Lecture 8: Declarative Goals and Planning in BDI Agent-Oriented Programming Languages

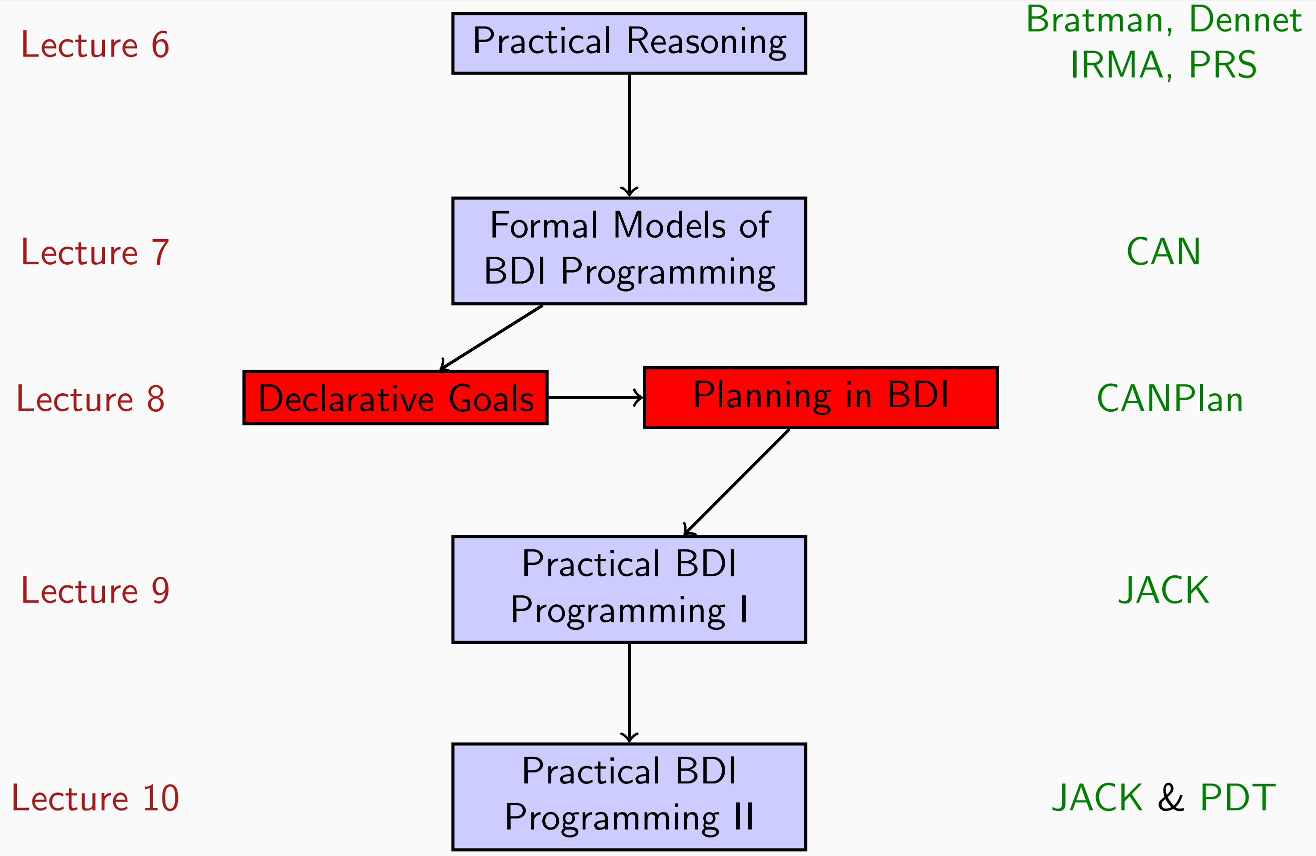
Autonomous Agents and Multiagent Systems
DIS, La Sapienza - PhD Course

Sebastian Sardina[1]

[1]Department of Computer Science and Information Technology
RMIT University
Melbourne, AUSTRALIA

November 29, 2007

Roadmap for Next Lectures



Review

In the previous lecture "*Formal Models of BDI Programming*" we have seen:

1. Basic concepts of BDI programming:
   - programming using mentalistic concepts such as beliefs, desires, capabilities, etc.
   - goal-oriented programming – via events;
   - implicit programming – via plan library & context conditions;
   - rational execution cycle: on-the-fly recombination of plans.
2. CAN formal BDI programming language:
   - captures the basic notions of BDI programming: rational executor;
   - formal operational semantics;
   - includes built-in failure handling.

This Lecture: Declarative Goals & Planning

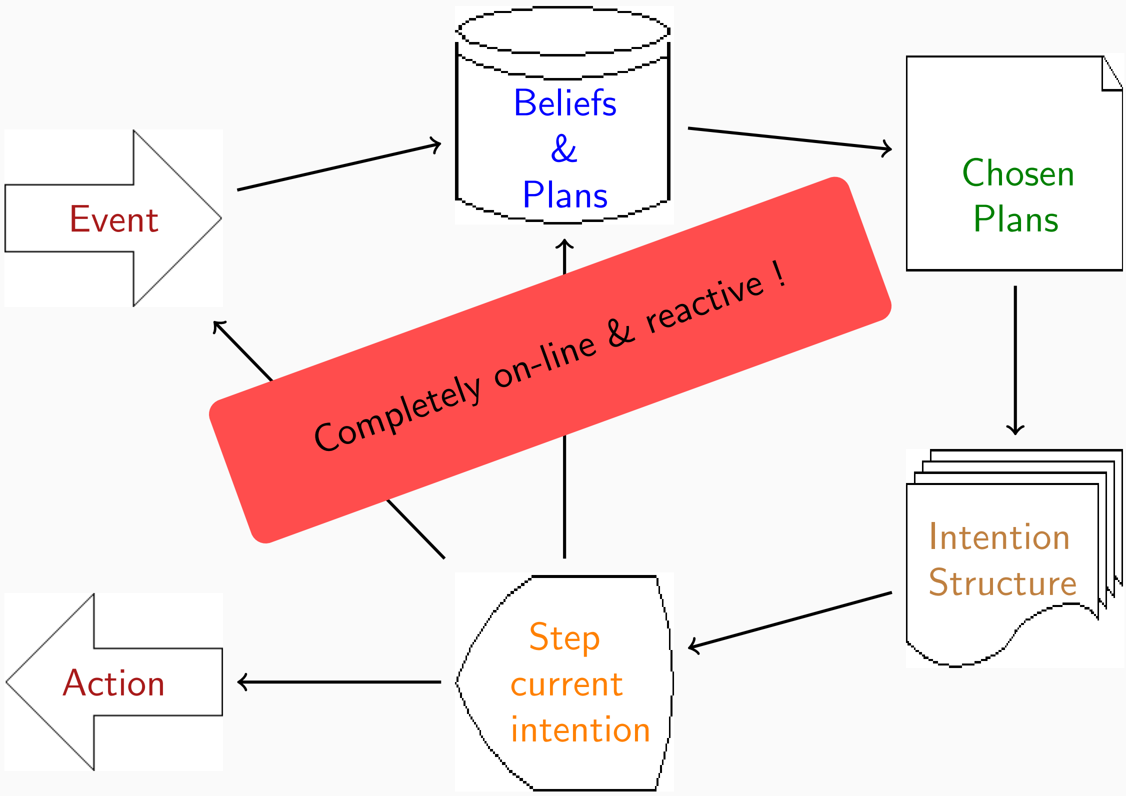
In the next lecture we show how to:

1. Accommodate declarative goals into CAN.
   - to provide an hybrid account of goals with both declarative & procedural aspects.
2. Accommodate hierarchical HTN-style planning into CAN.
   - to perform some "offline" lookahead reasoning within the whole online "reactive" execution scheme.

Outline

1. Review of CAN
2. Declarative Goals in CAN
3. CANPlan: CAN + HTN Planning
   - Motivation
   - HTN Planning
   - Plan Construct
4. Conclusions

The BDI Execution Cycle [Rao&Georgeff 92]



The CAN Language [Winikoff et al. 2002]

CAN: **C**onceptual **A**gent **N**otation

Can be seen as an extension of Rao's AgentSpeak.

A CAN agent is defined as $Agt = \langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \rangle$, where:

- $\mathcal{N}$ is the agent name.
- $\mathcal{B}$ is the belief base: current agent's knowledge.
- $\mathcal{A}$ is the sequence of actions executed so far.
- Π is a plan library containing plan rules $e : \psi \leftarrow P$:
  - e is the triggering event
  - ψ is the context condition
  - P is the plan-body
- Γ is the intention base: partially uninstantiated plan-bodies.

The CAN Language: Plans & Intentions

- Π is a plan library containing plan rules $e : \psi \leftarrow P$:

act	*primitive action*
$+b$	*belief addition*
$-b$	*belief deletion*
$?\phi$	*tests goal*
$!e$	*posting of achievement event goal*
$P_1; P_2$	*sequence*
$P_1 \| P_2$	*interleaved concurrency*

  Plus, the following system-auxiliary constructs:

$nil(\theta)$	*empty program with bindings*
$P_1 \rhd P_2$	*try P_1; else P_2*
$(\!| \psi_1 : P_1, \ldots, \psi_n : P_n |\!)$	*guarded plans*

- Γ is the intention base: set of partially uninstantiated plan-bodies.
  - E.g.: `(?phone(john,N);call(N);!talk) || !cook_dinner`

Semantics of CAN

The semantics of CAN is modularly defined in two levels:

1. Agent-level semantics: $\langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \rangle \Longrightarrow \langle \mathcal{N}, \Pi, \mathcal{B}', \mathcal{A}', \Gamma' \rangle$.
   - State that agent configuration $\langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \rangle$ may legally evolve to configuration $\langle \mathcal{N}, \Pi, \mathcal{B}', \mathcal{A}', \Gamma' \rangle$.
2. Intention-level semantics: $\langle \Pi, \mathcal{B}, \mathcal{A}, P \rangle \longrightarrow \langle \Pi, \mathcal{B}', \mathcal{A}', P' \rangle$.
   - State that intention configuration $\langle \Pi, \mathcal{B}, \mathcal{A}, P \rangle$ may legally evolve to configuration $\langle \Pi, \mathcal{B}', \mathcal{A}', P' \rangle$.

Legal transitions are characterized by a set of rules of the form:

$$\frac{\text{Set of conditions}}{C \longrightarrow C'}\ \mathit{RuleName}$$

Definition (BDI Agent Execution)

A BDI *execution* E of an agent $C_0 = \langle \mathcal{N}, \Pi, \mathcal{B}_0, \mathcal{A}_0, \Gamma_0 \rangle$ is a, possibly infinite, sequence of agent configurations $C_0 \cdot C_1 \cdot \ldots$ such that $C_i \Longrightarrow C_{i+1}$, for every $i \geq 0$. A *terminating* execution is a finite execution $C_0 \cdot \ldots \cdot C_n$ with $\Gamma_n = \{\}$.

Agent-Level Semantics

Assume $\langle \mathcal{B}, \mathcal{A}, P \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P' \rangle$ is given.

$$\frac{P \in \Gamma \quad \langle \mathcal{B}, \mathcal{A}, P \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P' \rangle}{\langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \rangle \Longrightarrow \langle \mathcal{N}, \Pi, \mathcal{B}', \mathcal{A}', (\Gamma \setminus \{P\}) \cup \{P'\} \rangle}\ Agt_{step}$$

Execute an active intention P.

$$\frac{e \text{ is a new external event}}{\langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \rangle \Longrightarrow \langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \cup \{!e\} \rangle}\ Agt_{event}$$

Assimilate an external event e.

$$\frac{P \in \Gamma \quad \langle \mathcal{B}, \mathcal{A}, P \rangle \not\longrightarrow}{\langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \rangle \Longrightarrow \langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \setminus \{P\} \rangle}\ Agt_{clean}$$

Remove an active intention P that is blocked.

Intention-Level Semantics

$$\frac{\mathcal{B} \models \phi\theta}{\langle \mathcal{B}, \mathcal{A}, ?\phi \rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, nil(\theta) \rangle} \; ? \qquad \frac{}{\langle \mathcal{B}, \mathcal{A}, act \rangle \longrightarrow \langle \mathcal{B}, \mathcal{A} \cdot act, nil(\emptyset) \rangle} \; do$$

$$\frac{}{\langle \mathcal{B}, \mathcal{A}, +b \rangle \longrightarrow \langle \mathcal{B} \cup \{b\}, \mathcal{A}, nil(\emptyset) \rangle} \; +b \qquad \frac{}{\langle \mathcal{B}, \mathcal{A}, -b \rangle \longrightarrow \langle \mathcal{B} \setminus \{b\}, \mathcal{A}, nil(\emptyset) \rangle} \; -b$$

$$\frac{\langle \mathcal{B}, \mathcal{A}, P_1 \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P_1' \rangle}{\langle \mathcal{B}, \mathcal{A}, P_1; P_2 \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P_1'; P_2 \rangle} \; Seq \qquad \frac{\langle \mathcal{B}, \mathcal{A}, P_1 \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P' \rangle}{\langle \mathcal{B}, \mathcal{A}, P_1 \parallel P_2 \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P' \parallel P_2 \rangle} \; \|_1$$

Intention-Level Semantics: Selection & Failure

$$!e \longrightarrow (\!| \psi_1 : P_1, \ldots, \psi_n : P_n |\!) \longrightarrow$$

$$P_i\theta_i \rhd (\!| \psi_1 : P_1, \ldots, \underline{\psi_i \wedge \vec{x} \neq \theta_i : P_i}, \ldots, \psi_n : P_n |\!) \xrightarrow{*}$$

$$\frac{\Delta = \{\psi_i\theta : P_i\theta \mid e' : \psi_i \leftarrow P_i \in \Pi \wedge \theta = \mathrm{mgu}(e, e')\}}{\langle \mathcal{B}, \mathcal{A}, !e \rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, (\!|\Delta|\!) \rangle} \; Event$$

$$\frac{\psi_i(\vec{x}) : P_i \in \Delta \quad \mathcal{B} \models \psi_i(\vec{x})\theta}{\langle \mathcal{B}, \mathcal{A}, (\!|\Delta|\!) \rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, P_i\theta \rhd (\!| \, (\Delta \setminus \{\psi_i(\vec{x}) : P_i\}) \cup \{\psi_i(\vec{x}) \wedge \vec{x} \neq \theta : P_i\} \, |\!) \rangle} \; Sel$$

$$\frac{\langle \mathcal{B}, \mathcal{A}, P_1 \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P' \rangle}{\langle \mathcal{B}, \mathcal{A}, P_1 \rhd P_2 \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P' \rhd P_2 \rangle} \; \rhd \qquad \frac{}{\langle \mathcal{B}, \mathcal{A}, (nil \rhd P_2) \rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, nil \rangle} \; \rhd_t$$

$$\frac{P_1 \neq nil \quad \langle \mathcal{B}, \mathcal{A}, P_1 \rangle \not\longrightarrow \quad \langle \mathcal{B}, \mathcal{A}, P_2 \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P_2' \rangle}{\langle \mathcal{B}, \mathcal{A}, P_1 \rhd P_2 \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P_2' \rangle} \; \rhd_f$$

Goals-to-be vs Goals-to-do

Contrast the following two kind of goals:

Goals-to-be Bring about a state of affairs.

- Used in automated planning and agent theory.
- Have a declarative flavor.
- E.g., achieve $BeAtLocation(20, 12)$.

Goals-to-do Complete a task or procedure.

- Used in hierarchical planning and high-level programming (ConGolog).
- Have a procedural flavor.
- E.g., complete $goToLocation(20, 12)$.

Agent goals in BDI programming as modelled with **events**!

Agent goals in BDI programming as events are pure **goals-to-do**!

Advantages of Declarative Goals

1. Help decouple plan execution and goal achievement.
   - Completing the plan vs achieving the goal.
2. Facilitate goal dynamics.
   - When is a (new) sub-goal required for another goal?
   - When should a goal be dropped? (e.g., it can never be obtained)
3. Facilitate plan failure handling
   - What can be done if a plan for a goal fails?
   - Is that the same as failing the goal itself?
4. Enable reasoning about goal and plan interaction
   - Does a plan *guarantee* a goal?
   - Can a plan *achieve* a goal?
   - Can a plan *preclude* a goal?
5. Enhance goal and plan communication
   - Communicate desires/requests vs communicate know-how.

Towards a Hybrid Goals

We want to keep the procedural advantages of event-goals:

- Built-in know-how available from domain experts.
- Reduced search space to cope online with dynamic environments.
- BDI execution engine with failure handling.

However, we would like to bring in the following features of declarative goals:

Declarative To facilitate potential reasoning and communication.

Persistent A rational agent should not abandon a goal without good reasons.

Unachieved A rational agent should not be pursuing goals that are already true.

Possible A rational agent should only pursue goals that are eventually possible to achieve.

Adding Declarative Goals to CAN

Normal event goals-to-do $!e$ are extended with declarative information:

$$\mathrm{Goal}(\phi_s, !e, \phi_f)$$

Achieve goal ϕ_s by using/posting event e; failing if ϕ_f becomes true.

1. ϕ_s is the success condition – when the goal assumed achieved.
2. e is the know-how of the goal – how the goal can be realized.
3. ϕ_f is the failure condition – when the goal is impossible or not needed.

Example

$\mathrm{Goal}(\neg Hungry, !eatFood, \neg foodAvailable)$.

OBS.: there may be many plans in the library for event *eatFood*.

Goal-programs Properties

Goal(ϕ_s, P, ϕ_f) Properties

1. If ϕ_s becomes true, terminate successfully. [unachieved]
2. If ϕ_f becomes true, terminate with failure. [possible]
3. If P ends successfully and ϕ_s is still *not* true, P is re-tried. [persistent]
4. If P fails and ϕ_s is still *not* true, P is re-tried. [persistent]

Example (Goal($\neg Hungry, !eatFood, \neg foodAvailable$))

1. If agent happens not to be hungry anymore (e.g., maybe feels sick), then the goal terminates successfully.
2. If another agent ate all food, then goal terminates with failure – agent is still hungry.
3. If agent ate some food (and thus completed !*eatFood*, but agent is still hungry, then more food will be eaten (!*eatFood* is re-tried).
4. If agent fails to eat pizza, then it should try eating something else! (!*eatFood* is re-tried)..

Goal-programs Semantics

$$\frac{\mathcal{B} \not\models \phi_s \vee \phi_f \quad \langle \mathcal{B}, \mathcal{A}, !e\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P\rangle}{\langle \mathcal{B}, \mathcal{A}, \mathrm{Goal}(\phi_s, !e, \phi_f)\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', \mathrm{Goal}(\phi_s, P \rhd P, \phi_f)\rangle}\ G_{Adopt}$$

$$\frac{\mathcal{B} \not\models \phi_s \vee \phi_f \quad \langle \mathcal{B}, \mathcal{A}, P_1\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P'\rangle}{\langle \mathcal{B}, \mathcal{A}, \mathrm{Goal}(\phi_s, P_1 \rhd P_2, \phi_f)\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', \mathrm{Goal}(\phi_s, P' \rhd P_2, \phi_f)\rangle}\ G_{Step}$$

$$\frac{\mathcal{B} \models \phi_s}{\langle \mathcal{B}, \mathcal{A}, \mathrm{Goal}(\phi_s, P, \phi_f)\rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, nil\rangle}\ G_s \qquad \frac{\mathcal{B} \models \phi_f}{\langle \mathcal{B}, \mathcal{A}, \mathrm{Goal}(\phi_s, P, \phi_f)\rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, ?false\rangle}\ G_f$$

$$\frac{P_1 \neq P_2 \quad \mathcal{B} \not\models \phi_s \vee \phi_f \quad \langle \mathcal{B}, \mathcal{A}, P_1\rangle \not\longrightarrow}{\langle \mathcal{B}, \mathcal{A}, \mathrm{Goal}(\phi_s, P_1 \rhd P_2, \phi_f)\rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, \mathrm{Goal}(\phi_s, P_2 \rhd P_2, \phi_f)\rangle}\ G_{Restart}$$

Goal is adopted – $P = (\!|\psi_1 : P_1, \ldots, \psi_n : P_n|\!)$ encodes all plans for e.
Plan is executed – only current strategy is updated!
If ϕ_s holds, whole program terminates successfully
If ϕ_f holds, whole program terminates with failure.
If plan is finished or blocked, original options are re-instantiated.

Properties of Goal($\phi_s, !e, \phi_f$)

1. Can be used by the BDI programmer to specify both declarative & procedural aspects of goals.
   - The whole original BDI reactive-online approach is maintained!;
2. Agent never adopts a goal that is already achieved or deemed impossible.
3. Agent never drops an adopted goal unless achieved or impossible.
   - Agent keeps trying and trying...
4. Agent never pursues goals which are achieved or impossible.
   - Will be terminated successfully or with failure.
5. Agent never pursues a goal that serves as a sub-goal for some higher-level motivating goal.
   - All sub-goals are removed when motivating goal is removed.

Generating Top-Level Goals

How can the agent self-generate new top-level goals?

Equip our agents with a *motivation library* $\mathcal{M}$ to accommodate the consideration of new goals in a proactive manner—the intrinsic agent's motivations or desires. Library $\mathcal{M}$ consists of rules of the form:

$$\psi \leadsto \mathrm{Goal}(\phi_s, !e, \phi_f),$$

If the agent comes to believe ψ, she should *consider adopting* the declarative goal-program Goal($\phi_s, !e, \phi_f$).

Example

$RoomDirty \wedge \neg Busy \leadsto \mathrm{Goal}(\neg RoomDirty, !clean, HasWork)$.

Generating Top-Level Goals (cont.)

To accommodate the motivational library we need to:

1. Extend agent-configurations to tuples of the form $\langle \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle$
2. Add a new agent-level semantic rule:

$$\frac{\psi \rightsquigarrow \mathsf{Goal}(\phi_s, !e, \phi_f) \in \mathcal{M} \quad \mathcal{B} \models \psi \quad \mathsf{Goal}(\phi_s, P, \phi_f) \notin \Gamma}{\langle \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \Longrightarrow \langle \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \cup \{\mathsf{Goal}(\phi_s, !e, \phi_f)\} \rangle} \; Agt_{motiv}$$

Observe that:

- Defined at the agent-level as it modifies the intention base.
- A completely new intention is created – new focus of attention.
- This mechanism has also been called elsewhere agent desires (3APL) or automatic events (JACK).

Motivation

BDI-style agent-oriented programming is a novel approach to programming complex large systems in highly dynamic environments.

These systems are very open to changes in the environment:

- reactive – observe and respond to the environment;
- flexible – multiple options to achieve goals;
- robust – retry alternatives upon failure; commitment.

However, they usually lack any lookahead planning capabilities.

- Hypothetical reasoning about the future.

HTN planning is a well-known practical approach to planning.

> Incorporate HTN planning into current BDI programming frameworks with:
> (i) a clear semantics; (ii) a direct implementation.

Basic Architecture of CAN

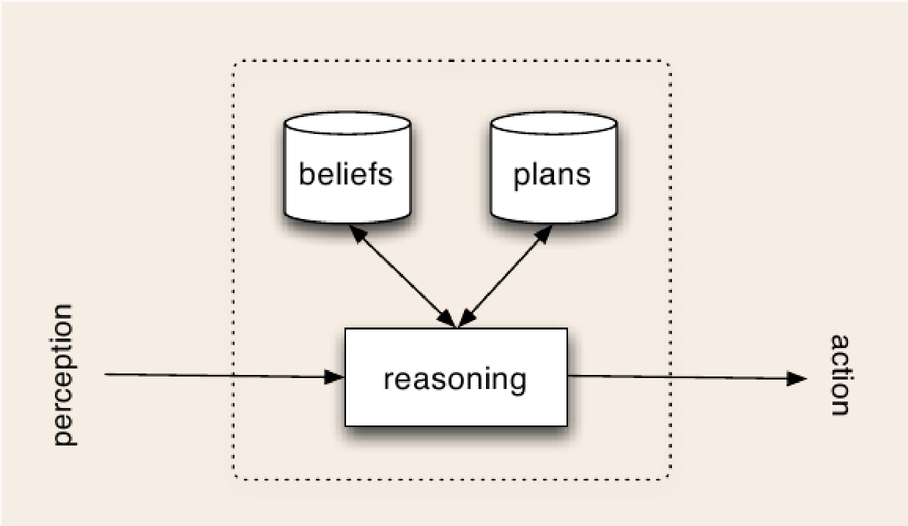
Core Aspects of BDI Programming

Beliefs: information about the world.

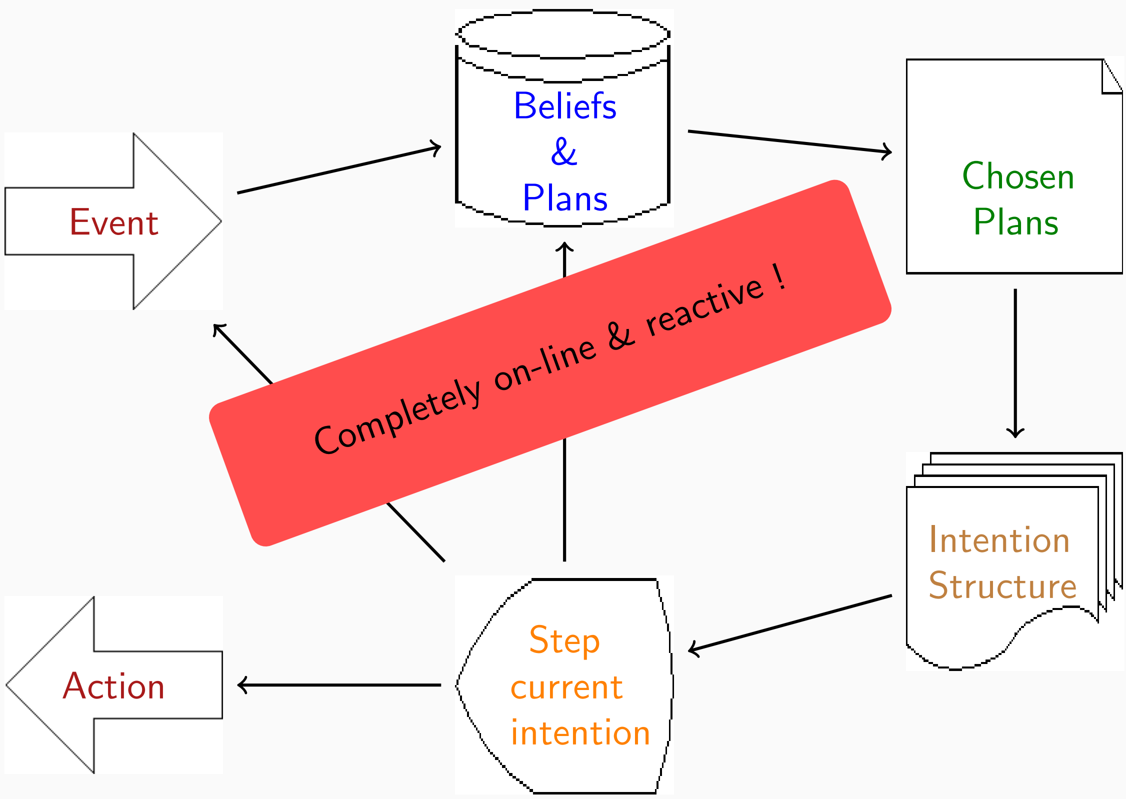
Events: goals/desires to resolve; internal or external.

Plan library: recipes for handling goals-events.

Intentions: partially uninstantiated programs with commitment.



The BDI Execution Cycle [Rao&Georgeff 92]



Key Points of BDI Programming

$\boxed{\text{BDI Progr.} = \text{Implicit Goal-based Programming} + \text{Rat. Online Executor}}$

- Flexible and responsible to the environment: "reactive planning." $\surd$
- Well suited for soft real-time reasoning and control. $\surd$
- Relies on context sensitive subgoal expansion: "act as you go." $\surd$
- Leave for as late as possible the choice of which plans to commit to as the chosen course of action to achieve (sub)goals. $\surd$
- Modular and incremental programming. $\surd$
- Nondeterminism on choosing plans and bindings. $\surd$
- **BUT: No mechanism for doing lookahead for solving choices!** $\otimes$
  - Generally programmed *explicitly* by the BDI programmer.

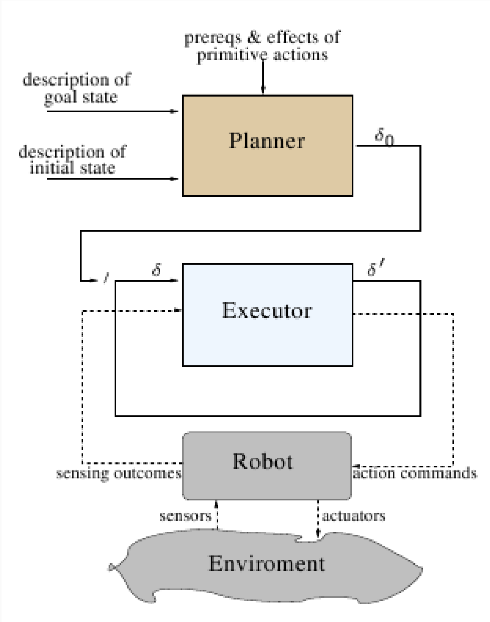
Why do we want planning?

Some reasons for lookahead capabilities for BDI agents:

- Important resources may be used in taking actions that do not lead to successful outcome (e.g., fuel).
- Actions may sometimes not be reversible (e.g., break glass).
- Execution of actions may take substantial more time than just "thinking" ahead (e.g., traveling to a distant location).
- Actions' execution may include undesirable side-effects (e.g., email notifications).

Classical Planning [SRI Shakey "The Robot" '70]

- Planning from first-principles.
- Idea: achieve a goal by performing actions.
- Goals-to-be: *achieve* ϕ $(At(aiport))$.
- Input to planner:
  - action descriptions;
  - initial state;
  - goal state.
- Output of planner:
  - a sequence of actions that achieves the goal.

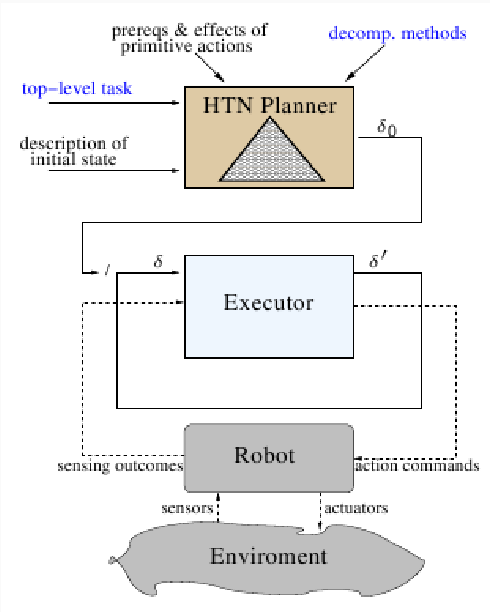


Obs.: planner has no domain information on how the goal state may be reached (e.g., common ways to go to the airport).

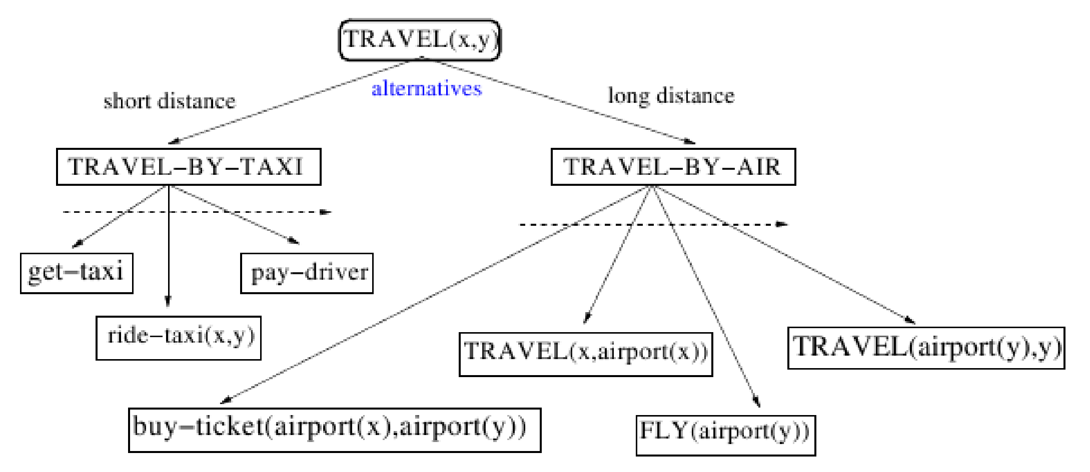
- planning algorithms don't take advantage of **all** problem structure.

Hierarchical Task Networks (HTN) Planning

- Idea: accomplish some set of tasks, rather than to achieve a goal.
- Goals-to-do: $travelTo(loc)$.
- Includes domain procedural knowledge.
- Input to planner:
  - actions/operator descriptions;
  - initial state;
  - top-level task;
  - decomposition methods: how to decompose a task into a set of sub-tasks.
- Output of planner:
  - a sequence of actions that achieves the goal.
- Planning:
  - Decompose tasks by applying methods until all tasks in network are primitive actions.



HTN Planning: An Example



State: set of atoms: $At(loc)$.

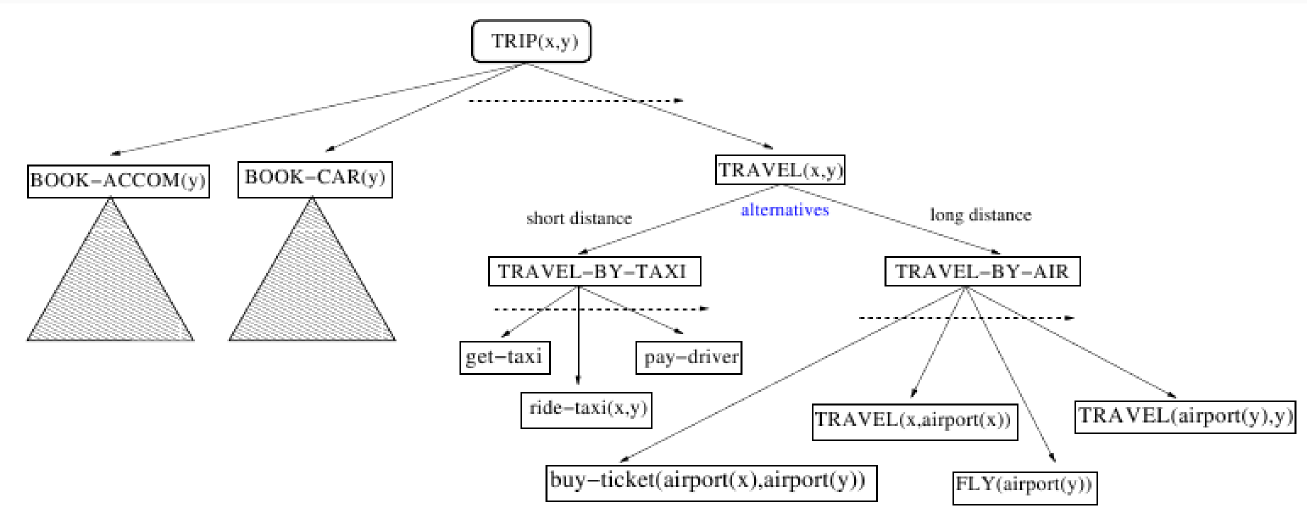
Tasks: primitive or compound.

Task Network: set of tasks T + order/state constraints ϕ.

Method: a way to solve a compound task e using a network d.

Plan: a sequence of primitive tasks.

HTN Planning: An Example (cont.)



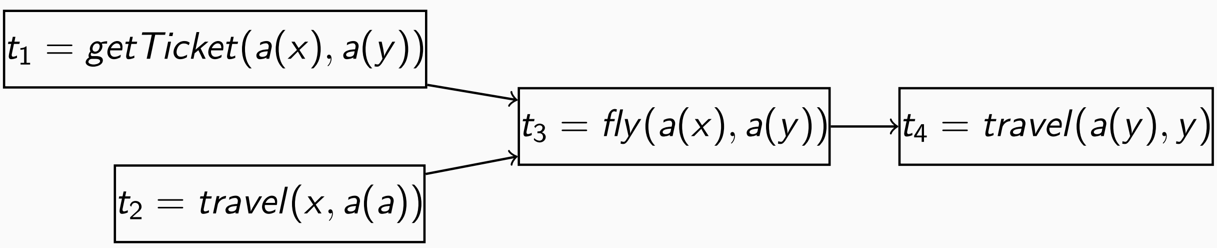
Key Points of HTN Planning

- goals-to-do instead of goals-to-be: $goToLocation(YYZ)$ vs $At(YYZ)$.
- Decomposition of high-level tasks.
- User provides (procedural) knowledge capturing lots of useful domain information:
  - Methods correspond to "recipes for accomplishing things.";
  - **If we already have an idea of how to solve a problem, we might as well use it in our planner!**
- Well understood semantics and implementations.
  - Operational and model-theoretic semantics.
  - Implementations: SHOP, SHOP2, UMCP, JSHOP, etc.
- HTN planners have been some of the most successful & commonly used.
- Some similarities with "programming".
- Subsumes first-principle "STRIP" planning.

Components of HTN Planning

- State: set of atoms + CWA
- Task: a common goal that the agent may need to achieve.
  - Primitive task a: with precondition and effects: $ride(x, y)$, $payTaxi$, etc.
  - High-level task e: cannot be directly executed: $travel(x, y)$
- Task Network $d = (T, \phi)$: set of tasks T + order/state constraints ϕ.

  $d_{travel} = (\ \{t_4 = travel(a(y), y), t_2 = travel(x, a(x)), t_3 = fly(a(x), a(y)),$
  $t_1 = getTicket(a(x), a(y))\},\ (t_1 \prec t_3 \wedge t_2 \prec t_3 \wedge t_3 \prec t_4)\ )$



- Method $m = [e, \psi, d]$: "recipe" to solve task e.

  $[travel(x, y), LongDistance(x, y), d_{travel}]$

Formal Specification of HTN-Planning [Erol et al 94]

- An HTN planning domain is a pair $\mathcal{D} = (\Pi, \Lambda)$:
  - method library Π;
  - STRIP-like (add and delete lists) action description Λ.
- An HTN problem is a triple $\mathbf{P} = \langle d, \mathcal{B}, \mathcal{D} \rangle$:
  - solve task network d in state $\mathcal{B}$ using domain $\mathcal{D}$.
- A plan σ is a sequence of primitive tasks.

$$getCab \cdot ride(rome, a(rome)) \cdot checkIn \cdot walk(gate) \ldots$$

- An HTN planning solution for problem $\mathbf{P}$ is a plan σ that stands for a *full successful decomposition* of task network d.
  - $sol(d, \mathcal{B}, \mathcal{D})$: set of all plans that solve d.

The CAN Language [Winikioff et al. 2002]

A CAN agent is defined as $Agt = \langle \mathcal{N}, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle$, where:

- $\mathcal{N}$ is the agent name.
- $\mathcal{B}$ is the belief base: current agent's knowledge.
- $\mathcal{A}$ is the sequence of actions executed so far.
- Π is a plan library containing plan rules $e : \psi \leftarrow P$:
  - e is the triggering event; ψ is the context condition; P is the plan-body:
    $P ::= nil \mid act \mid ?\phi \mid +b \mid -b \mid P_1; P_2 \mid P_1 \| P_2 \mid \underline{!e} \mid \underline{\mathrm{Goal}(\phi_s, P_1, \phi_f)}$
- $\mathcal{M}$ is the motivation library.
- Γ is the intention base: partially uninstantiated plan-bodies.

The CAN Language

However, CAN does not include any planning mechanism.

The objective: add lookahead capabilities to CAN in a way that:

1. the original semantics can be extended;
2. an implementation can be easily produced.

We will extend CAN with:

1. A library of STRIP-like action descriptions Λ (precondition & effects);

$$a : \psi \leftarrow \Phi^-; \Phi^+ \in \Lambda$$

$$pickUp(x) : Clear(x) \leftarrow \{Clear(x)\}^-; \{Holding(x)\}^+ \in \Lambda$$

2. A new programming construct Plan(P).
   - Plan(P): look/check for a full solution for BDI plan P.

observe similarity with Σ in IndiGolog!

BDI Systems and HTN Planners: Similarities

BDI Systems	HTN Systems
belief base	state
plan library	method library
event	high-level task
plan rule	method
plan-body/program	network task
plan rule context	method precondition
action	primitive task
test $?p$ in plan-body	state constraints
sequence in plan-body	ordering constraint $\prec$
parallelism in plan-body	no ordering constraint

$$[travel(x, y), LongDist(x, y), d_{travel}]$$

$$d_{travel} = (\ \{t_4 = travel(a(y), y), t_2 = travel(x, a(x)), t_3 = fly(a(x), a(y)), t_1 = getTicket(a(x), a(y))\},\ (t_1 \prec t_3 \wedge t_2 \prec t_3 \wedge t_3 \prec t_4)\ )$$

$$travel(x, y) :\ LongDist(x, y) \leftarrow (!getTicket(a(x), a(y)) \parallel !travel(x, a(x)));\ !fly(a(x), a(y));\ !travel(a(y), y)$$

The CANPlan Language: Operational Semantics

Recall semantics of CAN is modularly defined in two levels:

1. Agent-level semantics: $\langle \mathcal{N}, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \Longrightarrow \langle \mathcal{N}, \Pi, \mathcal{M}, \mathcal{B}', \mathcal{A}', \Gamma' \rangle$.
2. Intention-level semantics: $\langle \Pi, \mathcal{B}, \mathcal{A}, P \rangle \longrightarrow \langle \Pi, \mathcal{B}', \mathcal{A}', P' \rangle$.

We now use 2 labelled intention-level transitions:

- $\xrightarrow{\mathsf{bdi}}$: transitions for the BDI execution cycle – all rules as before.
- $\xrightarrow{\mathsf{plan}}$: transitions when planning – almost all the rules as before.

We also add the STRIP-like action library Λ:

1. Agent-level semantics:
   $\langle \mathcal{N}, \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \Longrightarrow \langle \mathcal{N}, \Lambda, \Pi, \mathcal{M}, \mathcal{B}', \mathcal{A}', \Gamma' \rangle$.
2. Intention-level semantics:

   $\langle \Lambda, \Pi, \mathcal{B}, \mathcal{A}, P \rangle \xrightarrow{\mathsf{bdi}} \langle \Lambda, \Pi, \mathcal{B}', \mathcal{A}', P' \rangle$ and
   $\langle \Lambda, \Pi, \mathcal{B}, \mathcal{A}, P \rangle \xrightarrow{\mathsf{plan}} \langle \Lambda, \Pi, \mathcal{B}', \mathcal{A}', P' \rangle$.

The CANPlan Language: Operational Semantics

New construct Plan(P): perform off-line lookahead on plan P.

$$\frac{\langle \mathcal{B}, \mathcal{A}, P \rangle \xrightarrow{\mathsf{plan}} \langle \mathcal{B}', \mathcal{A}', P' \rangle \quad \langle \mathcal{B}', \mathcal{A}', P' \rangle \xrightarrow{\mathsf{plan}}{}^{*} \langle \mathcal{B}'', \mathcal{A}'', nil \rangle}{\langle \mathcal{B}, \mathcal{A}, \mathsf{Plan}(P) \rangle \xrightarrow{\mathsf{bdi}} \langle \mathcal{B}', \mathcal{A}', \mathsf{Plan}(P') \rangle} \; \mathsf{Plan}$$

Inspired by the semantics of $\Sigma(\delta)$ in IndiGolog
(De Giacomo & Levesque 99)

The CANPlan Language: Operational Semantics (cont.)

We also need to do the following three changes:

1. account for actions' preconditions and effects;

$$\frac{a:\ \psi \leftarrow \Phi^-;\Phi^+ \in \Lambda \quad a\theta = act \quad \mathcal{B} \models \psi\theta}{\langle \mathcal{B}, \mathcal{A}, act\rangle \longrightarrow \langle (\mathcal{B} \setminus \Phi^-) \cup \Phi^+, \mathcal{A} \cdot act, nil\rangle} \ do$$

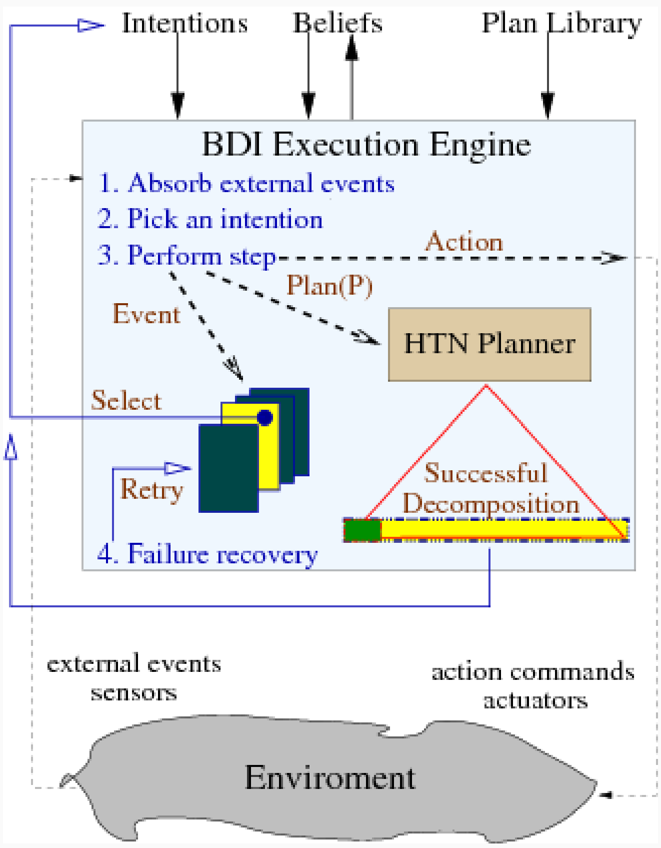
2. allow retry of alternatives *only* in the BDI execution cycle;

$$\frac{P_1 \neq nil \quad \langle \mathcal{B}, \mathcal{A}, P_1\rangle \not\longrightarrow \quad \langle \mathcal{B}, \mathcal{A}, P_2\rangle \longrightarrow \xrightarrow{\text{bdi}} \langle \mathcal{B}', \mathcal{A}', P_2'\rangle}{\langle \mathcal{B}, \mathcal{A}, P_1 \rhd P_2\rangle \longrightarrow \xrightarrow{\text{bdi}} \langle \mathcal{B}', \mathcal{A}', P_2'\rangle} \ \rhd_f$$

3. perform agent steps based on bdi-type basic transitions.

$$\frac{P \in \Gamma \quad \langle \mathcal{B}, \mathcal{A}, P\rangle \longrightarrow \xrightarrow{\text{bdi}} \langle \mathcal{B}', \mathcal{A}', P'\rangle}{\langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma\rangle \Longrightarrow \langle \mathcal{N}, \Pi, \mathcal{B}', \mathcal{A}', (\Gamma \setminus \{P\}) \cup \{P'\}\rangle} \ Agt_{step}$$

CANPlan Architecture



Some Remarks About Plan

1. Plan is built-in within the BDI execution cycle.
   - on demand look-ahead planning.
2. Plan uses exactly the same domain knowledge as given to the BDI system.
   - no separate knowledge needs to be given!
   - because BDI and HTN system work on the same kind of information...
3. Plan's semantics is defined "in terms" of the BDI cycle.
   - defined in terms of the same semantic rules;
   - because BDI and HTN system work in a similar way...
4. But failure-handling is left to the BDI cycle only!

Main Results About Plan(P) (cont.)

Theorem

An intention Plan(P) *never fails in an environment-free execution.*

Theorem

All executions of Plan(P) *correspond to HTN solutions of P w.r.t. the agent's plan-library Π (and vice-versa).*

Basis for an implementation: JACK+ JSHOP (Lavindra & Padgham 05):

- JACK is a BDI-style programming language similar to CAN;
- JSHOP is an Java-based HTN planner implementation.

(no concurrency $\|$ and no goal-programs Goal, yet)

Planning for Declarative Goals: Use P for achieving ϕ_s

(A) P is used towards the eventual satisfaction of goal ϕ_s.

(B) Allow partial executions of P achieving ϕ_s.

(C) Commitment on the goal ϕ_s.

(D) Goal dropping mechanism.

(E) P may be solved partially.

ALTERNATIVES	A	B	C	D	E
$\text{Plan}(P; ?\phi_s)$	$\surd$				
$\text{Plan}(\text{Goal}(\phi_s, P, \phi_f))$	$\surd$	$\surd$		$\surd$	
$\text{Goal}(\phi_s, \text{Plan}(P), \phi_f)$		$\surd$	$\surd$	$\surd$	
$\text{Goal}(\phi_s, \text{Plan}(P; ?\phi_s), \phi_f)$	$\surd$	$\surd$	$\surd$	$\surd$	
$\text{Goal}(\phi_s, \text{Plan}(\text{Goal}(\phi_s, P, \phi_f)), \phi_f)$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$

$$\text{Plan}(\phi_s, P, \phi_f) \stackrel{\text{def}}{=} \text{Goal}(\phi_s, \text{Plan}(\text{Goal}(\phi_s, P, \phi_f)), \phi_f)$$

Other Open Issues

1. Plan monitoring & replanning:
   - What to do when a planning solution fails?
2. Planning in the context of other intentions.
   - Reasoning about negative & positive interactions.
3. Planning up to some level of abstraction.
   - Leave details to the BDI execution engine.
4. Using first-principles planning for learning new plans:
   - When no relevant plan is available $\Longrightarrow$ classical planning.
   - Add new plan to the BDI plan library Π.
5. ...

Review

In this lecture we have shown how to

1. Accommodate **declarative goals** in CAN:
   - motivated the need for declarative goals (e.g., decoupling goal achievement vs plan termination).
   - incorporated new construct $\mathsf{Goal}(\phi_s, P, \phi_f)$ into the language.
2. Accommodate **on-demand planning** in CAN:
   - motivated the need for planning in BDI systems;
   - reviewed HTN-planning: off-line decomposition of tasks;
   - incorporated new construct $\mathsf{Plan}(P)$ to the language;

Next Lecture

JACK Agent-Oriented Programming Language

Declarative Goals

Michael Winikoff, Lin Padgham, James Harland, and John Thangarajah.
Declarative & procedural goals in intelligent agent systems.
In *Proceedings of the KR-02*, 2002.

Birna van Riemsdijk, Mehdi Dastani, Frank Dignum and John-Jules Ch. Meyer
Dynamics of Declarative Goals in Agent Programming.
In *Proceedings of the DALT-05*, 2005.

Jomi H ubner, Rafael, Bordini and and Michael Wooldridge.
Programming declarative goals using plan patterns.
In *Proceedings of the DALT-06*, 2006.

Frank S. de Boer, Koen V. Hindriks, Wiebe van der Hoek, and John-Jules Ch. Meyer.
A Verification Framework for Agent Programming with Dec. Goals.
Journal of Applied Logic, 5(2):277–302, 2007.

Planning in BDI Systems

Kutluhan Erol, James Hendler, and Dana S. Nau.
Complexity Results for HTN Planning
Annals of Math in Artificial Intelligence, 18(1):69-93, 1996

D. Nau, Y. Cao, A. Lotem, and H. Muñoz Avila.
Shop: Simple hierarchical ordered planner.
In *Proceedings of IJCAI-99*, pages 968–973, 1999.

Olivier Despouys and Francois Felix Ingrand.
Propice-plan: Toward a unified framework for planning and execution.
In *Proc. of European Conference on Planning*, pages 278–293, 1999.

Jürgen Dix, Héctor Muñoz-Avila, Dana S. Nau, and Lingling Zhang.
IMPACTing SHOP: Putting an AI planner into a multi-agent env.
Annals of Mathematics and Artificial Intelligence, 37(4):381–407, 2003.

Sebastian Sardina, Lavindra P de Silva, and Lin Padgham.
Hierarchical planning in BDI agent programming languages: A formal approach.
In *Proceedings of AAMAS-06*, pages 1001–1008, 2006.