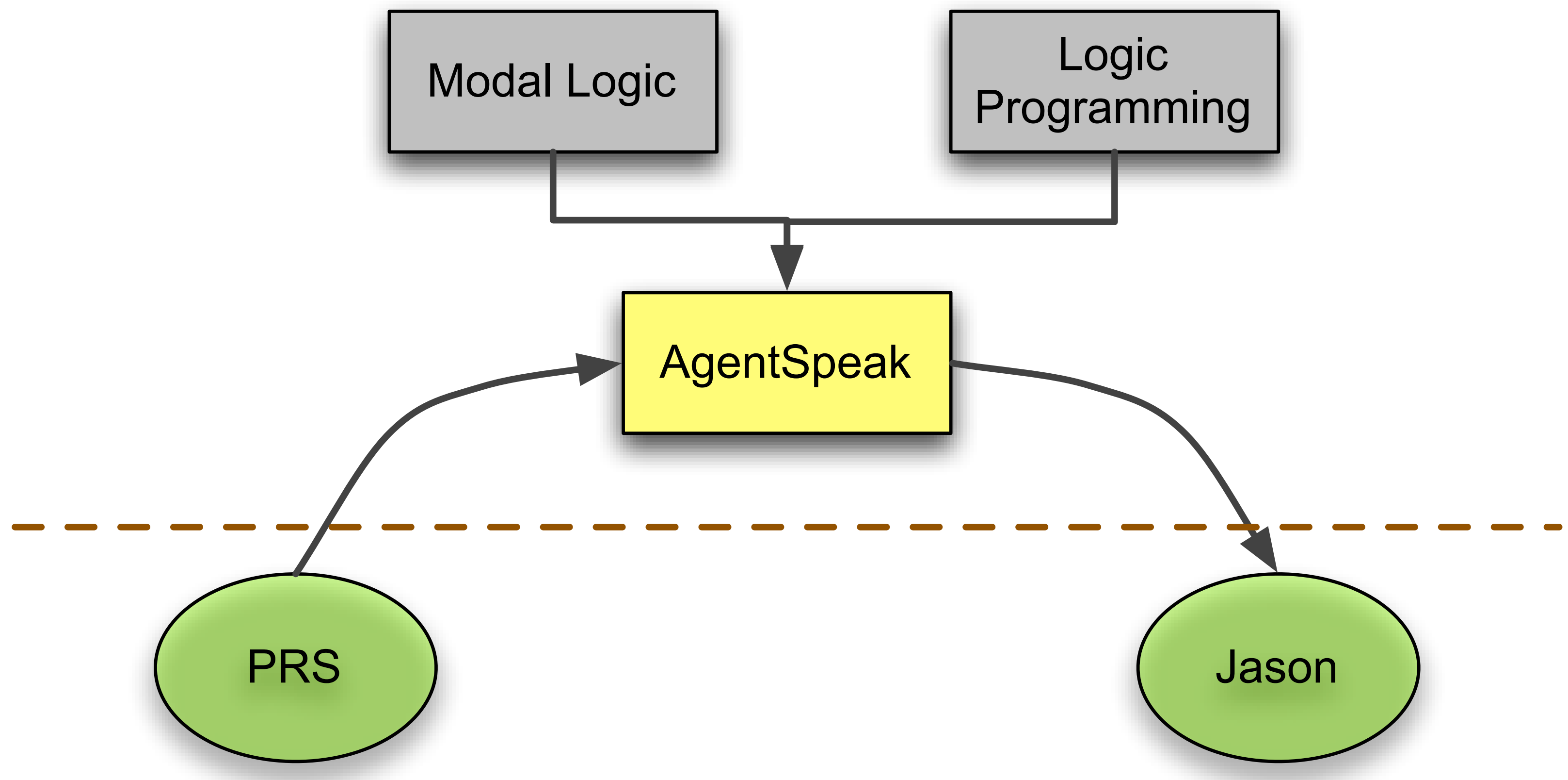
```
+!find_rooms(1) : true
        <- .broadcast(tell, require_rooms(1));
           !wait;
           !show_result.
```

- `hotel_agent` will receive `<travel_agent, tell, require_rooms(1)>`
- the belief `require_rooms(1 [source(travel_agent)]` will be added to belief base of `hotel_agent`.

Hotel agent response

```
+require_rooms(1)[source(Travel)] : ...
        <-      iactions.checkDB(...);
                .send(Travel, tell, reply(...)).
```

Where *Jason* Fits In



Summary

- BDI: psychologically oriented model.
- Claim: people use 'folk psychology' to help understand and reason about complex systems.
- Jason couples BDI with notion of reactive system; also includes some normative / social aspects.
- Can be used to develop models of 'intelligent' decision-making in SemWeb applications.
- Message-exchange built on top of internal actions, beliefs and planning, using KQML performatives.