Belief, Desire, Intention Agents

Agent-Oriented Software Engineering

A.A. 2019-2020

Prof. Paolo Giorgini

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

Practical vs. Theoretical Reasoning

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 affairs 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 would 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 would 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.

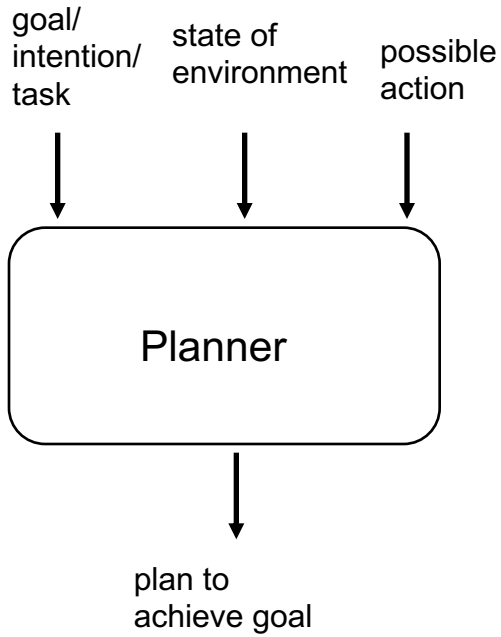
What is Means-End Reasoning?

- 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

- Essentially, this is

automatic programming



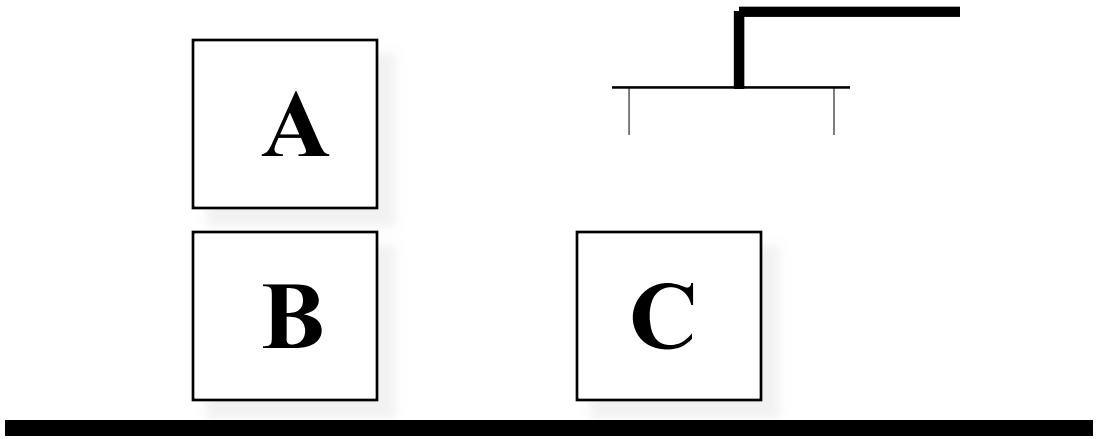
Planning

- Question: How do we *represent*. . .
  - goal to be achieved
  - state of environment
  - actions available to agent
  - plan itself

The STRIP approach

- First and most influential planner in AI history (late 60s).
- Two basic knowledge components:
  - model of the world as a set of first-order logic formulae
  - a set of *action schemata:* preconditions and effects of all actions available to the planner

The Blocks World

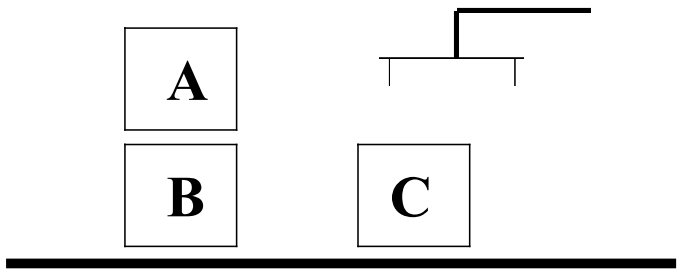


- We'll illustrate the techniques with reference to the *blocks world* (like in the previous class)
- Contains a robot arm, 3 blocks (A, B, and C) of equal size, and a table-top

The Blocks World

- To represent this environment, need an *ontology*

On(x, y)	obj x on top of obj y
OnTable(x)	obj x is on the table
Clear(x)	nothing is on top of obj x
Holding(x)	arm is holding x
ArmEmpty	arm is empty (not holding anything)



- Here is a representation of the blocks world described above:

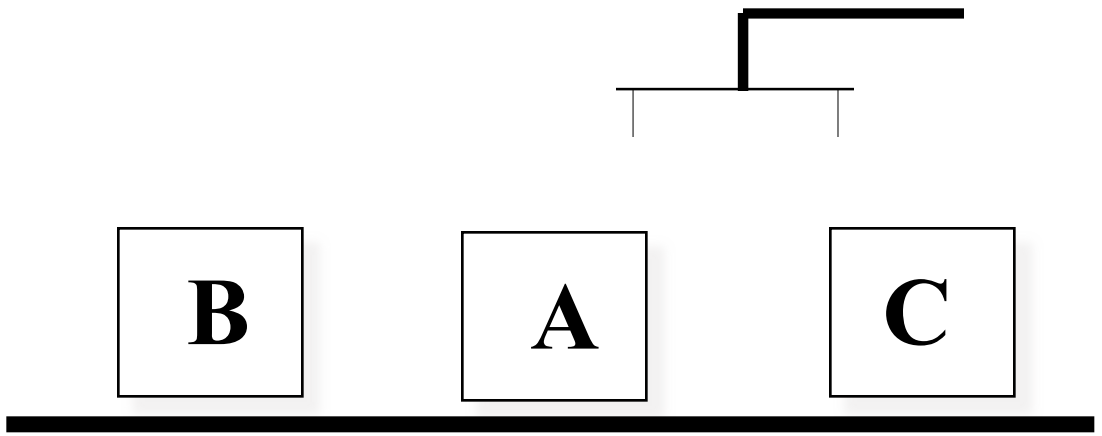
Clear(A)
On(A, B)
OnTable(B)
OnTable(C)

- Use the *closed world assumption*: anything not stated is assumed to be *false*

The Blocks World - goals

- A *goal* is represented as a set of formulae
- Here is a goal:

$$OnTable(A) \wedge OnTable(B) \wedge OnTable(C)$$



The Blocks World - actions

- *Actions* are represented using a technique that was developed in the STRIPS planner
- Each action has:
  - a *name*
    which may have arguments
  - a *pre-condition list*
    list of facts which must be true for action to be executed
  - a *delete list*
    list of facts that are no longer true after action is performed
  - an *add list*
    list of facts made true by executing the action

Each of these may contain *variables*

Actions

- `Stack(x,y)`
  - the arm places an object x on top of an object y
- `UnStack(x,y)`
  - the arm picks an object x from on top of an object y
- `PickUp(x)`
  - the arm picks up an object x from the table
- `PutDown(x)`
  - the arm places an object x onto the table

Actions: pre and post cond.s

Action	Precondition list	Delete list	Add list
Stack(*x*,*y*)	{*Clear*(*y*), *Holding*(*x*)}	{*Clear*(*y*), *Holding*(*x*)}	{*ArmEmpty*, *On*(*x*,*y*)}
UnStack(*x*,*y*)	{*On*(*x*,*y*), *Clear*(*x*), *ArmEmpty*}	{*On*(*x*,*y*), *ArmEmpty*}	{*Holding*(*x*), *Clear*(*y*)}
PickUp(*x*)	{*Clear*(*x*), *OnTable*(*x*), *ArmEmpty*}	{*OnTable*(*x*), *ArmEmpty*}	{*Holding*(*x*)}
PutDown(*x*)	{*Holding*(*x*)}	{*Holding*(*x*)}	{*ArmEmpty*, *OnTable*(*x*)}

The Blocks World Operators

Example 1:

The *stack* action occurs when the robot arm places the object x it is holding is placed on top of object y.

Stack(x, y)
pre $Clear(y) \wedge Holding(x)$
del $Clear(y) \wedge Holding(x)$
add $ArmEmpty \wedge On(x, y)$

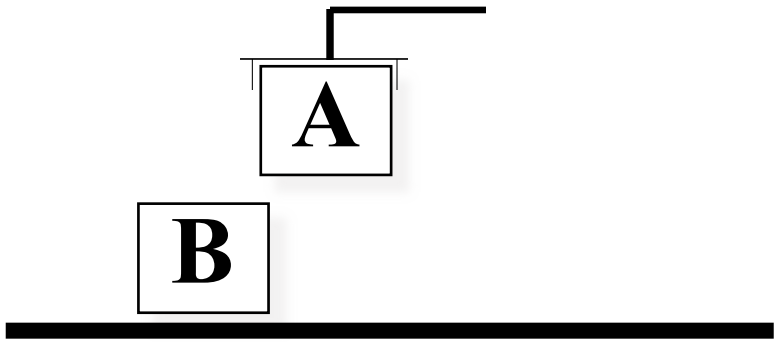
The Blocks World Operators

Example 2:

The *unstack* action occurs when the robot arm picks an object x up from on top of another object y.

UnStack(x, y)
pre $On(x, y) \wedge Clear(x) \wedge ArmEmpty$
del $On(x, y) \wedge ArmEmpty$
add $Holding(x) \wedge Clear(y)$

Stack and UnStack are *inverses* of one-another.



The Blocks World Operators

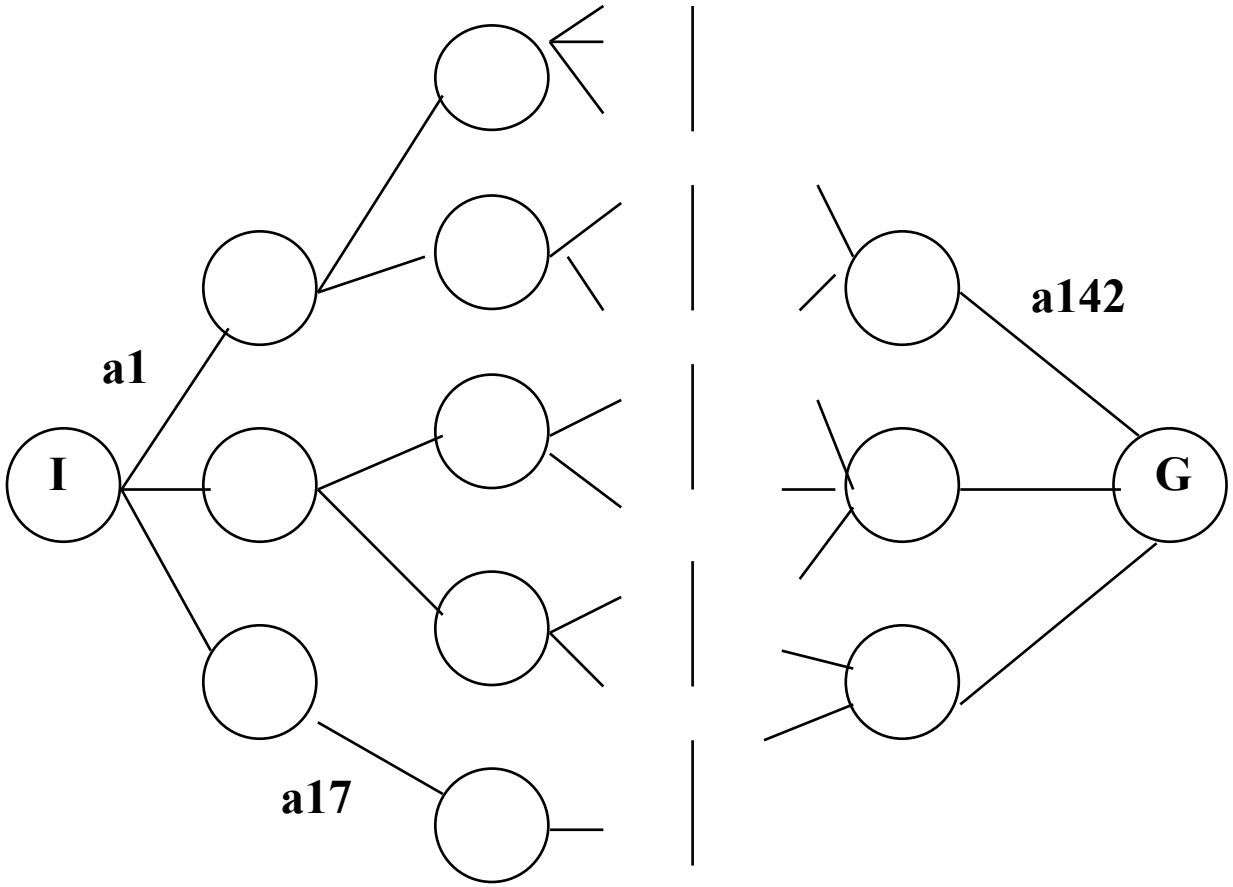
- Example 3:
  The *pickup* action occurs when the arm picks up an object x from the table.

  Pickup(x)
  pre $Clear(x) \wedge OnTable(x) \wedge ArmEmpty$
  del $OnTable(x) \wedge ArmEmpty$
  add $Holding(x)$

- Example 4:
  The *putdown* action occurs when the arm places the object x onto the table.

  Putdown(x)
  pre $Holding(x)$
  del $Holding(x)$
  add $Clear(x) \wedge OnTable(x) \wedge ArmEmpty$

A Plan



- What is a plan?
  A sequence (list) of actions, with variables replaced by constants.

The STRIPS approach

- The original STRIPS system used a goal stack to control its search
- The system has a database and a goal stack, and it focuses attention on solving the top goal (which may involve solving subgoals, which are then pushed onto the stack, etc.)

Search “off-line”, then execute with “eyes closed”

The Basic STRIPS Idea

- Place goal on goal stack:

- Considering top Goal1, place onto it its subgoals:

- Then try to solve subgoal GoalS1-2, and continue…

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
```

- (We will not be concerned with stages (2) or (3))

Implementing Practical Reasoning Agents

- Problem: deliberation and means-ends reasoning processes are not instantaneous.
  They have a *time cost* !!!
- Suppose the agent starts deliberating at t_0, begins means-ends reasoning at t_1, and begins executing the plan at time t_2. Time to deliberate is

$$t_{deliberate} = t_1 - t_0$$

- and time for means-ends reasoning is

$$t_{me} = t_2 - t_1$$

Implementing Practical Reasoning Agents

- Further, suppose that deliberation is *optimal* in that if it selects some intention to achieve, then this is the best thing for the agent. (Maximizes expected utility.)
- So at time t_1, the agent has selected an intention to achieve that would have been optimal *if it had been achieved at* t_0.
  But unless $t_{deliberate}$ is vanishingly small, then the agent runs the risk that the intention selected is no longer optimal by the time the agent has fixed upon it.
- This is *calculative rationality*.
- Deliberation is only half of the problem: the agent still has to determine *how* to achieve the intention.

Implementing Practical Reasoning Agents

So, the agent will have overall optimal behavior in the following circumstances:

1. When deliberation and **means-ends reasoning take a vanishingly small** amount of time; or
2. When the **world** is guaranteed to **remain static while the agent is deliberating and performing means-ends reasoning**, so that the assumptions upon which the choice of intention to achieve and plan to achieve the intention remain valid until the agent has completed deliberation and means-ends reasoning; or
3. When an **intention** that is optimal when achieved at time t_0 (the time at which the world is observed) is guaranteed **to remain optimal until time t_2** (the time at which the agent has found a course of action to achieve the intention).

Implementing Practical Reasoning Agents

- Let's make the algorithm more formal:

```
Agent Control Loop Version 2
1.   B := B_0; /* 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
- Chosen options are then intentions
- The *deliberate* function can be decomposed into 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 competing 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 := 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.       execute(π)
11.   end while
```

Selects the best
option for the
agent to commit to

Commitment Strategies

- The following *commitment strategies* are commonly discussed in the literature of rational agents:
  - *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.
  - *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.
  - *Open-minded commitment*
    An open-minded agent will maintain an intention as long as it is still believed possible.

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)
- Currently, our agent control loop is overcommitted, both to means and ends

- Modification: *replan* if ever a plan goes wrong

If it ever determines that its plan is no longer appropriate in order to achieve the current intention, then it engages further means-ends reasoning in order to find an alternative plan.

Given that its beliefs are updated every time it executes an action, this implies at least some degree of reactivity

```
Agent Control Loop Version 4
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) 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
```

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*)

```
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
```

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
- Modification: 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
```

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
```

Possible Interactions

- The possible interactions between meta-level control and deliberation are:

Situation number	Chose to deliberate?	Changed intentions?	Would have changed intentions?	*reconsider*(...) optimal?
1	No	—	No	Yes
2	No	—	Yes	No
3	Yes	No	—	No
4	Yes	Yes	—	Yes

Intention Reconsideration

- In situation (1), the agent did not choose to deliberate, and as consequence, did not choose to change intentions. Moreover, if it *had* chosen to deliberate, it would not have changed intentions. In this situation, the *reconsider(…)* function is behaving optimally.
- In situation (2), the agent did not choose to deliberate, but if it had done so, it *would* have changed intentions. In this situation, the *reconsider(…)* function is not behaving optimally.
- In situation (3), the agent chose to deliberate, but did not change intentions. In this situation, the *reconsider(…)* function is not behaving optimally.
- In situation (4), the agent chose to deliberate, and did change intentions. In this situation, the *reconsider(…)* function is behaving optimally.
- An important assumption: cost of *reconsider(…)* is *much* less than the cost of the deliberation process itself.

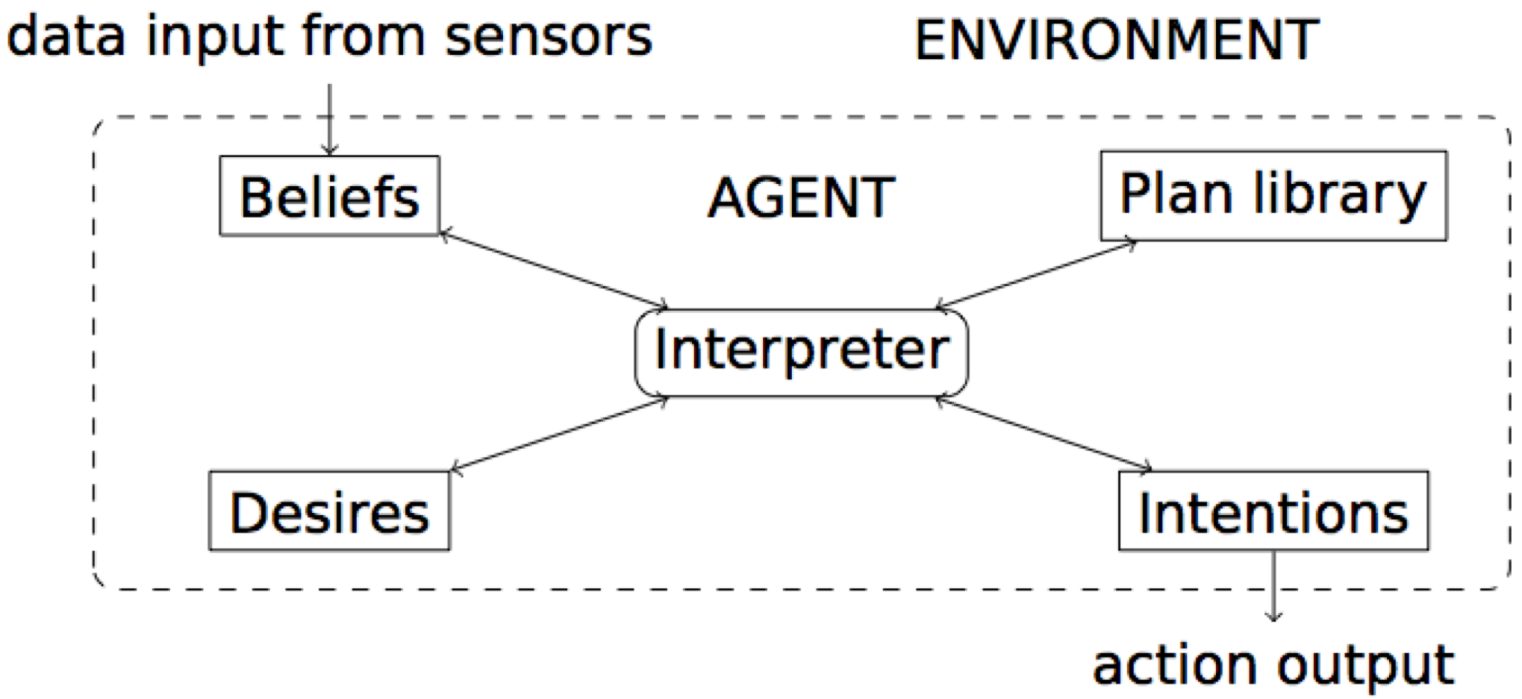
Optimal Intention Reconsideration

- Kinny and Georgeff's experimentally investigated effectiveness of intention reconsideration strategies
- Two different types of reconsideration strategy were used:
  - *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

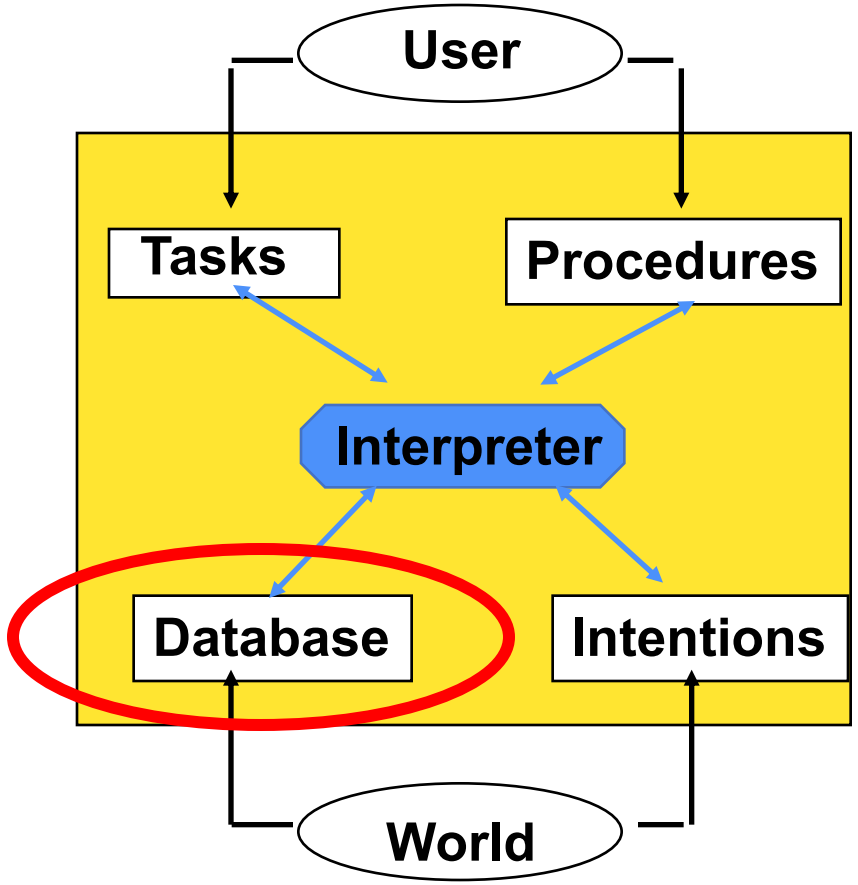
- Perhaps the first and most influential BDI agent architecture (Georgeff, Lansky)
- each agent is equipped with a *plan library*, representing that agent's *procedural knowledge*
  - knowledge about the mechanisms that can be used by the agent in order to realize its intentions
- Peculiar management 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



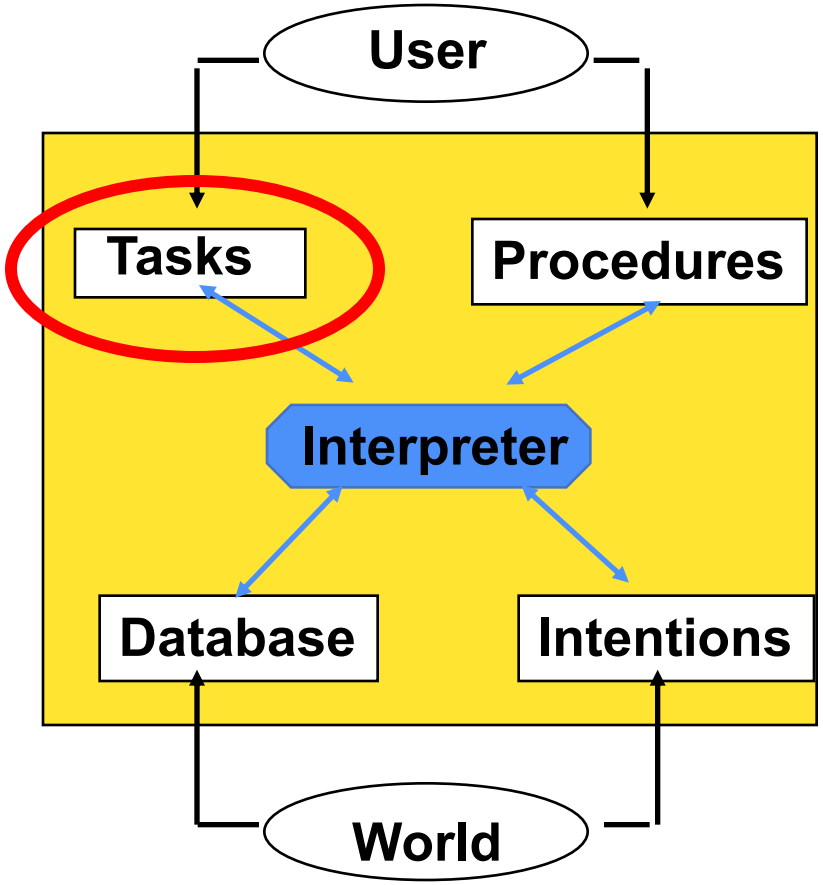
PRS Architecture: Database

- Contains beliefs or facts about the world
- Includes meta-level information
  - Eg goal G is active



PRS Architecture: Tasks

- Represent desired behavior
- Conditions over some time interval
  - eg (walk a b): set of behaviors in which agent walks from a to b)

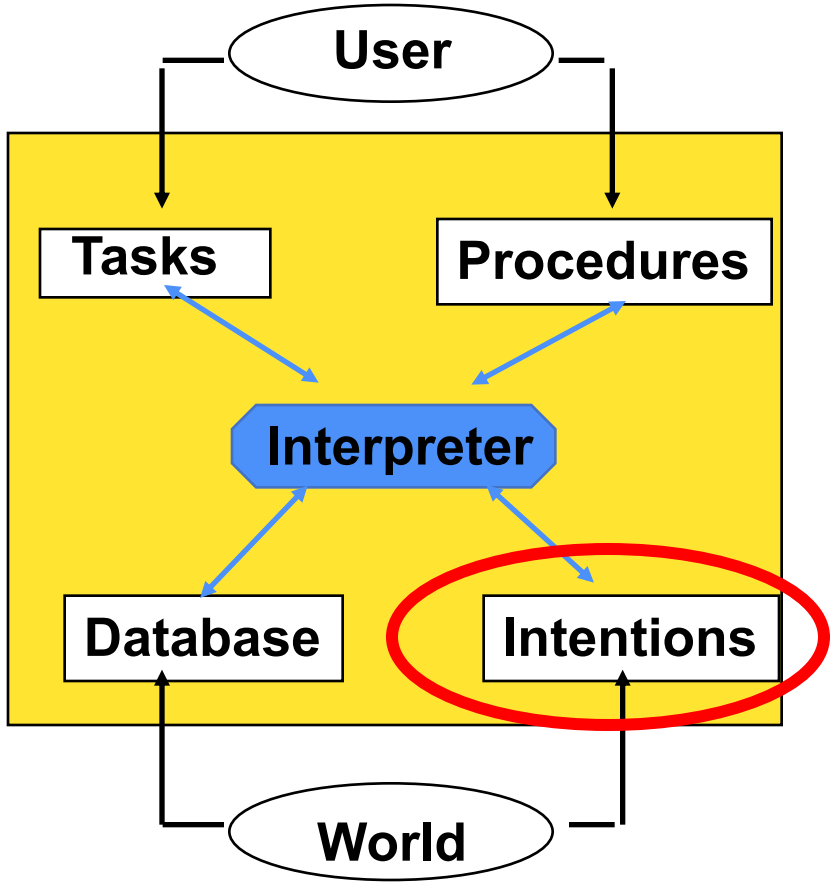


Expressing Tasks in a Dynamic Environment

- (! P) -- achieve P
- (? P) -- test P
- (# P) -- maintain P
- (^ C) -- wait until C
- (-> C) -- assert C
- (~> C) -- retract C

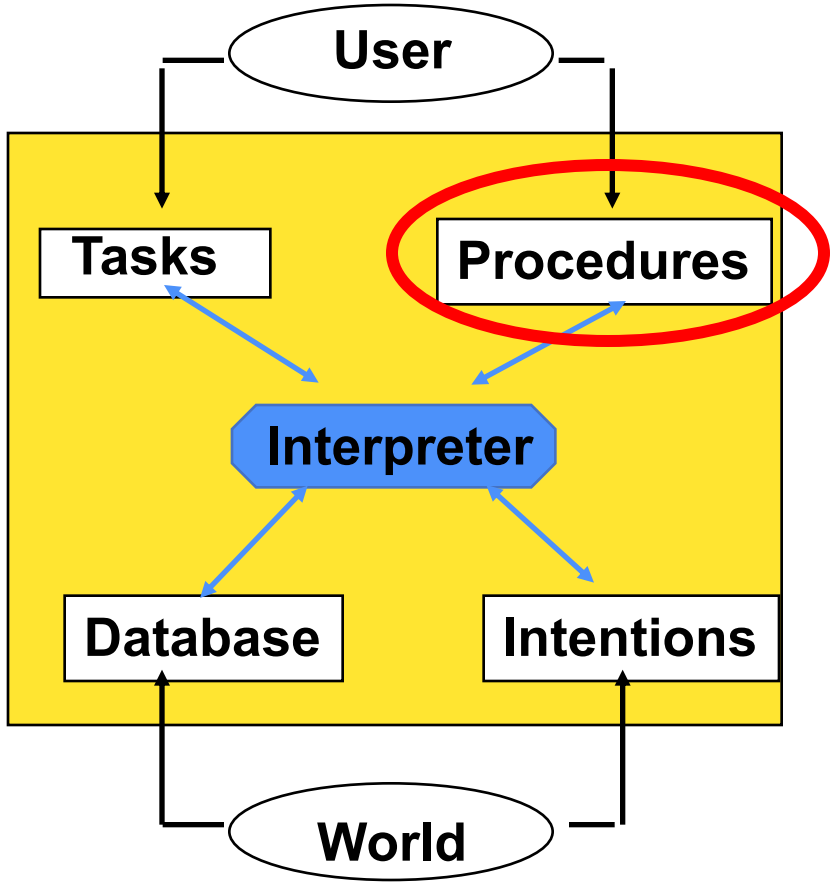
PRS Architecture: Intentions

- Currently active procedures
- Procedure currently being executed



PRS Architecture: Procedures

- Pre-compiled procedures
- Express actions and tests to achieve goals or to react to conditions



Representing Procedures with Act Formalism

Environment conditions

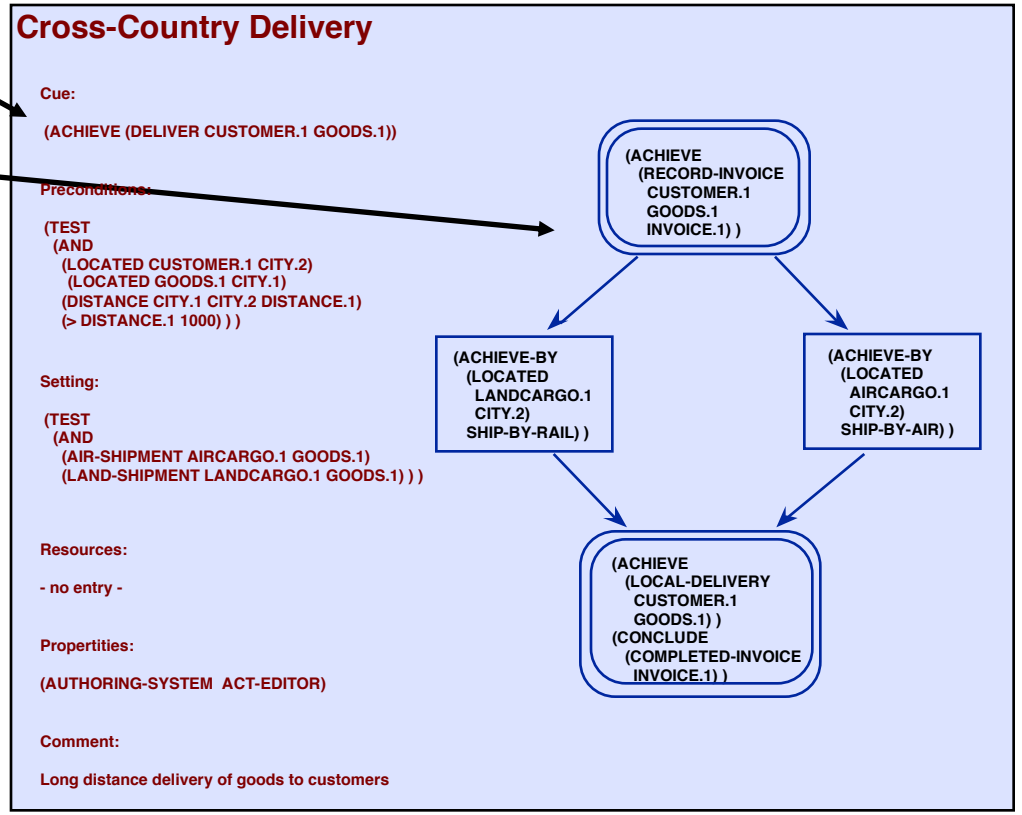
- Purpose (goal or condition)
- applicability criteria

Plot

- directed graph
- partially ordered conditional & parallel actions, loops
- Successful node execution by achievement of node's goals
- If no body: primitive action

Metapredicates

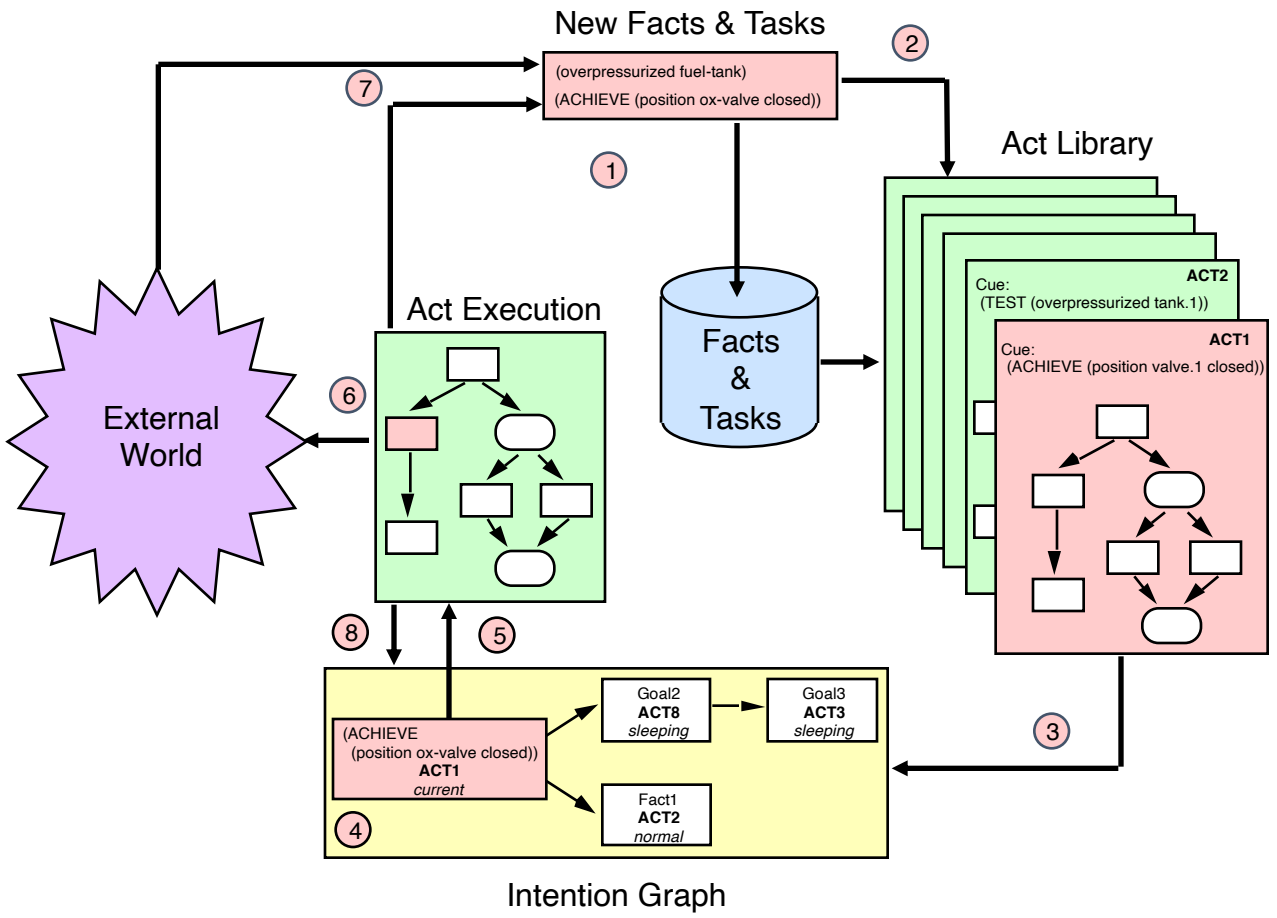
- Achieve
- Test
- Wait-Until
- Require-Until
- – Achieve-By {proc}
- – Conclude {effects}
- – Use-Resource



PRS Interpreter

Execution Cycle

1. New information arrives that updates facts and/or tasks
2. Acts are triggered by new facts or tasks
3. A triggered Act is intended
4. An intended Act is selected
5. That intention is activated
6. An action is performed
7. New facts or tasks are posted
8. Intentions are updated



PRS execution

PRS execution:

- An agent is initialized with some initial beliefs and a top-level goal
- The top-level goal is pushed onto the agent's *intention stack.*
- A plan is selected (deliberation: e.g., by utilities) among those whose post-condition matches the goal. If no such plan exists, the agent has failed.
- Each time a goal is found in a plan's body, it is pushed onto the intention stack.
- And so on, until the stack is empty.

JACK - Background

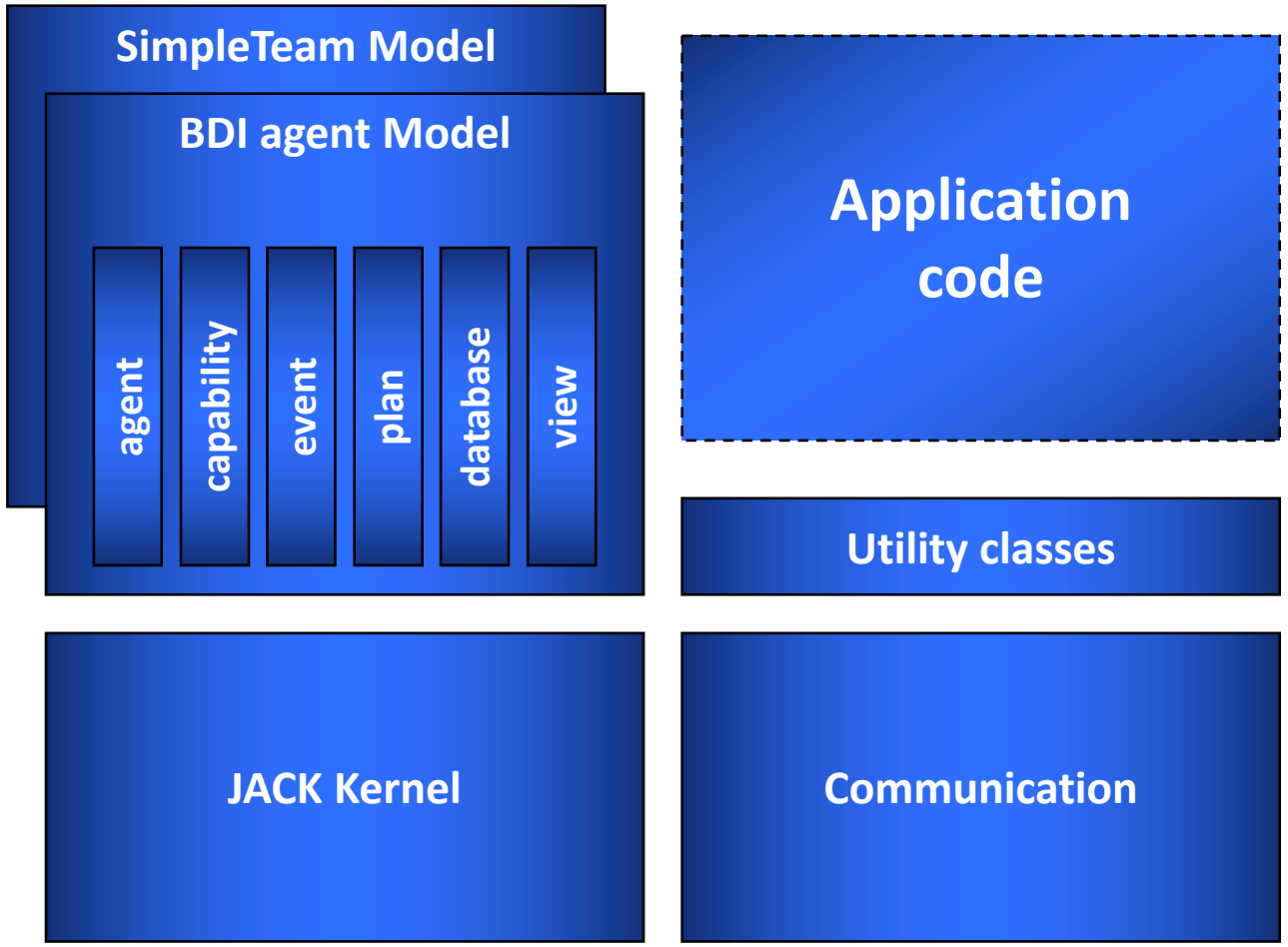
- JACK is an agent development environment produced by the Agent Oriented Software Group - first released in 1998 and currently at version 5.3 - http://www.aosgrp.com

- There are two principles underpinning the development of JACK
  - Agent-oriented development can be thought of as an extension of object-oriented development. As a result, JACK operates on top of the Java programming language, acting as an extension that provides agent-related concepts
  - Agents in JACK are intelligent agents in that they are based on the Belief-Desire-Intention architecture

Jack environment

- The JACK development environment can be divided into 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, including concurrent actions, using the *TaskManager*
- Beliefs represent the knowledge that an agent possesses about the world
- Plans are sequences of actions that agents execute on recording an event
- 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 Agent Language

Class Constructs

- Agent, Event, Plan, Capability, Database, View

Declarations

- #handles, #uses, #posts, #sends, #reads, ...

Reasoning Method Statements

- @wait-for, @maintain, @send, @reply, @subtask, @post, @achieve, @insist, @test, @determine

Where are the goals?

Where are the Goals?

- When we talk about agents we talk about goals
  - Pro-actively achieving goals…..
  - Achieving multiple concurrent goals…..
  - In design we ask you to: Identify system goals, subsidiary goals……
- But BDI implementations (such as JACK) don't explicitly model goals - instead, they treat "acquire new goal" as an event
  - Issue: if goal not handled, dropped (not persistent)
  - Issue: hard to reason about goals

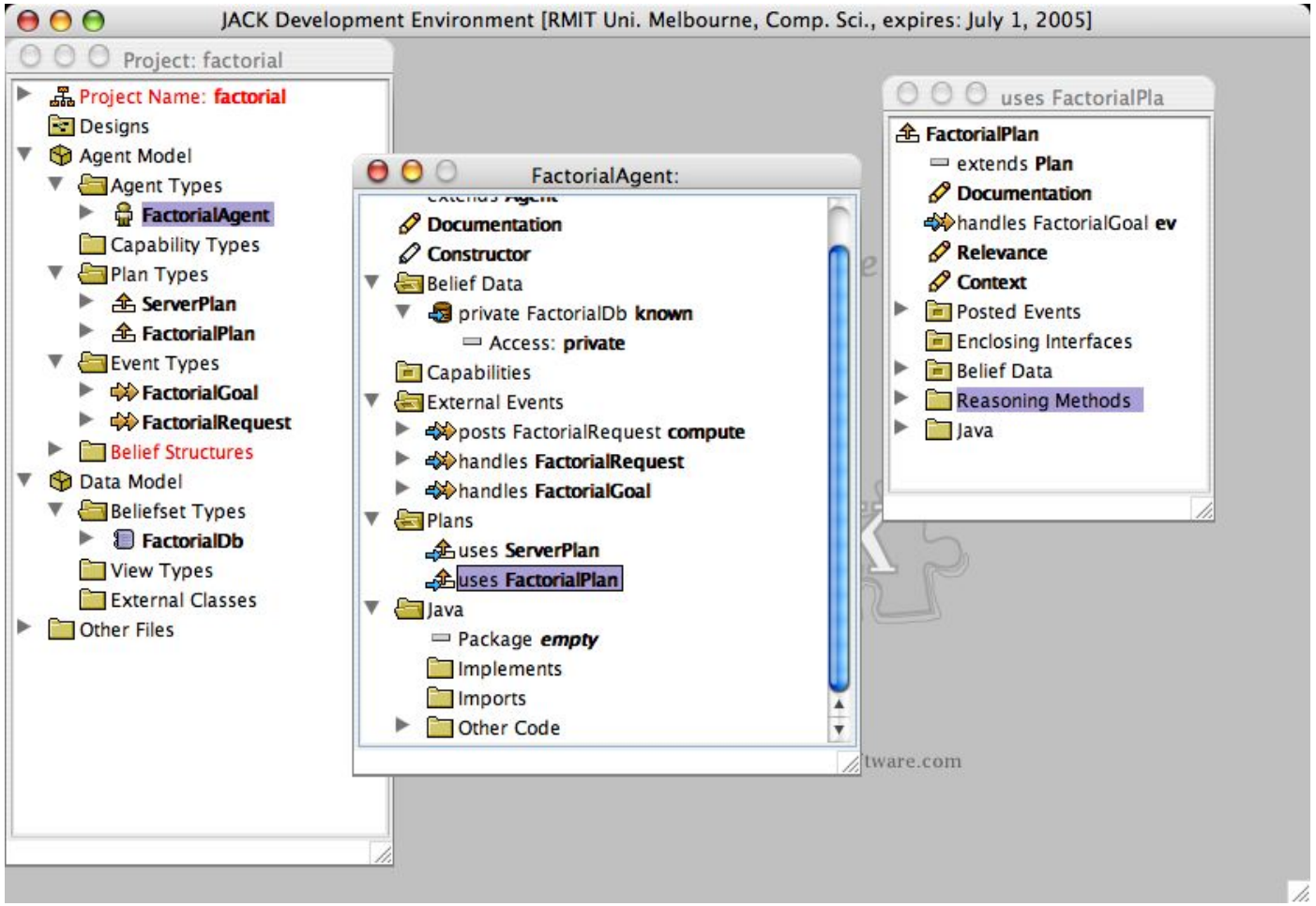
JACK – Multi-agent systems

- 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

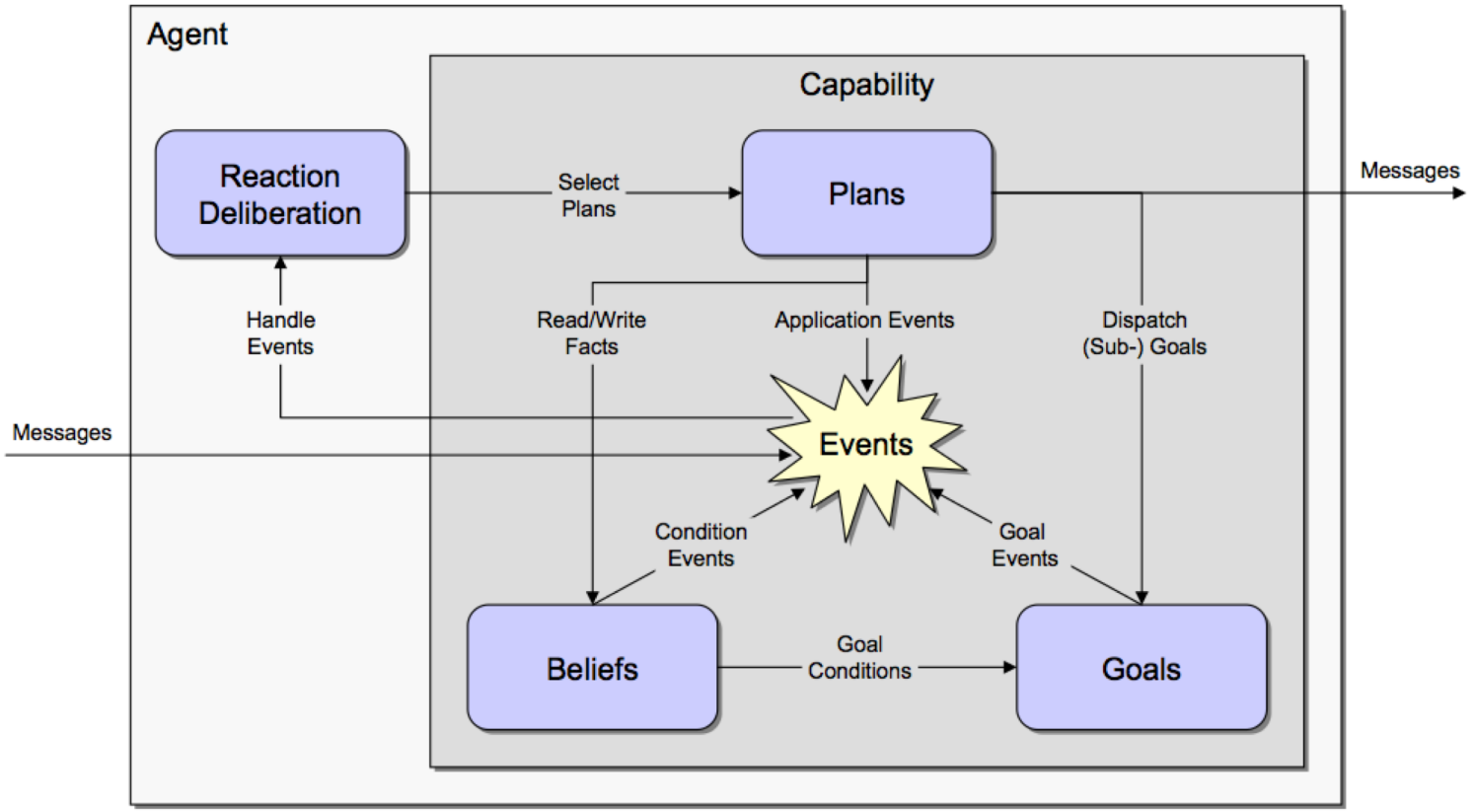
Jack - JDE


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

Jadex

- Integrate agent theories with object-orientation and XML descriptions
- Object-oriented representation of BDI concepts
- Explicit representation of goals allows reasoning about (manipulation of) goals
- Jadex is based on JADE Platform

Jadex architecture



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