BDI agents

Autonomous Software Agents

A.A. 2022-2023

Prof. Paolo Giorgini

Dr. Marco Robol

UNIVERSITY OF TRENTO - Italy

Department of Information
and Communication Technology

Practical Reasoning

- Practical reasoning is reasoning directed towards actions — the process of figuring out what to do:

  "Practical reasoning is a matter of weighing conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what the agent believes." (Bratman 1990)

- Practical reasoning is distinguished from *theoretical reasoning* – theoretical reasoning is directed towards beliefs

  Theoretical Reasoning

  If I believe that all men are mortal, and If I believe that Socrates is a man, then I will usually conclude that Socrates is mortal

  Practical reasoning

  Deciding to catch a bus instead of a train

Practical Reasoning

- Human practical reasoning consists of two activities:
  - deliberation
    deciding what state of affair we want to achieve
  - means-ends reasoning
    deciding how to achieve these states of affairs
- Once the goal has been chosen, employ knowledge and reasoning to find out how to reach it
  - The outputs of deliberation are intentions

Intentions in Practical Reasoning

- Intentions pose problems for agents, who need to determine ways of achieving them - *If I have an intention to ϕ, you expect me to devote resources to deciding how to bring about ϕ*

- Intentions provide a "filter" for adopting other intentions, which must not conflict - *If I have an intention to ϕ, you will not expect me to adopt an intention ψ such that ϕ and ψ are mutually exclusive*

- Agents track the success of their intentions, and are inclined to try again if their attempts fail - *If an agent's first attempt to achieve ϕ fails, then all other things being equal, it will try an alternative plan to achieve ϕ*

Intentions in Practical Reasoning

- Agents believe their intentions are possible - *That is, they believe there is at least some way that the intentions could be brought about*
- Agents do not believe they will not bring about their intentions - *It would not be rational of me to adopt an intention to ϕ if I believed ϕ was not possible*
- Under certain circumstances, agents believe they will bring about their intentions - *It would not normally be rational of me to believe that I* would *bring my intentions about; intentions can* fail*. Moreover, it does not make sense that if I believe ϕ is inevitable that I would adopt it as an intention*

Intentions in Practical Reasoning

- Agents need not intend all the expected side effects of their intentions.
  If I believe $\phi \rightarrow \psi$ and I intend that ϕ, I do not necessarily intend ψ also.
  (Intentions are not closed under implication.)

- This last problem is known as the side effect or package deal problem. I may believe that going to the dentist involves pain, and I may also intend to go to the dentist — but this does not imply that I intend to suffer pain!

Intentions in Practical Reasoning

- Notice that intentions are much stronger than mere desires:

  "*My desire to play basketball this afternoon is merely a potential influencer of my conduct this afternoon. It must vie with my other relevant desires [. . .] before it is settled what I will do. In contrast, once I intend to play basketball this afternoon, the matter is settled: I normally need not continue to weigh the pros and cons. When the afternoon arrives, I will normally just proceed to execute my intentions*"
  (Bratman, 1990)

Complications

- Practical reasoning has to be implemented as a computational process.

→ Resource bounds:

  - available memory;
  - time constraints.

- Implications:
  - computation is an important resource, that impacts on performance;
  - agents cannot deliberate indefinitely: they have to commit to one state of affairs, which may not be optimal

Intentions

Definition: Intentions are the states of affairs that an agents has chosen and committed to

- Intentions as *pro-attitudes:* they tend to lead to actions
- Intentions *persist*
- Intentions *constrain* practical reasoning
- Related to beliefs about the future.

- Therefore, intentions *interact* with agent's beliefs and other mental states

Representation

- Agents keep an explicit representation of their beliefs, desires and intentions
- No assumption on how such information is represented
- Notation
  - B agent's current beliefs, *Bel* set of all such beliefs (B ⊆ *Bel*)
  - D agent's current desires, *Des* set of all such desires (D ⊆ *Des*)
  - I agent's current intentions, *Int* set of all such intentions (I ⊆ *Int*)

Deliberation

- Option generation: an agent generates a set of options among the possible desires:
  - An agent's option generation function maps the agent's current beliefs and intentions to the agent's new desires:

$$\text{Option: Bel x Int} \rightarrow \text{Des}$$

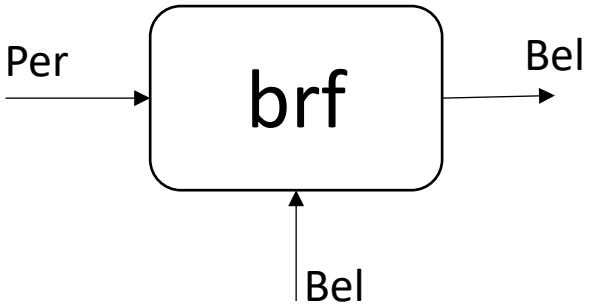
- Option filtering: an agent chooses among the possible options, committing to the best:
  - An agent's filtering function maps the agent's current beliefs, desires and intentions to a set of intentions

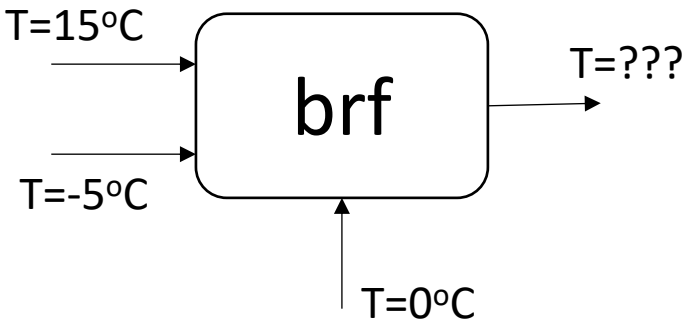
$$\text{Filter: Bel x Des x Int} \rightarrow \text{Int}$$

Belief Revision

- An agents updates its current beliefs when it perceives new information from the environment
  - An agent's belief revision function maps the agent's current beliefs and the current percept to the agent's new beliefs

`brf: Bel x Per → Bel`





Means-end reasoning

- The process of deciding how to achieve an *end* using the available *means*
- Known in AI as *planning*

Input:

- A *goal, intention* or *task:* a state of affair that the agent wants to achieve or maintain
- The current *state of the environment:* the agent's beliefs
- A set of *actions* available to the agent.

Output:

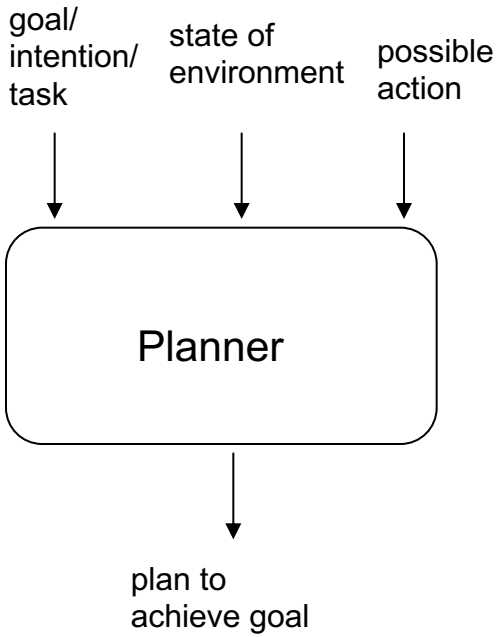
- A sequence of actions, supposed to produce the desired state of affairs.

What is Means-End Reasoning?

- Basic idea is to give an agent:
  - representation of goal/intention to achieve
  - representation actions it can perform
  - representation of the environment

and have it generate a plan to achieve the goal

- *automatic programming*



Implementing Practical Reasoning Agents

- A first pass at an implementation of a practical reasoning agent:

```
Agent Control Loop Version 1
1. while true
2.     observe the world;
3.     update internal world model;
4.     deliberate about what intention to achieve next;
5.     use means-ends reasoning to get a plan for the intention;
6.     execute the plan
7. end while
```

Implementing Practical Reasoning Agents

- Let's make the algorithm more formal:

```
Agent Control Loop Version 2
1.  B := B0; /* initial beliefs */
2.  while true do
3.      get next percept ρ;
4.      B := brf(B, ρ);
5.      I := deliberate(B);
6.      π := plan(B, I);
7.      execute(π)
8.  end while
```

The environment does not change

Deliberation

- How does an agent deliberate?
  - begin by trying to understand what the *options* available to you are
  - choose between them, and commit to some
- Two distinct functional components:
  - option generation
    in which the agent generates a set of possible alternatives;
    Represent option generation via a function, *options*, which takes the agent's current beliefs and current intentions, and from them determines a set of options (= *desires*)
  - filtering
    in which the agent chooses between alternatives and commits to achieving them.
    In order to select between competing options, an agent uses a filter function.

Deliberation

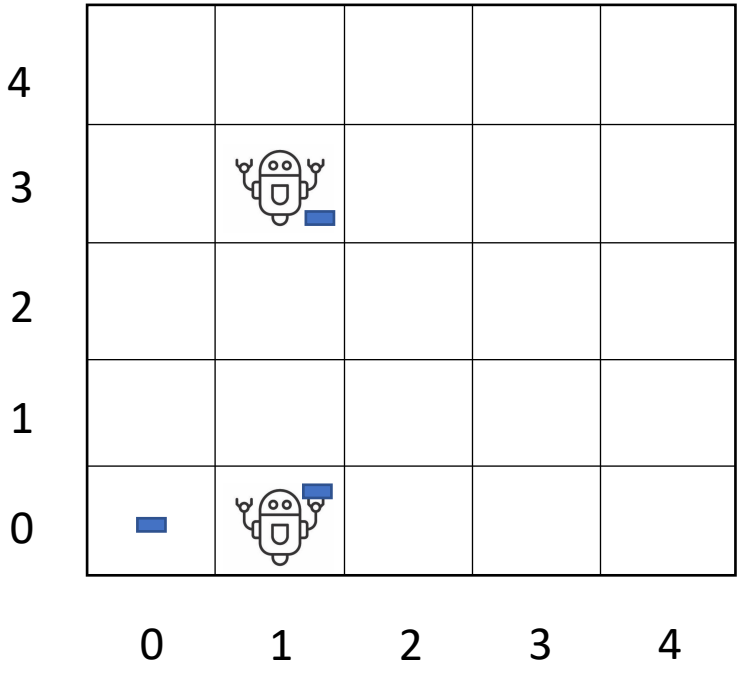
In an ideal world,
an agent would like
all its desires achieved

```
Agent Control Loop Version 3
1.
2.    B := B_0;
3.    I := I_0;
4.    while true do
5.        get next percept ρ;
6.        B := brf(B, ρ);
7.        D := options(B, I);
8.        I := filter(B, D, I);
9.        π := plan(B, I);
10.       execute(π)
11.   end while
```

Selects the best
option for the
agent to commit to

Commitment Strategies

- Commitment strategies:
  - Blind commitment
    A blindly committed agent will continue to maintain an intention until it believes the intention has actually been achieved. Blind commitment is also sometimes referred to as fanatical commitment.

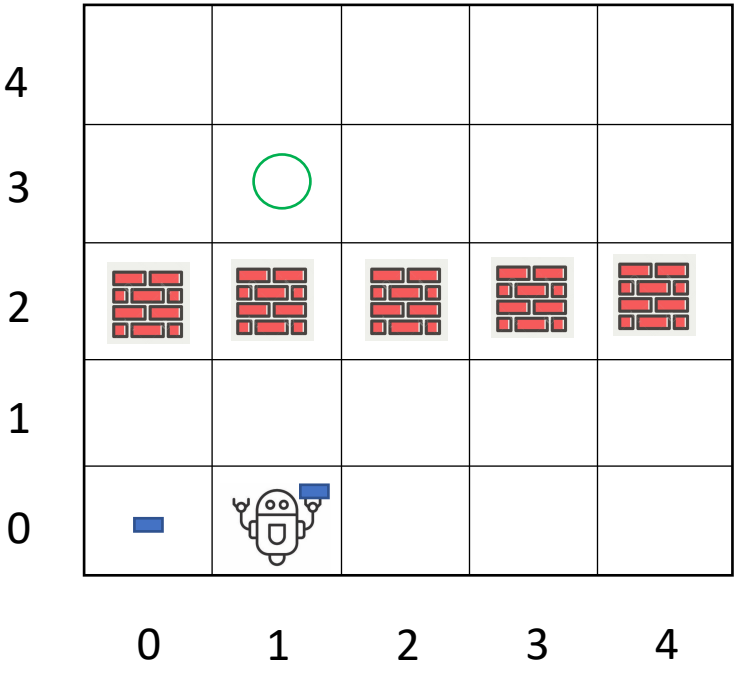



```
-- t=0
B = {In(1,0),carry(pack_1)}
I = {}
P = {}
Do: Null

-- t=1
B = {In(1,0},carry(pack_1)}
I = {In(1,3,pack_1)}
P={move(1,1),move(1,2),move(1,3),put_down(pack_1)}
Do: Null

-- t=2
B = {In(1,0),carry(pack_1),In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,2),move(1,3),put_down(pack_1)}
Do: move(1,1)
```

Blind commitment

```
-- t=3
B = {In(1,1),carry(pack_1),In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,3),put_down(pack_1)}
Do: move(1,2)

-- t=4
B = {In(1,2),carry(pack_1),In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {put_down(pack_1)}
Do: move(1,3)
```

```
-- t=5
B = {In(1,3),carry(pack_1),In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {}
Do: put_down(pack_1)

-- t=6
B = {In(1,3), In(1,3,pack_2), In(0,0,pack_2)}
I = {}
P = {}
Do: Null
```




```
-- t=0
B = {In(1,0),carry(pack_1)}
I = {}
P = {}
Do: Null

-- t=1
B = {In(1,0},carry(pack_1)}
I = {In(1,3,pack_1)}
P={move(1,1),move(1,2),move(1,3),put_down(pack_1)}
Do: Null

-- t=2
B = {In(1,0),carry(pack_1),In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,2),move(1,3),put_down(pack_1)}
Do: move(1,1)
```

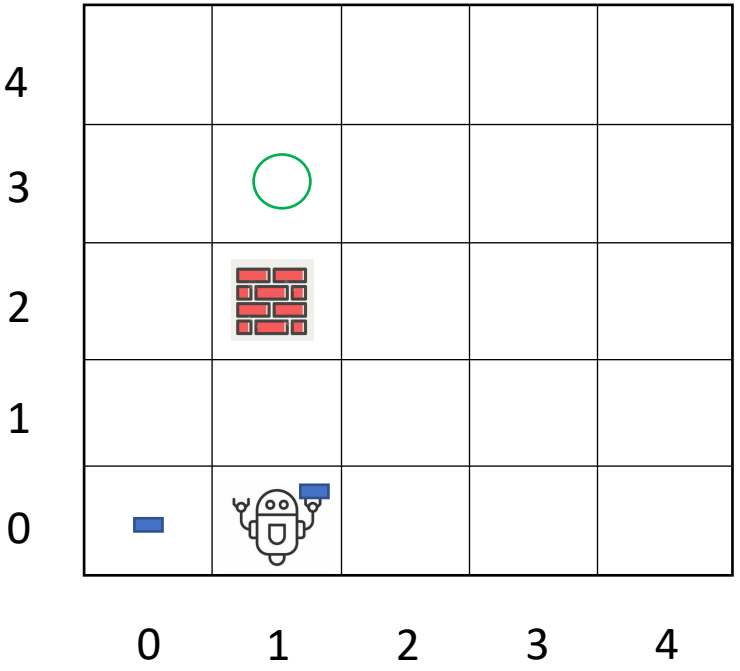
```
-- t=3
B = {In(1,1),carry(pack_1),In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,3),put_down(pack_1)}
Do: move(1,2)

-- t=4
B = {In(1,1),carry(pack_1),In(0,0,pack_2),block(2)}
I = {In(1,3,pack_1)}
P = {move(1,3),put_down(pack_1)}
Do: move(1,2)
```

Blind commitment does not work

Commitment Strategies

- An agent has commitment both to ends (i.e., the wishes to bring about), and means (i.e., the mechanism via which the agent wishes to achieve the state of affairs)
- If a plan goes wrong
  - Replan – try another plan keeping the commitment

```
Agent Control Loop Version 4
1.
2.   B := B_0;
3.   I := I_0;
4.   while true do
5.        get next percept ρ;
6.        B := brf(B, ρ);
7.        D := options(B, I);
8.        I := filter(B, D, I);
9.        π := plan(B, I);
10.       while not empty(π) do
11.            α := hd(π);
12.            execute(α);
13.            π := tail(π);
14.            get next percept ρ;
15.            B := brf(B, ρ);
16.            if not sound(π, I, B) then
17.                 π := plan(B, I)
18.            end-if
19.       end-while
20.  end-while
```




```
-- t=0
B = {In(1,0),carry(pack_1)}
I = {}
P = {}
Do: Null

-- t=1
B = {In(1,0},carry(pack_1)}
I = {In(1,3,pack_1)}
P={move(1,1),move(1,2),move(1,3),put_down(pack_1)}
Do: Null

-- t=2
B = {In(1,0),carry(pack_1),In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,2),move(1,3),put_down(pack_1)}
Do: move(1,1)
```

Replan

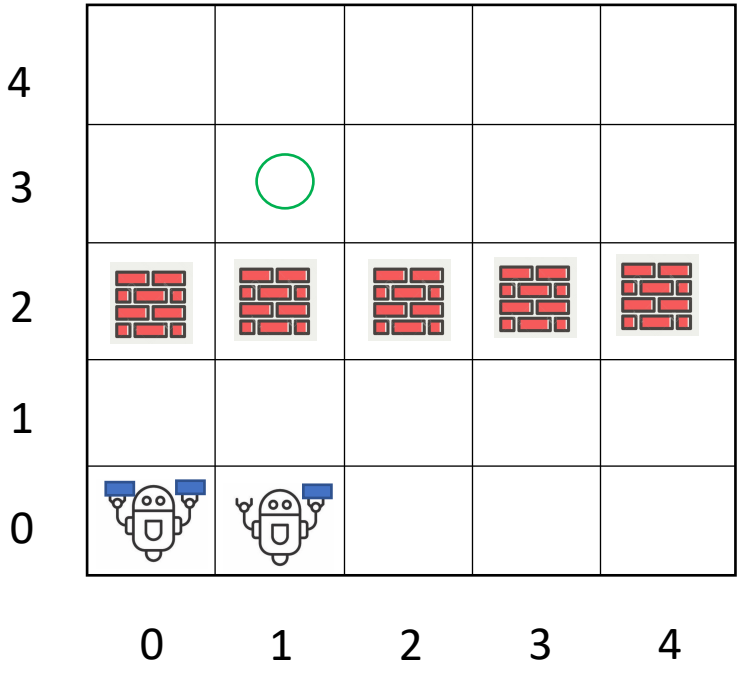
```
-- t=3
B = {In(1,1),carry(pack_1),In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,3),put_down(pack_1)}
Do: move(1,2)

-- t=4
B = {In(1,1),carry(pack_1),In(0,0,pack_2),block(1,2)}
I = {In(1,3,pack_1)}
P = {move(0,1),move(0,2),move(0,3), move(1,3),put_down(pack_1)}
Do: Null
```

Commitment Strategies

- Commitment strategies:
  - Still overcommitted to intentions
    - never stops to consider whether or not its intentions are appropriate
  - Modification: stop to determine whether intentions have succeeded or whether they are impossible

  - Single-minded commitment
    A single-minded agent will continue to maintain an intention until it believes that either the intention has been achieved, or else that it is no longer possible to achieve the intention

```
Agent Control Loop Version 5
2.  B := B0;
3.  I := I0;
4.  while true do
5.      get next percept ρ;
6.      B := brf(B, ρ);
7.      D := options(B, I);
8.      I := filter(B, D, I);
9.      π := plan(B, I);
10.     while not empty(π)
                or succeeded(I, B)
                or impossible(I, B)) do
11.         α := hd(π);
12.         execute(α);
13.         π := tail(π);
14.         get next percept ρ;
15.         B := brf(B, ρ);
16.         if not sound(π, I, B) then
17.             π := plan(B, I)
18.         end-if
19.     end-while
20. end-while
```




```
-- t=0
B = {In(1,0),carry(pack_1)}
I = {}
P = {}
Do: Null

-- t=1
B = {In(1,0},carry(pack_1)}
I = {In(1,3,pack_1)}
P={move(1,1),move(1,2),move(1,3),put_down(pack_1)}
Do: Null

-- t=2
B = {In(1,0),carry(pack_1),In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,2),move(1,3),put_down(pack_1)}
Do: move(1,1)
```

Single-minded commitment

```
-- t=3
B = {In(1,1),carry(pack_1),In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,3),put_down(pack_1)}
Do: move(1,2)

-- t=4
B = {In(1,1),carry(pack_1),In(0,0,pack_2),block(2)}
I = {}
P = {}
Do: Null
```

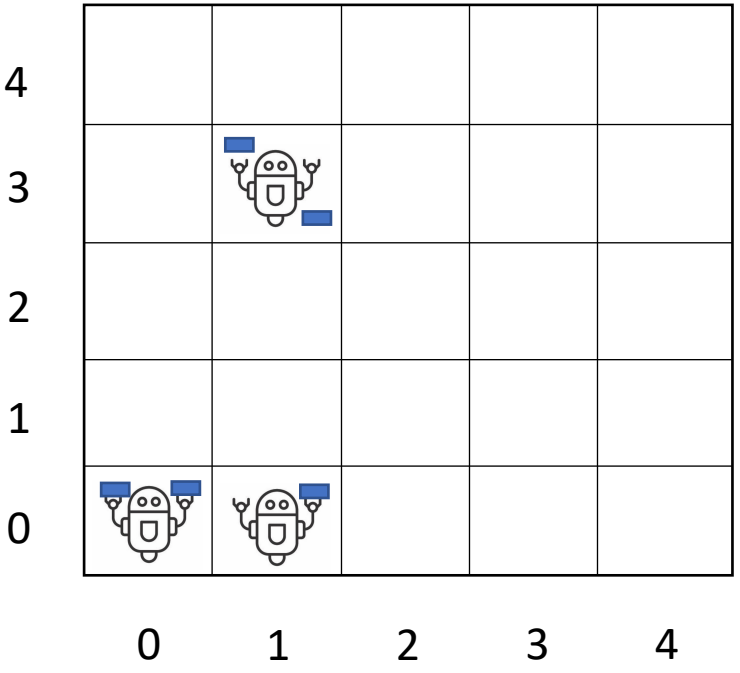
```
-- t=5
B = {In(1,1),carry(pack_1),In(1,0,pack_2)}
I = {carry(pack_2)}
P = {move(0,1), move(0,0),pick_up(pack_2)}}
Do: Null

_
_
_
_
```

Intention Reconsideration

- Our agent gets to reconsider its intentions once every time around the outer control loop, i.e., when:
  - it has completely executed a plan to achieve its current intentions; or
  - it believes it has achieved its current intentions; or
  - it believes its current intentions are no longer possible.
- This is limited in the way that it permits an agent to *reconsider* its intentions
  - In a competitive environment the agent could decide to pick_up(pack_2) before deliver(pack_1)
  - What about if pack_3 appear?

- Reconsider intentions after executing every action

```
Agent Control Loop Version 6
1.
2.   B := B0;
3.   I := I0;
4.   while true do
5.        get next percept ρ;
6.        B := brf(B, ρ);
7.        D := options(B, I);
8.        I := filter(B, D, I);
9.        π := plan(B, I);
10.       while not (empty(π)
                  or succeeded(I, B)
                  or impossible(I, B)) do
11.           α := hd(π);
12.           execute(α);
13.           π := tail(π);
14.           get next percept ρ;
15.           B := brf(B, ρ);
16.           D := options(B, I);
17.           I := filter(B, D, I);
18.           if not sound(π, I, B) then
19.                π := plan(B, I)
20.           end-if
21.       end-while
22.  end-while
```




```
-- t=0
B = {In(1,0),carry(pack_1)}
I = {}
P = {}
Do: Null

-- t=1
B = {In(1,0},carry(pack_1)}
I = {In(1,3,pack_1)}
P={move(1,1),move(1,2),move(1,3),put_down(pack_1)}
Do: Null

-- t=2
B = {In(1,0),carry(pack_1),In(0,0,pack_2)}
I = {In(1,3,pack_1)}
P = {move(1,2),move(1,3),put_down(pack_1)}
Do: move(1,1)
```

Intentions reconsideration

```
-- t=3
B = {In(1,1),carry(pack_1),In(0,0,pack_2)}
I = {carry(pack_2),In(1,3,pack_1)}
P=  {move(1,0),move(0,0),pick_up(pack_2),move(1,1),
     move(1,2),move(1,3),put_down(pack_1)}
Do: Null

-- t=4
B = {In(1,1),carry(pack_1),In(0,0,pack_2)}
I = {carry(pack_2),In(1,3,pack_1)}
P=  {move(0,0),pick_up(pack_2),move(1,1),  move(1,2),
     move(1,3),put_down(pack_1)}
Do: move(1,0)…….
```

Intention Reconsideration

- But intention reconsideration is *costly*!
  A dilemma:
  - an agent that does not stop to reconsider its intentions sufficiently often will continue attempting to achieve its intentions even after it is clear that they cannot be achieved, or that there is no longer any reason for achieving them
  - an agent that constantly reconsiders its intentions may spend insufficient time actually working to achieve them, and hence runs the risk of never actually achieving them
- Solution: incorporate an explicit meta-level control component, that decides whether or not to reconsider

```
Agent Control Loop Version 7
1.
2.   B := B0;
3.   I := I0;
4.   while true do
5.        get next percept ρ;
6.        B := brf(B, ρ);
7.        D := options(B, I);
8.        I := filter(B, D, I);
9.        π := plan(B, I);
10.       while not (empty(π)
                  or succeeded(I, B)
                  or impossible(I, B)) do
11.           α := hd(π);
12.           execute(α);
13.           π := tail(π);
14.           get next percept ρ;
15.           B := brf(B, ρ);
16.           if reconsider(I, B) then
17.                D := options(B, I);
18.                I := filter(B, D, I);
19.           end-if
20.           if not sound(π, I, B) then
21.                π := plan(B, I)
22.           end-if
23.       end-while
24.  end-while
```

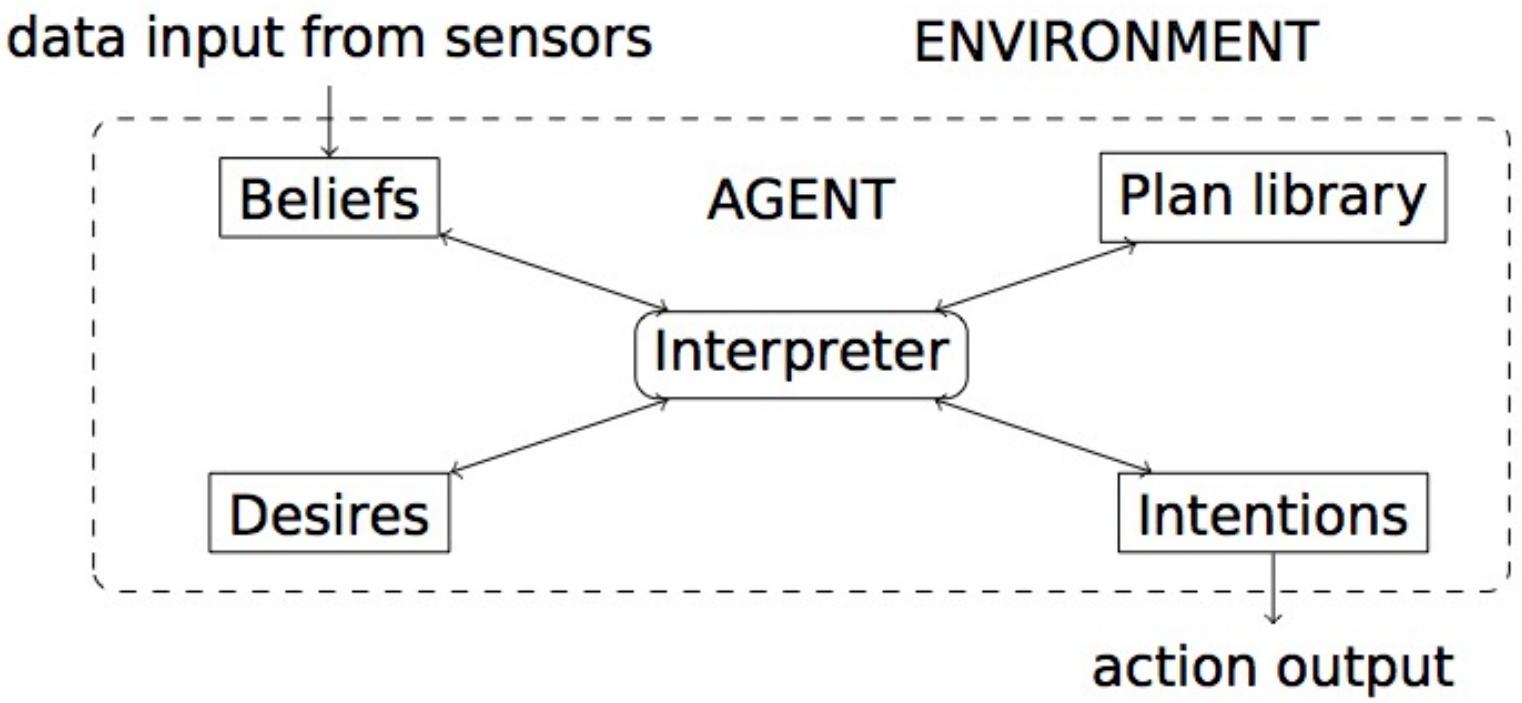
Optimal Intention Reconsideration

- Examples of reconsideration strategy (Kinny and Georgeff):
  - bold agents: never pause to reconsider intentions, and
  - cautious agents: stop to reconsider after every action
- Dynamism in the environment is represented by the rate of world change (g)
- Results (not surprising):
  - If g is low (i.e., the environment does not change quickly), then bold agents do well compared to cautious ones. This is because cautious ones waste time reconsidering their commitments while bold agents are busy working towards — and achieving — their intentions.
  - If g is high (i.e., the environment changes frequently), then cautious agents tend to outperform bold agents. This is because they are able to recognize when intentions are doomed, and also to take advantage of serendipitous situations and new opportunities when they arise.

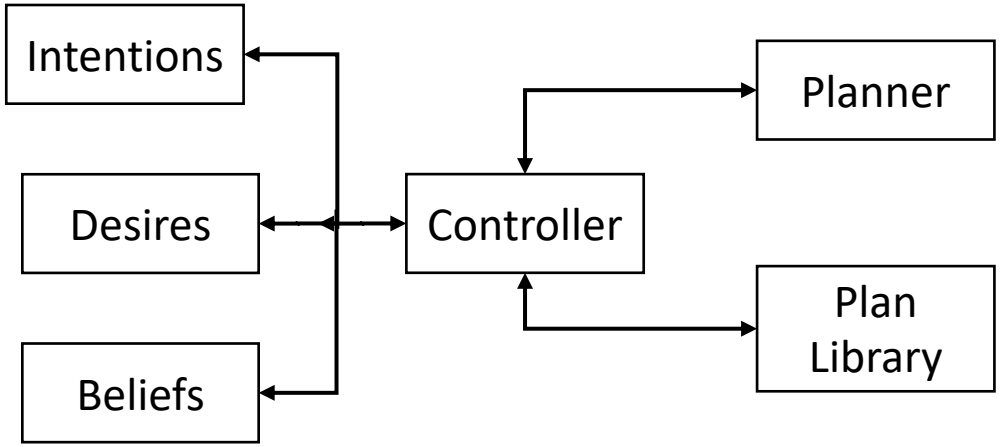
The Procedural Reasoning System

- Perhaps the first and most influential BDI agent architecture (Georgeff, Lansky)
  - each agent is equipped with a plan library, representing that agent's procedural knowledge
  - knowledge about the mechanisms that can be used by the agent in order to realize its intentions
- Peculiar management of plans:
  - plans are not built by the agent, but selected from a hand-written library;
  - plans have a precondition (context) and post-condition (goal);
  - The plan's body may contain not only actions, but also goals
  - The options available to an agent are directly determined by the plans an agent has

PRS architecture



Combining Procedural and Planning

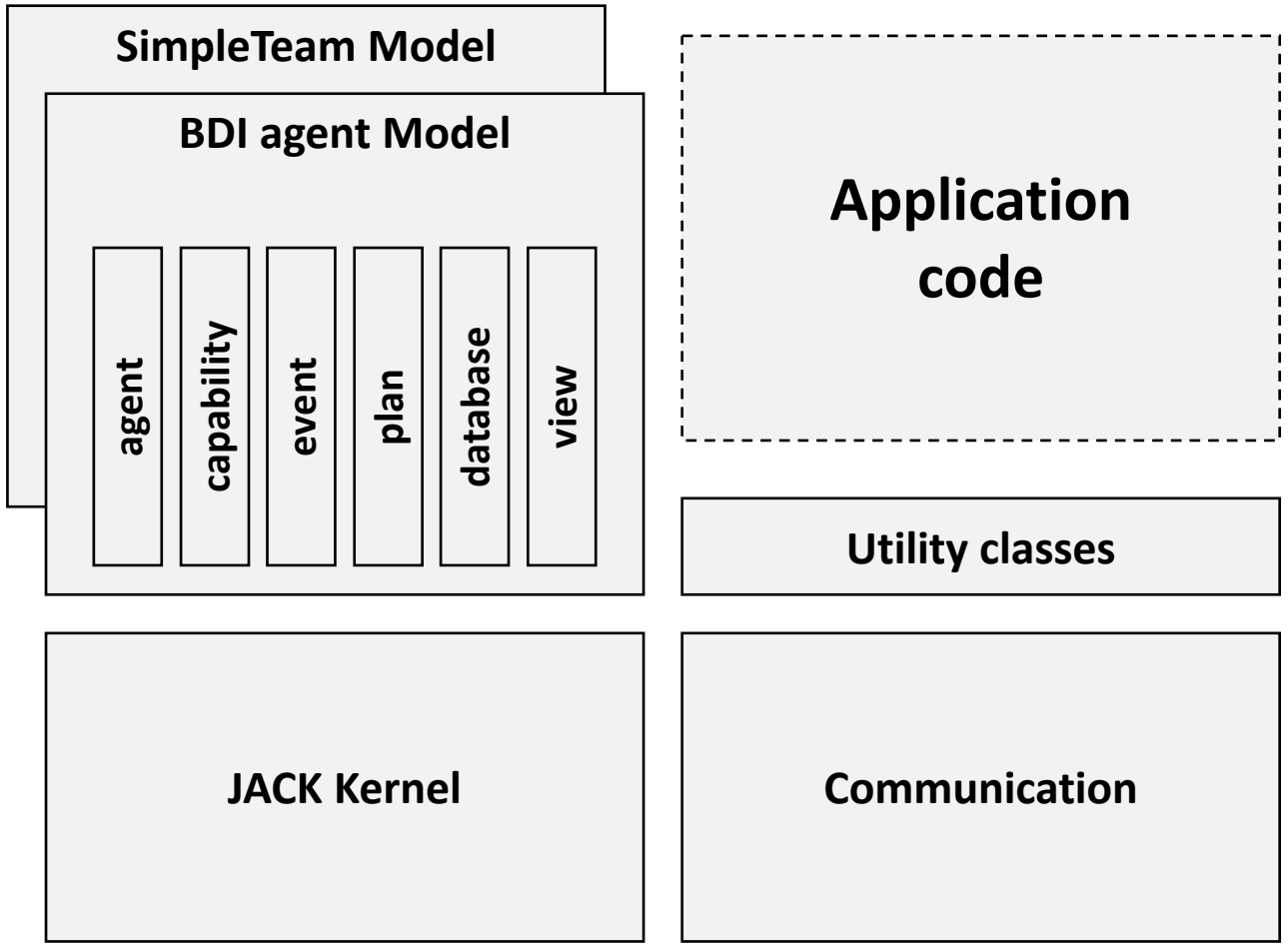


- Plan Library
  - Fast (re)planning -> reactive
  - Limited predefined plans
- Planning
  - No predefined plans
  - Time consuming

JACK

- JACK is an agent development environment produced by the Agent Oriented Software Group - first released in 1998 - http://www.aosgrp.com
- There are two principles underpinning the development of JACK
  - operates on top of the Java programming language, acting as an extension that provides agent-related concepts
  - based on the Belief-Desire-Intention architecture
- The development environment has three main components:
  - The JACK Agent Language is a superset of the Java language, and introduces new semantic and syntactic features, new base classes, interfaces, and methods to deal with agent-oriented concepts
  - The JACK Compiler compiles the JACK Agent Language down to pure Java, so that the resulting agents can operate on any Java platform
  - The JACK Agent Kernel is the runtime program within which JACK agents operate, and provides the underlying agent functionality that is defined within the JACK Agent Language

Jack architecture



JACK - Agents

- Agents schedule actions using the TaskManager
- Plans are sequences of actions that agents execute on recording an event
  - No automated planning !
- Events within the agent architecture are divided into:
  - external events (such as messages from other agents);
  - internal events initiated by the agent itself;
  - and motivations (which are described as goals that the agent wants to achieve)
- Capabilities provide a means for structuring a set of reasoning elements into a coherent cluster that can be plugged into agents

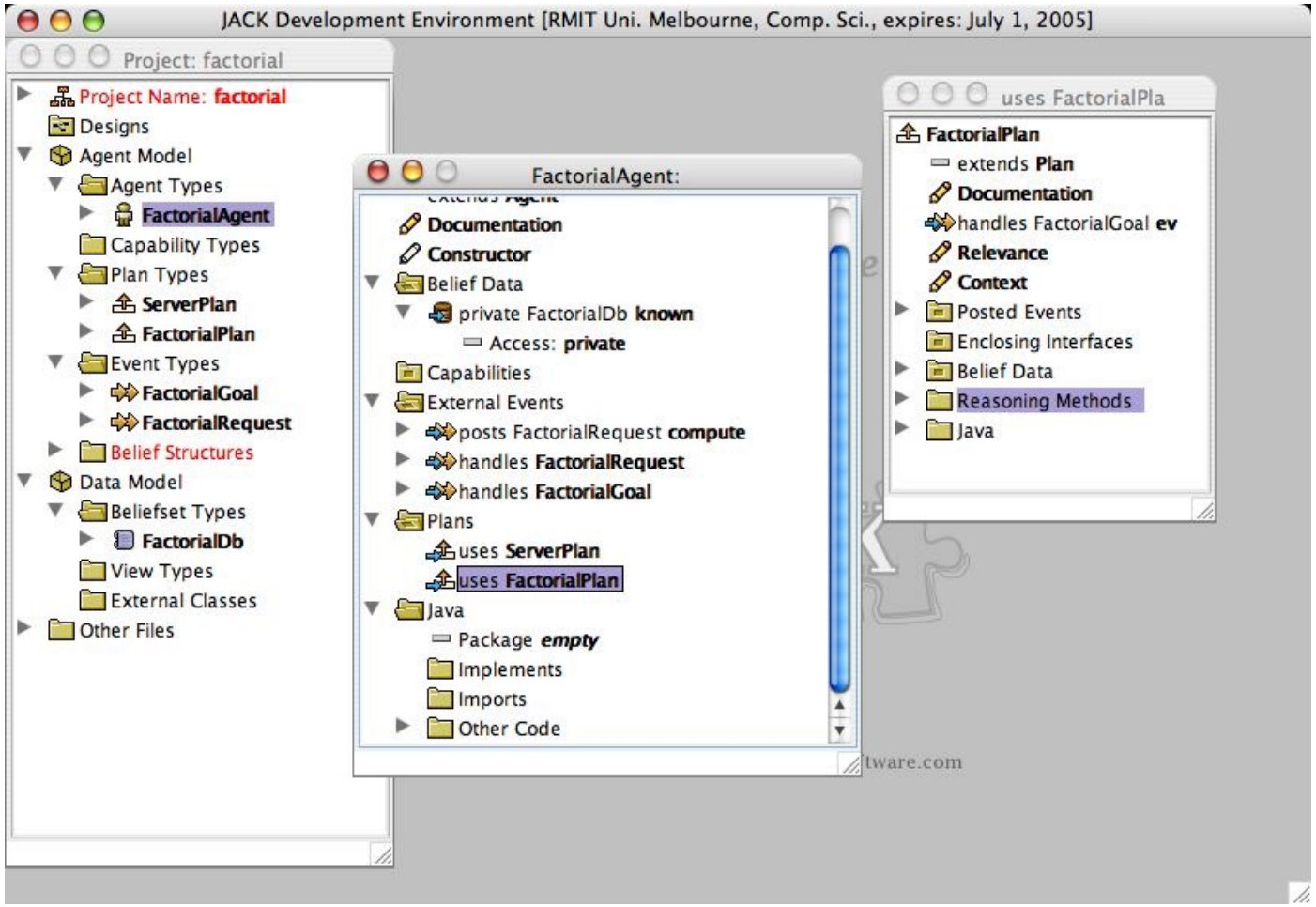
JACK – Multi-agent systems

- Networking capabilities in JACK are based on UDP over IP, with a thin layer of management on top of that to provide reliable peer-to-peer communication
- Agent communication between agents is handled by the JACK Kernel, which handles the routing of messages and the interface with lower-level networking infrastructure
- A rudimentary Agent Name Server is provided
- FIPA ACL is supported (IEEE standard)

JACK – Supporting Software

- JACK provides a comprehensive, graphical agent development environment
  - A high level design tool allows a multi-agent system application to be designed by defining the agents and relationships between them, in a notation similar to UML
  - A plan editor allows plans to be specified as decision diagrams
  - A plan tracing tool and an agent interaction tool allow developers to visualize the monitoring of an application
- An application can be monitored through an Agent Tracing Controller, which allows a developer to choose which agents to trace and provides a visual representation of the agents stepping through their plans

Jack - JDE


JACK Development Environment [RMIT Uni. Melbourne, Comp. Sci., expires: July 1, 2005]
Project: factorial
Project Name: factorial
Designs
Agent Model
Agent Types
FactorialAgent
Capability Types
Plan Types
ServerPlan
FactorialPlan
Event Types
FactorialGoal
FactorialRequest
Belief Structures
Data Model
Beliefset Types
FactorialDb
View Types
External Classes
Other Files
FactorialAgent:
Documentation
Constructor
Belief Data
private FactorialDb known
Access: private
Capabilities
External Events
posts FactorialRequest compute
handles FactorialRequest
handles FactorialGoal
Plans
uses ServerPlan
uses FactorialPlan
Java
Package empty
Implements
Imports
Other Code
uses FactorialPla
FactorialPlan
extends Plan
Documentation
handles FactorialGoal ev
Relevance
Context
Posted Events
Enclosing Interfaces
Belief Data
Reasoning Methods
Java

JAVA Agent DEvelopment Framework

is an open source platform for peer-to-peer agent based applications

JADE (Java Agent DEvelopment Framework) is a software Framework fully implemented in the Java language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools that support the debugging and deployment phases. A JADE-based system can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another, as and when required. JADE is completely implemented in Java language and the minimal system requirement is the version 5 of JAVA (the run time environment or the JDK).

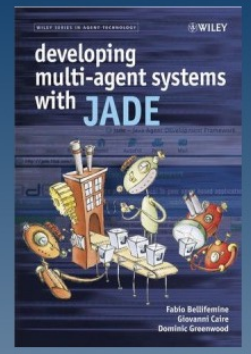
Besides the **agent abstraction**, JADE provides a simple yet powerful **task** execution and composition model, peer to peer agent **communication** based on the asynchronous message passing paradigm, a **yellow pages** service supporting publish subscribe discovery mechanism and many other advanced features that facilitates the development of a distributed system.

Thanks to the contribution of the LEAP project, ad hoc versions of JADE exist designed to deploy JADE agents transparently on different Java-oriented environments such as **Android** devices and

Latest news

- JADE 4.5 and WADE 3.6 have been released 08/06/2017
- JADE 4.4, WADE 3.5 and AMUSE 1.5 have been released 23/12/2015

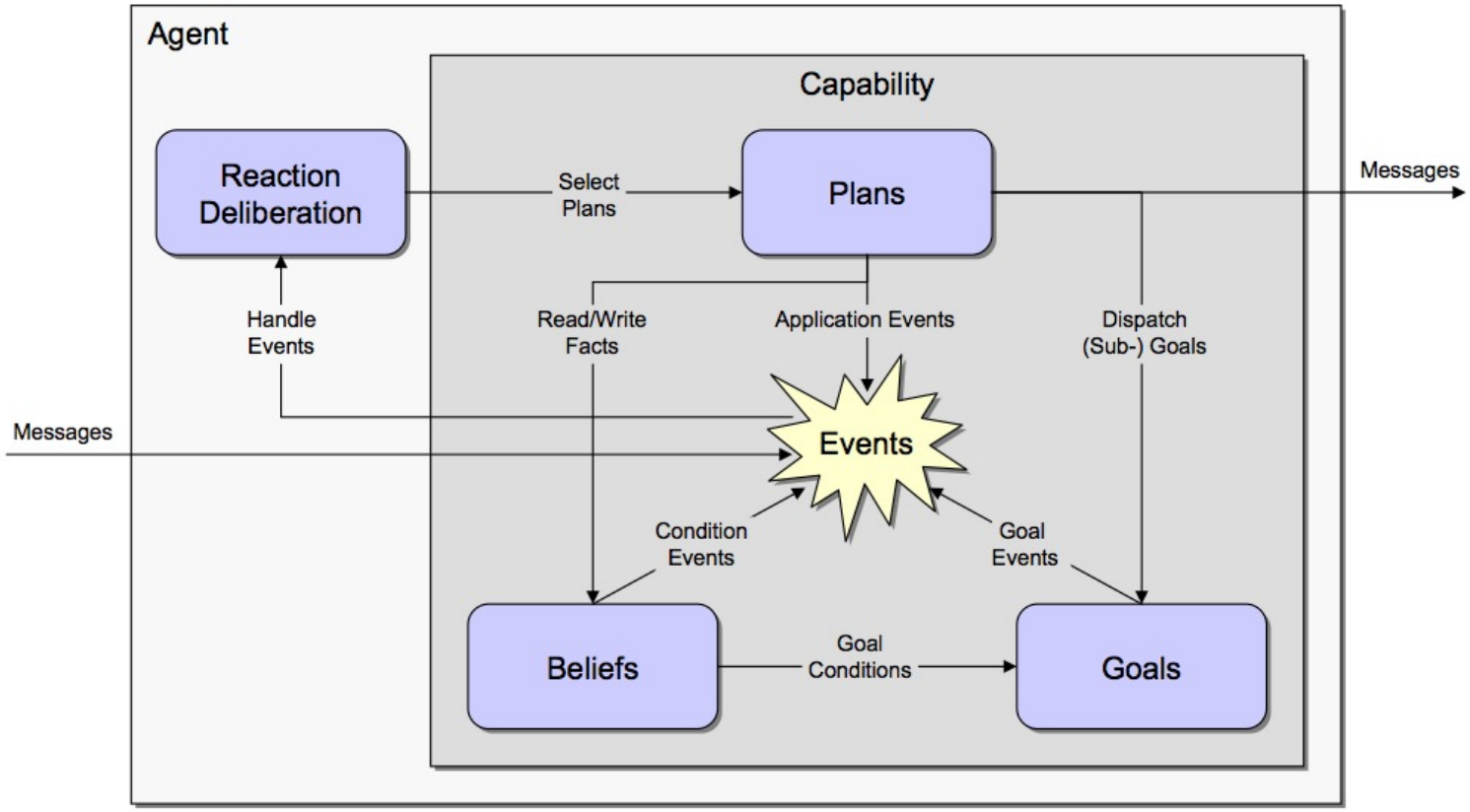
Jade Book



Developing Multi-Agent Systems with JADE can be ordered from Wiley

Jadex

- Jadex is based on JADE
  - Explicit representation of goals allows reasoning about goals



Jadex goals

- Generic goal types
  - perform (some action)
  - achieve (a specified world state)
  - query (some information)
  - maintain (reestablish a specified world state whenever violated)
- Are strongly typed with
  - name, type, parameters
  - BDI-flags enable non-default goal-processing
- Goal creation/deletion possibilities
  - initial goals for agents
  - goal creation/drop conditions for all goal kinds – top-level / subgoals from within plans