Introduction to JACK

Intelligent Software Agents

Michael Papasimeon

Intelligent Agent Lab

22 October 2003

Outline

- What is JACK?
- Review of the BDI Model
- JACK Concepts
  - Agents
  - Beliefs
  - Events
  - Plans
  - Capabilities
- The JACK Development Environment
- JDE Design Diagrams
- Resources

What is JACK?

- Commercial Agent Toolkit
  - Developed by Agent Oriented Software (AOS) in Melbourne
- Agent oriented programming language
- Team oriented programming language
- A set agent oriented extensions to Java and an agent oriented run time environment.
- Agent oriented development environment (JDE)
- A set of supporting tools and infrastructure

JACK Applications

- Modelling human decision making in multi-agent simulations.
- Control of unmanned vehicles.
- RoboCup
- Manufacturing
- ...

Beliefs, Desires and Intentions

Internal mental attitudes of a rational BDI agent (or mental state):

Beliefs

What an agent believes about the world, itself and other agents (informational).

Desires

What an agent want to achieve (motivational).

Intentions

How the agent tries to achieve desires (deliberational).

Rational Agency and BDI

Daniel Dennet: Folk Psychology

Michael Bratman: Rational Agency

Rao and Georgeff: Formal Logical Framework

Programming Languages: PRS, dMARS, JACK, JAM, C-PRS, IRMA

Theoretical BDI Interpreter (Rao and Georgeff)

BDI Interpreter

```
initialize-state();
repeat
       options := option-generator(event-queue);
       selected-options := deliberate(options);
       update-intentions(selected-options);
       execute();
       get-new-external-events();
       drop-successful-attitudes();
       drop-impossible-attitudes();
end repeat
```

Basic Agent Control Loop 1

Adapted from Wooldridge...

procedure AGENT CONTROL LOOP 1
  while True **do**
    observe-the-world();
    update-internal-world-model();
    deliberate-about-what-intention-to-achieve-next()
    use-means-end-reasoning-to-get-a-plan-for-next-intention()
    execute-the-plan
  end while
end procedure

Basic Agent Control Loop 2

Adapted from Wooldridge...

procedure AGENT CONTROL LOOP 2(B_0)
    $B \leftarrow B_0$
    while True **do**
        $\rho \leftarrow$ `get_next_percept()`;
        $B \leftarrow$ `brf`(B, ρ);
        $D \leftarrow$ `deliberate`(B);
        $\pi \leftarrow$ `plan`(B, I);
        `execute`(π);
    end while
end procedure

Basic Agent Control Loop 3

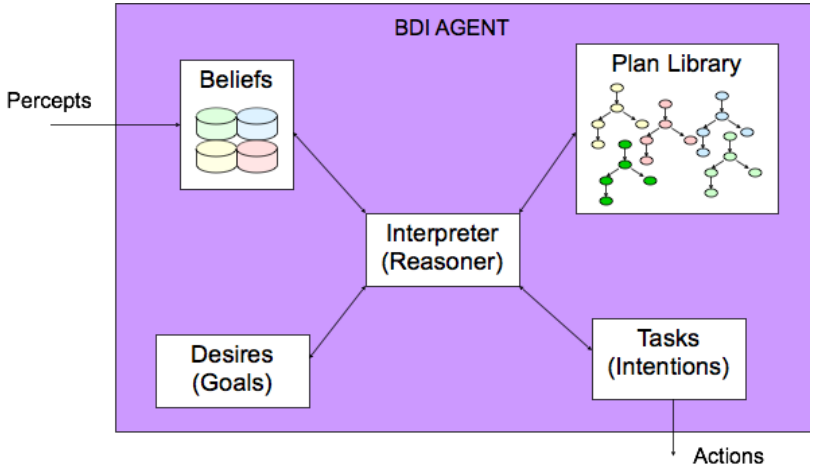
Adapted from Wooldridge...

procedure AGENT CONTROL LOOP 3(B_0, I_0)
    $B \leftarrow B_0$
    $I \leftarrow I_0$
    while True **do**
        $\rho \leftarrow$ `get_next_percept()`;
        $B \leftarrow$ `brf`(B, ρ);
        $D \leftarrow$ `options`(BI);
        $I \leftarrow$ `filter`(B, D, I);
        $\pi \leftarrow$ `plan`(B, I);
        `execute`(π);
    end while
end procedure

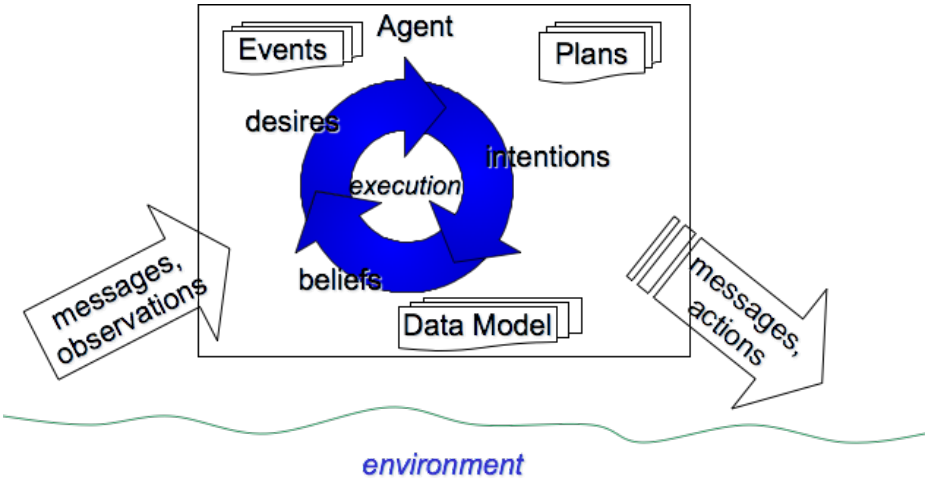
Ideas Behind JACK

- Implementations of the BDI model
- Idea of plans as reciples (pre-planning)
- Least commitment
- Bounded rationality
- Dynamic environment
- Goals, beliefs, plans
- Intentions and run-time (not design time) constructs

A BDI Agent Architecture



BDI Execution Concepts in JACK



BDI Dynamics (1)

1. An event occurs.
   - A goal is posted (internal).
   - A change in the environment and hence a change in belief (external).
2. Agent reasoner searches through the plan library to find the set of plans which can handle this event (defined by the invocation condition).
3. This may result in in 10 plans out of 500 which can handle the event. Out of these 10 plans, the agent reasoner then chooses only those which are appropriate for this **context** – that is, the current situation.

BDI Dynamics (2)

5. This may result in 6 plans out of the 10 which are **applicable** in this context.
6. The agent then chooses one of the plans, puts it on the **intention stack**, and starts executing the plan steps in the plan.
7. This executing plan is called an **intention** to achieve the original **goal**.
8. If the plan fails, the agent will try on of the other applicable plans until one of them succeeds in achieving the goal or all of them fail, in which case the goal will fail.

BDI Dynamics Notes

- It is possible to determine which plan is chosen in the applicable plan set by using **meta-level reasoning**.
- Plans can wait until particular beliefs are satisfied.
- Plan steps can involve trying to achieve **sub-goals**.
- When trying to achieve a sub-goal, the existing plan is suspended and the new plan is put on top of the **intention stack**.

Setting Up the JACK Environment

These instructions were correct in 2003

Needs Java 1.4.x, does not yet work with Java 1.5

Windows `set CLASSPATH=c:\ aos\ lib\ jack.jar;.`

Bash `CLASSPATH=/aos/jack/lib/jack.jar:. export CLASSPATH`

csh/tcsh `setenv CLASSPATH=/usr/local/aos/jack/lib/jack.jar:.`

JACK File Types

.jack	Any JACK object definition.
.agent	JACK agent definition.
.cap	JACK capability definition.
.plan	JACK plan definition.
.event	JACK event definition.
.bel	JACK beliefset definition.
.view	JACK view definition.
.java	Java class or interface definition.

Compiling JACK

The JACK Build is a two step process:

1. Generate Java code from the original `.agent`, `.plan`, `.bel` `.event` etc. files.
2. Compile the Java code

Building JACK code...

```
$ java aos.main.JackBuild
```

Running is the same as Java programs...

```
$ java Test
```

JACK Agent Language (JAL)

The JACK Agent Language extends Java to support Agent Oriented programming:

- It defines new base classes, interfaces and methods.
- It provides extensions to the Java syntax to support new agent-oriented classes, definitions and statements.
- It provides semantic extensions (runtime differences) to support the execution model required by an agent-oriented software system.

Class Level Constructs in JAL

Agent Used to define the behaviour of an intelligent software agent. This includes capabilities an agent has, what type of messages and events it responds to and which plans it will use to achieve its goals.

Capability Allows the functional components that make up an agent to be aggregated and reused. A capability can be made up of plans, events, beliefsets and other capabilities that together serve to give an agent certain abilities.

BeliefSet The beliefset construct represents agent beliefs using a generic relational model.

View The view construct allows general purpose queries to be made about an underlying data model. The data model may be implemented using multiple beliefsets or arbitrary Java data structures.

Event The event construct describes an occurrence in response to which the agent must take action.

Plan An agent's plans are analogous to functions. They are the instructions the agent follows to try to achieve its goals and handle its designated events.

Example Agent Plan

```
plan MovementResponse extends Plan
{
	#handles event RobotMoveEvent moveResponse;
	#uses agent implementing RobotInterface robot;

	static boolean relevant (RobotMoveQEvent ev)
	{
		return (ev.ID == RobotMoveQEvent.REPLY_SAFE || ev.ID == RobotMoveQEvent.REPLY_DEAD);
	}

	body()
	{
		if (moveResponse.ID==REPLAY_SAFE)
		{
		System.err.println("Robot Safe to Move");
		robot.updatePosition(moveResponse.Lane, moveResponse.Displacement);
		}
		else
		{
			// robot didn't make it
			System.err.println("Robot is Dead");
			robot.die();
		}
	}
}
```

Agent Declarations – `.agent`

- *BeliefSets* and *Views* which the agent can use and refer to.
- *Events* (both internal and external) that the agent is prepared to handle.
- *Plans* that the agent can execute.
- *Events* the agent can post **internally** (to be handled by other plans).
- *Events* the agent can send **externally** to other agents.

Example Pilot Agent `Pilot.agent`

```
agent Pilot extends Agent
{
        #private data Position myPosition;
        #private data Position airportPosition;
        #private data FuelLevel fuelRemaining;

        #handles event TakeOffClearance clearedForTakeOff;
        #posts event StormDetected detectedStorm;
        #sends event RequestLanding landingRequest;

        #uses plan TakeOff;
        #uses plan Cruise;
        #uses plan LandAircraft;
        #uses plan AvoidStorm;

        #has capability EmergencyLanding elanding;

        Pilot(String name)
        {
                super(name);
                ...
    }
}
```

Events (.event)

- Two broad categories of events
  1. Normal events
  2. BDI events
- The key difference between normal events and BDI events is how an agent selects plans for execution.
- With normal events, the agent selects the first applicable plan instance for a given event and executes that plan instance only.
- The handling of BDI events is more complex and powerful. An agent can assemble a plan set for a given event, apply sophisticated heuristics for plan choice and act intelligently on plan failure.

Event Types in JACK

- Event
- MessageEvent
- BDIFactEvent
- BDIMessageEvent
- BDIGoalEvent
- InferenceGoalEvent
- PlanChoice

Example Event – `TakeOffClearance.event`

```
event TakeOffClearance extends Event
{
        String airTrafficController;
        String runway;

        #posted as clearedForTakeOff(String atc, String rwy)
        {
                airTrafficController = atc;
                runway = rwy;
        }
}
```

Plans (.plans)

- The `Plan` class describes a sequence of actions that an agent can take when an event occurs.
- Whenever an event is posted and an agent adopts a task to handle it, the first thing the agent does is try to find a plan to handle the event.
- Plans can be thought of as pages from a procedures manual. They describe, in explicit detail, exactly what an agent should do when a given event occurs.
- Each plan is capable of handling a single event.
- When an instance of a given event arises, the agent may execute one of the plans that declare they handle this event.

Discriminating Between Plans

- An agent may further discriminate between plans that declare they handle an event by determining whether a plan is relevant.
- It does this by executing the `relevant()` method of each plan.
- Once the relevant plan(s) have been identified, the agent determines which of these are applicable. It does this by executing the plans' `context()` method.
- The context method is a JACK Agent Language logical **expression** that can bind the values of the plan's logical members.
- For every possible set of bindings, a separate applicable instance of the plan is generated.
- When the agent has found all applicable instances of each relevant plan, it selects one of these to execute.

Example Plan – `TakeOff.plan`

```
plan TakeOff extends Plan
{
        #handles event TakeOffClearance clearance;
        #reads data FuelLevel fuelLevel;

        static boolean relevant(TakeOffClearance toc)
        {
                return true;
        }

        context()
        {
                fuelLevel.get(FULL-TANK);
        }

        body()
        {
                . . . // put reasoning statements here
        }
}
```

Reasoning Method Statements

- `@wait_for(parameters)`
- `@action(parameters) <body>`
- `@maintain(parameters)`
- `@post(parameters)`
- `@reply(parameters)`
- `@send(parameters)`
- `@subtask(parameters)`
- `@sleep(parameters)`
- `@achieve(parameters)`
- `@insist(parameters)`
- `@test(parameters)`
- `@determine(parameters)`
- `@parallel(parameters) <body>`

Beliefset Declarations – `.bel`

- Beliefsets are used in JACK to maintain an agent's beliefs about the world.
- An agent's beliefset can be stored as either an OpenWorld or a ClosedWorld class.
- The beliefset represents these beliefs in a first order, tuple-based relational model

Closed World Beliefs

True or False

Open World Beliefs

True, False or Unknown

Example Beliefset – BookData.bel

```
public beliefset BookData extends ClosedWorld
{
	#key field String title;
	#key field String author;
	#value field double price;

	#indexed query get(String t, String a, logical double p);
}
```

Jack Capabilities

- A way to group related events, plans, beliefs, views etc.
- A way to make modular agent code.
- Groups related agent abilities together.
- Elements can belong to more than one capability.
- Like an index of elements
  - not a partitioning
  - same element mentioned in many capabilities
- Module of elements that together implement a function (ability)
- Allows posting event in one capability, handling in another
- Allows data being hosted by one capability, used by another
- May be enabled / disabled dynamically
  - if disabled, then its plans are not applicable
  - no effect on data

Resources

- http://www.agent-software.com/
- http://agent-software.com/products/jack/documentation_and_instructi/