Implementing the Proactive Behavior in Intelligent Agents: The Role of Abstraction

<u>Eloisa Vargiu</u>

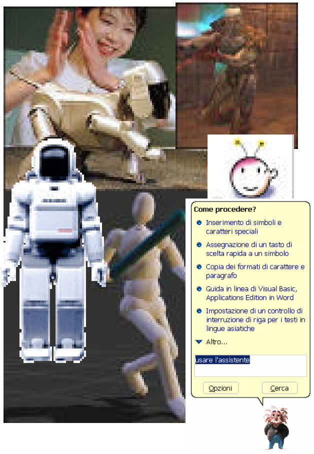
Dipartimento di Ingegneria Elettrica ed Elettronica

Università degli Studi di Cagliari

Outline

- Introduction
- Agents architectures
- Planning in classical domains
- Planning in real domains
- Conclusions

Introduction



Outline

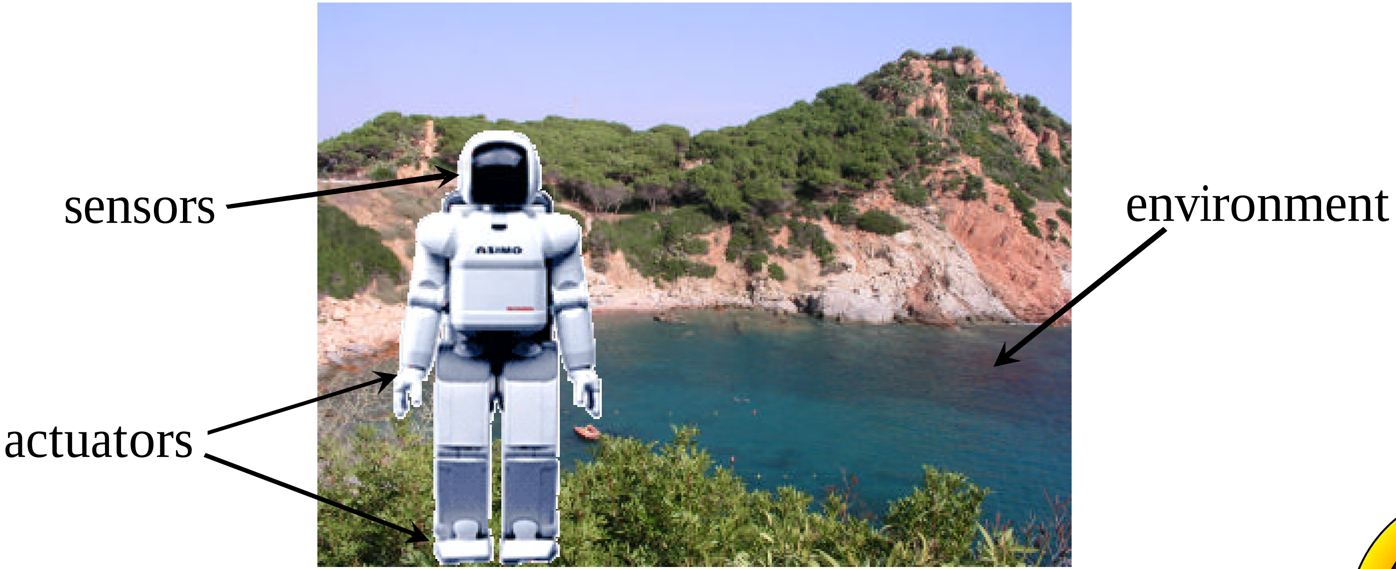
- **Introduction**
  - **Agent Definition**
  - **Environment Properties**
- Agents architectures
- Planning in classical domains
- Planning in real domains
- Conclusions

Notion of Agency [WJ94]

- **Weak notion**
  - An agent has the following properties:
    - Autonomy
    - Flexibility
- **A stronger notion**
  - An agent, in addition, is either conceptualized or implemented using concepts that are more usually applied to humans
- **Other attributes**
  - Various other attributes are sometimes discussed in the context of agency
    - Mobility
    - Veracity
    - Benevolence
    - Rationality

Autonomy

- By autonomy, we mean the capability of:
  - Acting independently
  - Exhibiting control over its internal state



Flexibility

- By flexibility, we mean
  - Reactivity
  - Proactiveness
  - Social ability

Flexibility

- **By flexibility, we mean**
  - **Reactivity**

Flexibility

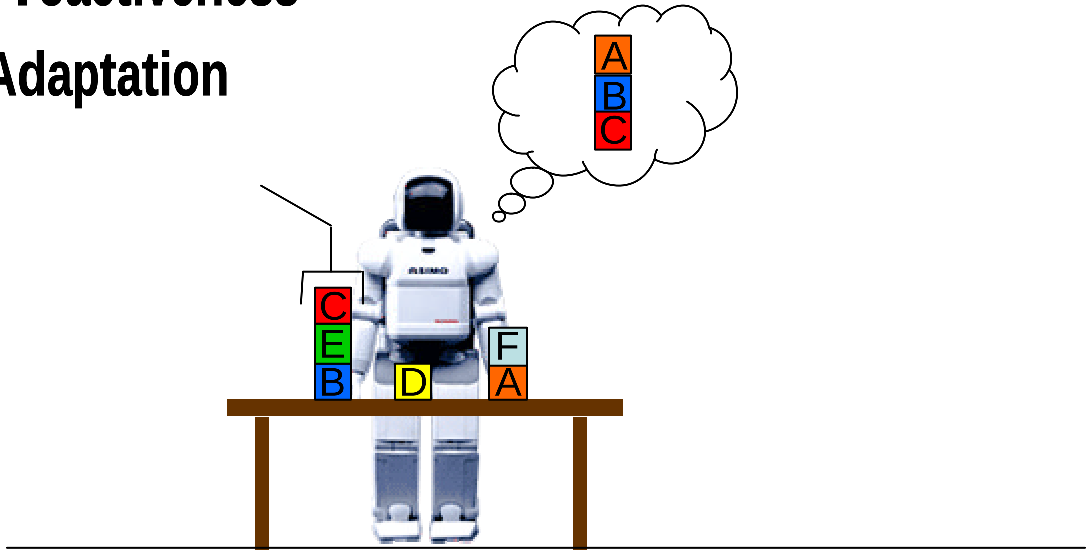
- By flexiblilty, we mean
  - Reactivity
  - Proactiveness

Flexibility

- By flexiblilty, we mean
  - Reactivity
  - Proactiveness
  - Social ability

The Ideal Agent

- The ideal agent, in our opinion, should exhibit:
  - Proactiveness
  - Adaptation



Environment Properties

- Russel and Norvig [RN04] suggest the following classification of environment properties:
  - Accessible vs inaccessible
  - Deterministic vs non-deterministic
  - Episodic vs non-episodic
  - Static vs dynamic
  - Discrete vs continuous

Environment Properties

- Russel and Norvig [RN04] suggest the following classification of environment properties:
  - Accessible vs inaccessible
  - Deterministic vs non-
  - Episodic vs non-epis
  - Static vs dynamic
  - Discrete vs continuou

Accessible environment:
- complete;
- accurate;
- up-to-date

information about the environment

Environment Properties

- Russel and Norvig [RN04] suggest the following classification of environment properties:
  - Accessible vs inaccessible
  - Deterministic vs non-deterministic
  - Episodic vs non-episodic
  - Static vs dynamic
  - Discrete vs continuous

Deterministic environment:
- any action has a single effect;
- no uncertainty about the state.

Environment Properties

- **Russel and Norvig [RN04] suggest the following classification of environment properties:**
  - **Accessible vs inaccessible**
  - **Deterministic vs non-deterministic**
  - **Episodic vs non-episodic**
  - **Static vs dynamic**
  - **Discrete vs continuou**

***Episodic environment*:**

the performance depends on a number of discrete episodes.

Environment Properties

- Russel and Norvig [RN04] suggest the following classification of environment properties:
  - Accessible vs inaccessible
  - Deterministic vs non-deterministic
  - Episodic vs non-episodic
  - Static vs dynamic
  - Discrete vs continuous

Static environment:

remains unchanged except by the performance of agent actions.

Environment Properties

- **Russel and Norvig [RN04] suggest the following classification of environment properties:**
  - **Accessible vs inaccessible**
  - **Deterministic vs non-deterministic**
  - **Episodic vs non-episodic**
  - **Static vs dynamic**
  - **Discrete vs continuous**

***Discrete environment*:**

fixed, finite number of actions and percepts in it.

Part I

Agents Architectures

Outline

- Introduction
- Agents architectures
  - Definition
- Planning in classical domains
- Planning in real domains
- Conclusions

Agent Architectures

- **«An agent architecture is a map of the internals of an agent»** [W99]
- **«It specifies how the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact»** [M91]

Agent Architectures

- **Reactive Architectures**
  - **Brook's Architecture** [B86]
- **Belief-Desire-Intention Architectures** [BIP88] [GL87]
- **Hybrid Architectures**
  - **Layered (horizontal, vertical) Architecture** [F92] [M97] [ACV01]

Reactive Architectures

- **Brooks' theses:**
  - **Intelligent behavior can be generated without an explicit AI representation**
  - **Intelligent behavior can be generated without an explicit AI abstract reasoning**
  - **Intelligence is an emergent property of certain complex systems**

Reactive Architectures

- **Brooks' ideas:**
  - **Situatedness and embodiment: "real" intelligence is situated in the world, not in disembodied systems**
  - **Intelligence and emergence: "intelligence" behavior arises as a result of agent's interaction with its environment**

Reactive Architectures

- **The Brooks' *subsumption architecture***
  - **hierarchy of task-accomplishing *behaviors***
  - **each behavior is a rather simple *rule-like structure***
  - **each behavior competes with others to *exercise control* over the agent**
  - ***lower layers* represent more primitive kinds of behavior**

BDI Architectures

- **These architectures have their roots in the philosophical tradition of understanding practical reasoning:**
  - **<u>what</u> goals we want to achieve (*deliberation*)**
  - **<u>how</u> we are going to achieve these goals (*means-ends reasoning*)**

BDI Architectures

- **The process of practical reasoning in a BDI agents can be summarized in seven components:**
  - **Belief**
  - **Belief Revision Function**
  - **Option Generation Function**
  - **Current Options**
  - **Filter Function**
  - **Current Intentions**
  - **Action Selection function**

BDI Architectures

- The process of practical reasoning in a BDI agents can be summarized in seven components:
  - Belief
  - Belief Revision Function
  - Option Generation Function
  - Current Options
  - Filter Function
  - Current Intentions
  - Action Selection function

information about the current environment

BDI Architectures

- **The process of practical reasoning in a BDI agents can be summarized in seven components:**
  - **Belief**
  - **Belief Revision Function**
  - **Option Generation Function**
  - **Current Options**
  - **Filter Function**
  - **Current Intentions**
  - **Action Selection function**

determines a new set of beliefs

BDI Architectures

- The process of practical reasoning in a BDI agents can be summarized in seven components:
  - Belief
  - Belief Revision Function
  - Option Generation Function
  - Current Options
  - Filter Function
  - Current Intentions
  - Action Selection function

agent's desires on the basis of agent's intentions

BDI Architectures

- The process of practical reasoning in a BDI agents can be summarized in seven components:
  - Belief
  - Belief Revision Function
  - Option Generation Function
  - Current Options
  - Filter Function
  - Current Intentions
  - Action Selection function

possible courses of actions

BDI Architectures

- **The process of practical reasoning in a BDI agents can be summarized in seven components:**
  - **Belief**
  - **Belief Revision Function**
  - **Option Generation Function**
  - **Current Options**
  - **Filter Function** — **the agent's deliberation process**
  - **Current Intentions**
  - **Action Selection function**

BDI Architectures

- The process of practical reasoning in a BDI agents can be summarized in seven components:
  - Belief
  - Belief Revision Function
  - Option Generation Function
  - Current Options
  - Filter Function
  - Current Intentions
  - Action Selection function

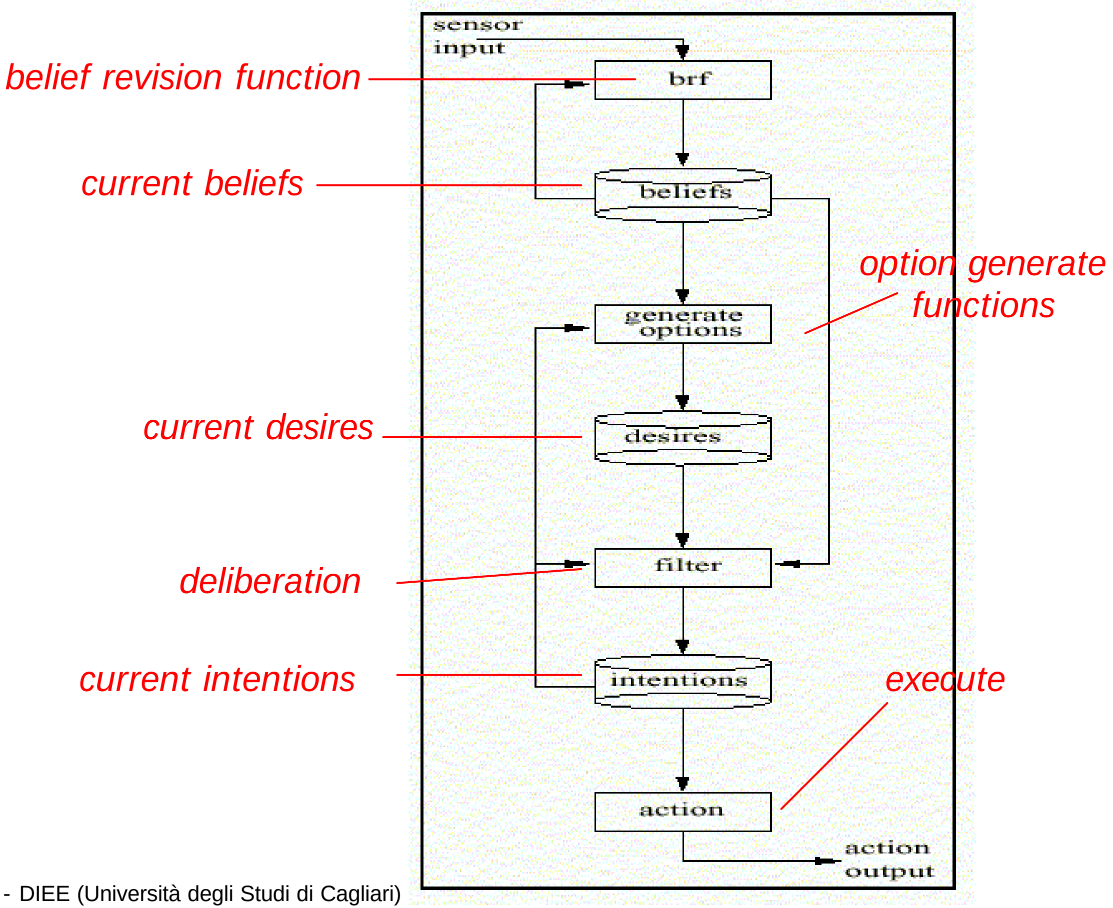
the agent's current focus

BDI Architectures

- The process of practical reasoning in a BDI agents can be summarized in seven components:
  - Belief
  - Belief Revision Function
  - Option Generation Function
  - Current Options
  - Filter Function
  - Current Intentions
  - Action Selection function

an action to perform on the basis of current intentions

BDI Architectures



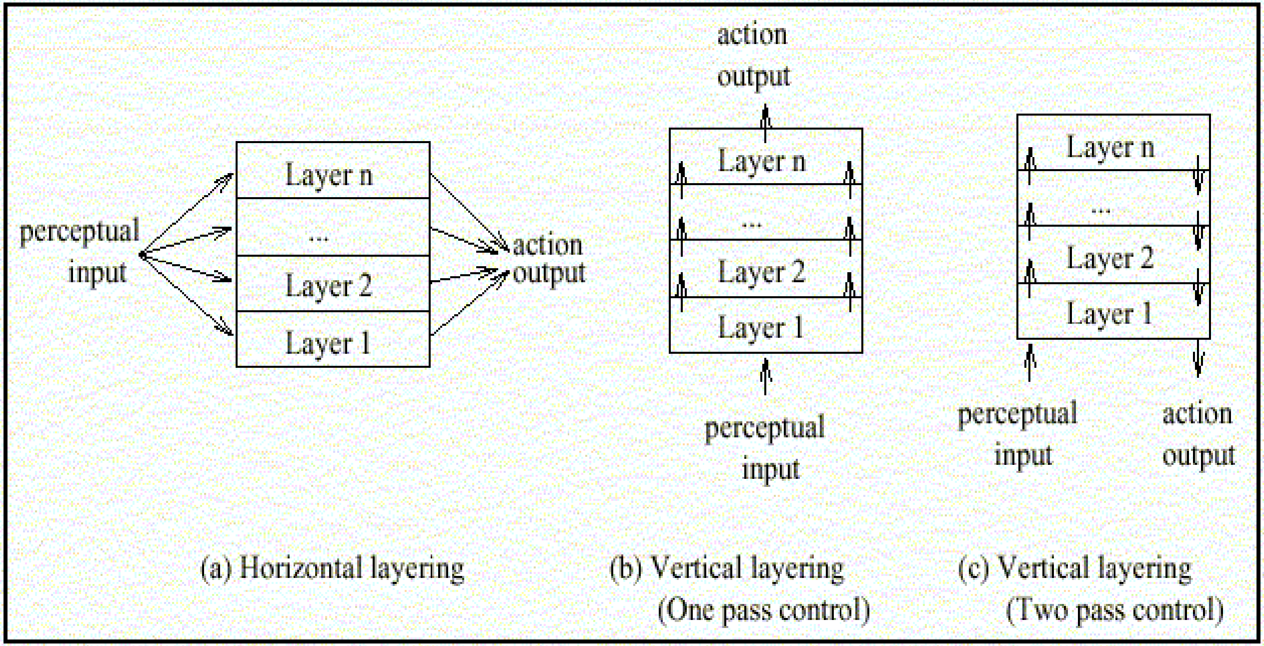
Hybrid Architectures

- **To build an agent out of two (or more) subsystems:**
  - ***deliberative***
    - **containing a symbolic world model**
  - ***reactive***
    - **capable of reacting to events without complex reasoning**

Hybrid Architectures

- **Agent's control subsystems are arranged into a hierarchy, with higher layers dealing with information at increasing levels of abstraction**
- **What kind of control framework do the agent's subsystems embed in?**
  - **Horizontal layering:** Layers are directly connected to the sensory input and action output
  - **Vertical layering:** Sensory input and action output are each dealt with by at most one layer each

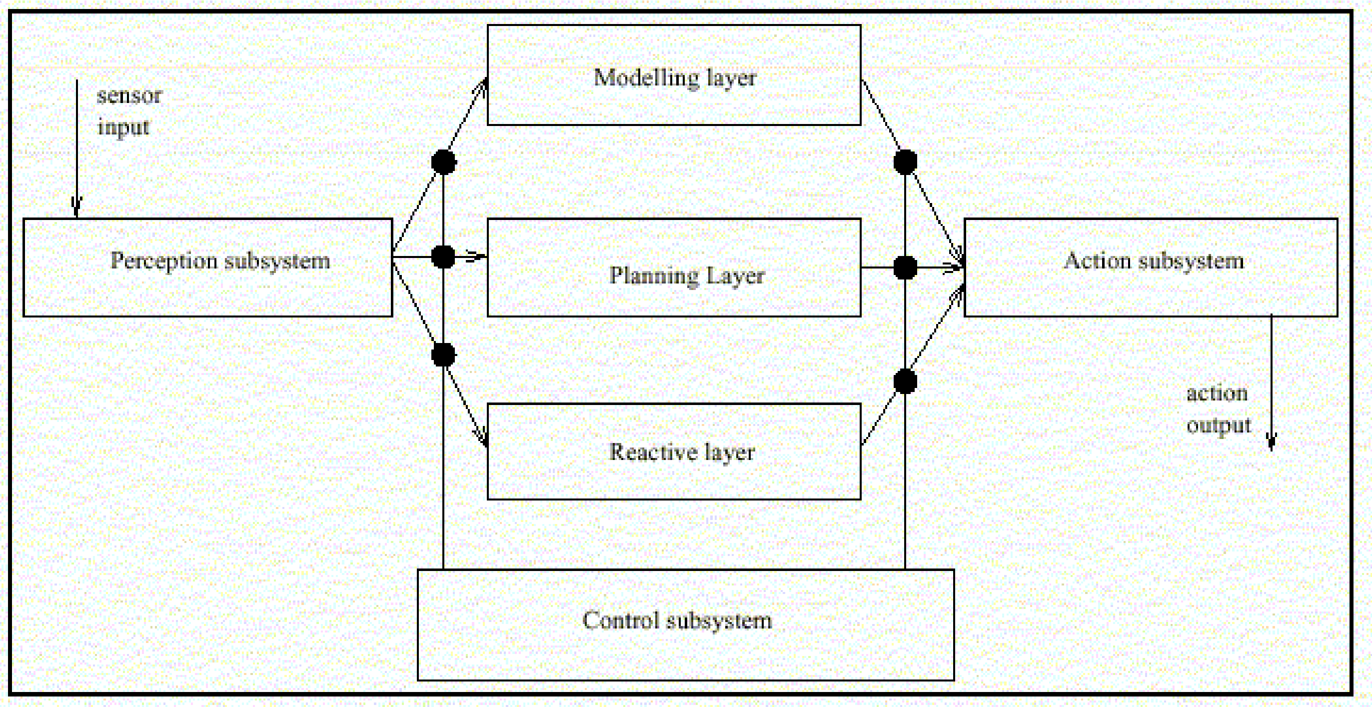
Hybrid Architectures



Hybrid Architectures

- **Horizontally layered:**
  - **The TouringMachines architecture consists of:**
    - **perception and action subsystems, which interface directly with the agent's environment**
    - **three control layers, embedded in a control framework, which mediates between the layers**

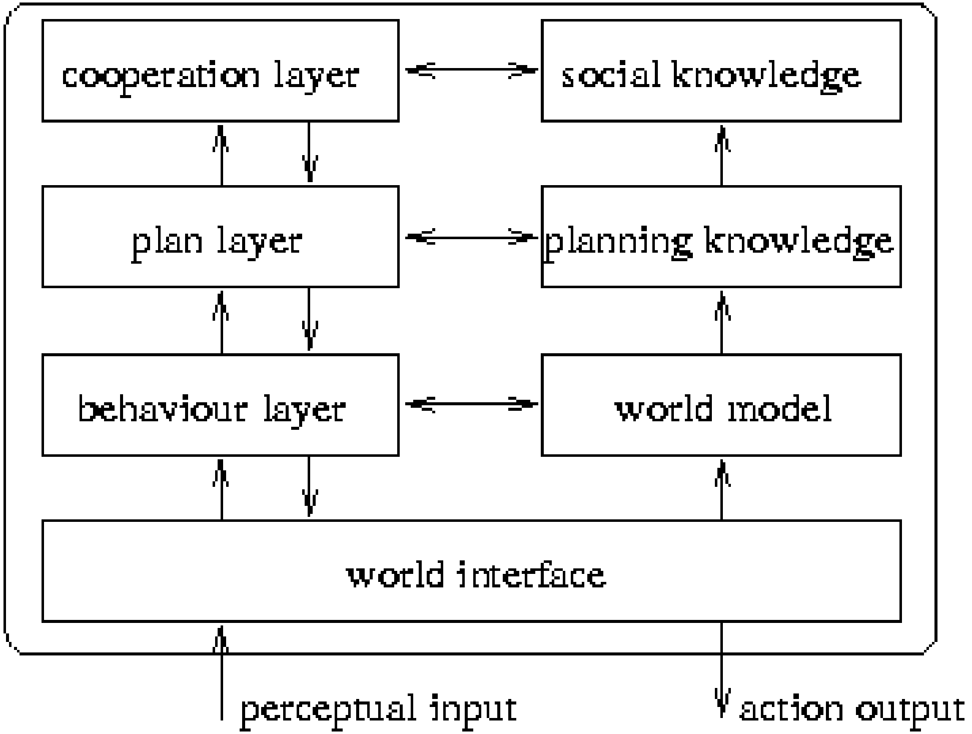
Hybrid Architectures



Hybrid Architectures

- **Vertically layered:**
  - **The Interrap architecture consists of**
    - **three control layers**
    - **the purpose of each layer appears to be rather similar to the purpose of each corresponding TouringMachines layers**
  - **The HIPE architecture consists of**
    - **N control layers**
    - **each layer is equipped with a deliberative, a proactive, and a reactive module**
    - **both vertical and horizontal interactions may occur**

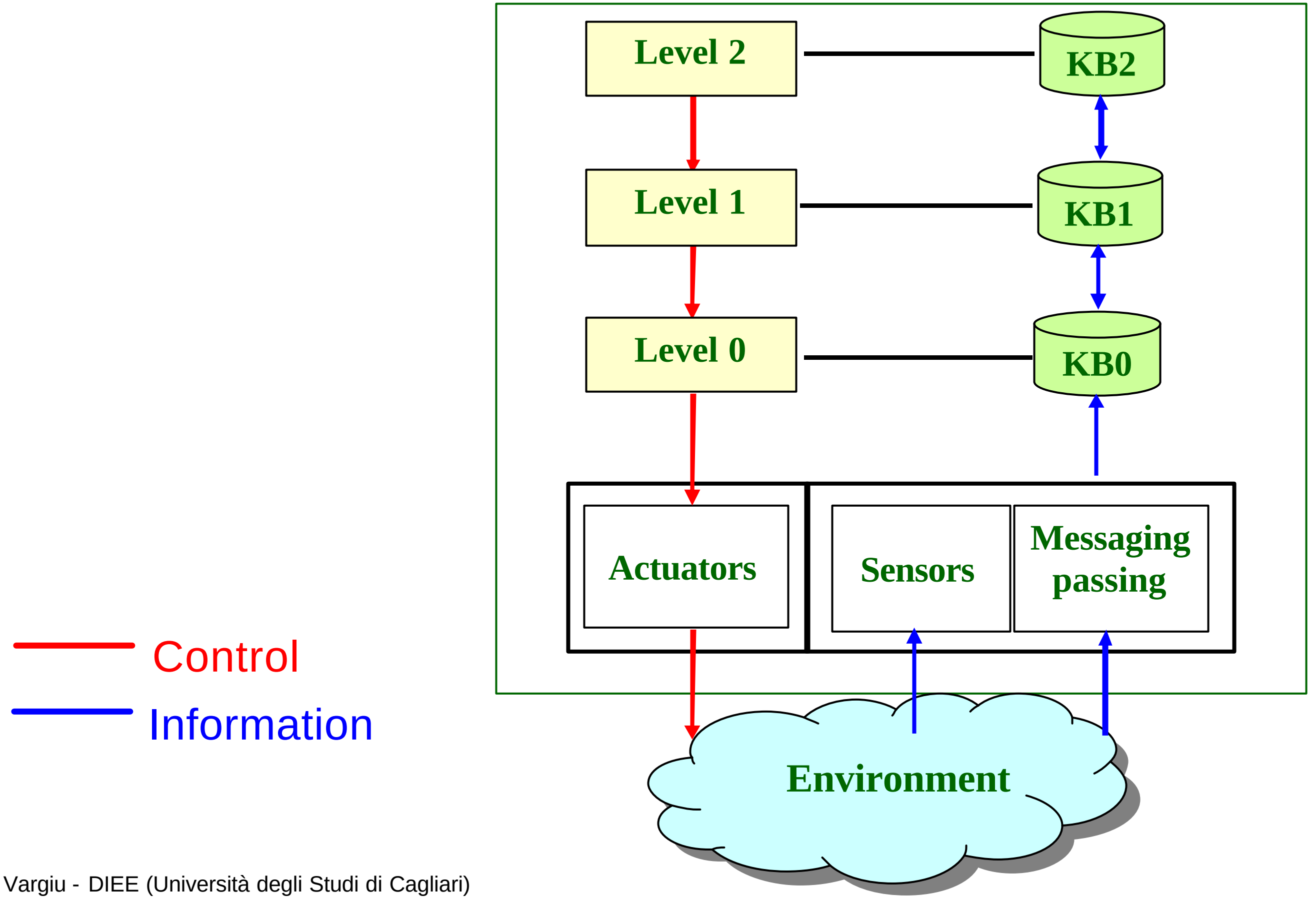
Hybrid Architectures



Hybrid Architectures

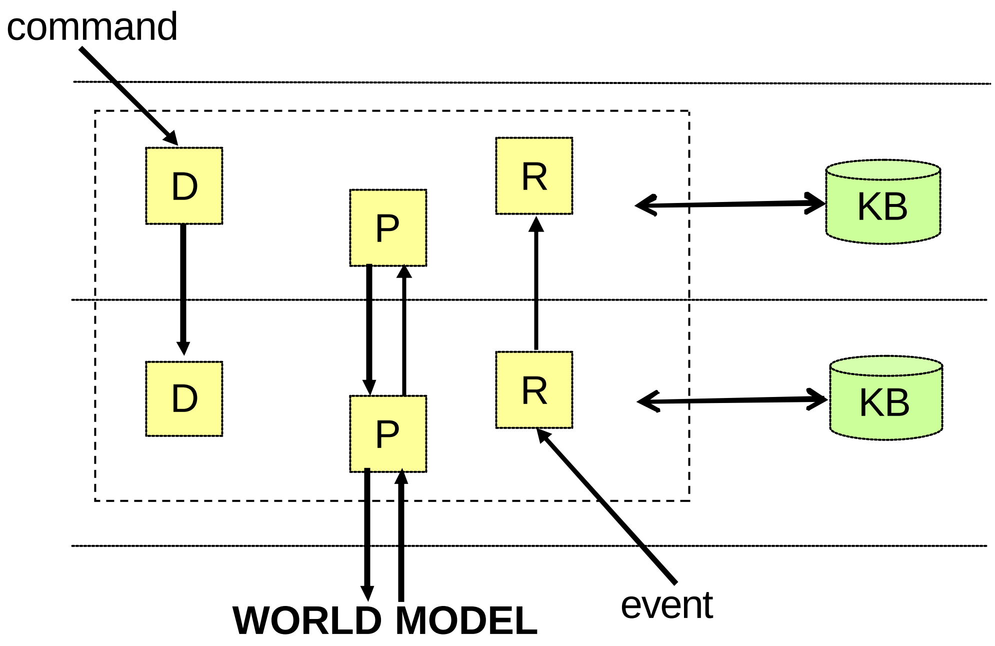
- Vertically layered:
  - The Interrap architecture consists of
    - three control layers
    - the purpose of each layer appears to be rather similar to the purpose of each corresponding TouringMachines layers
  - **The HIPE architecture consists of**
    - **N control layers**
    - **each layer is equipped with a deliberative, a proactive, and a reactive module**
    - **both vertical and horizontal interactions may occur**

Hybrid Architectures



Hybrid Architectures

➢ Vertical Interactions

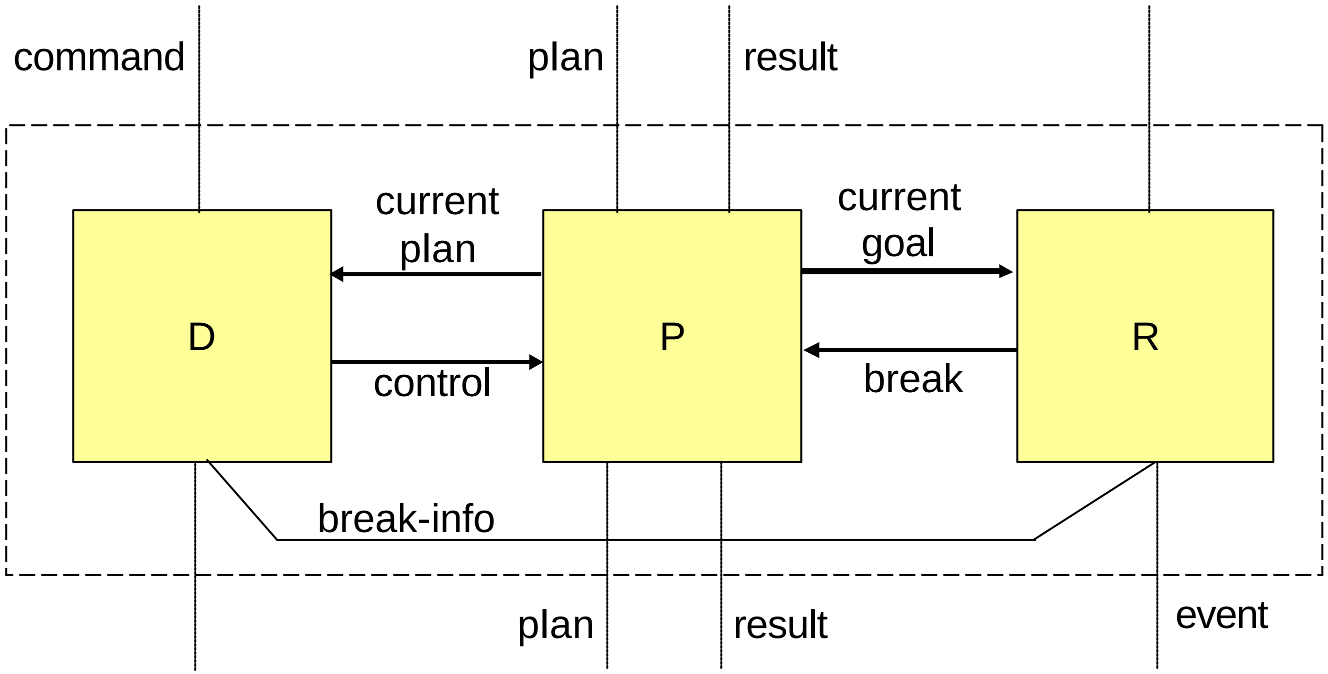


Hybrid Architectures

- **Vertical Interactions**
  - **Reactive Behavior**
    - **to adapt any incoming event with the kind of information contained in the KB of the corresponding layer**
  - **Proactive Behavior**
    - **is supported by a hierarchical planner distributed on N layers, each one devoted to cope with a different level of granularity**
  - **Deliberative Behavior**
    - **to let an external command propagate down to a layer able to understand it**

Hybrid Architectures

➢ Horizontal Interactions

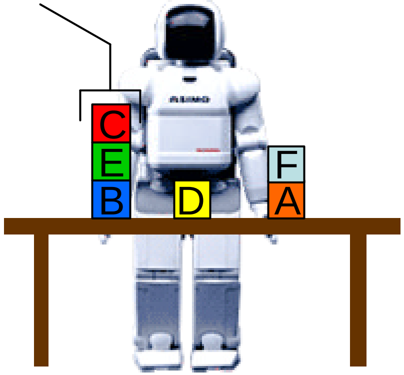


Hybrid Architectures

- **Horizontal Interactions**
  - **Reactive Behavior**
    - **contains a partially-ordered set of rules, each rule having the form *<pre, post, priority>***
  - **Proactive Behavior**
    - **has interaction with both the corresponding reactive and deliberative modules**
  - **Deliberative Behavior**
    - **interacts with the planner by sending a command in the set *{continue, start, reset, resume}***

Part II

Classical Planning

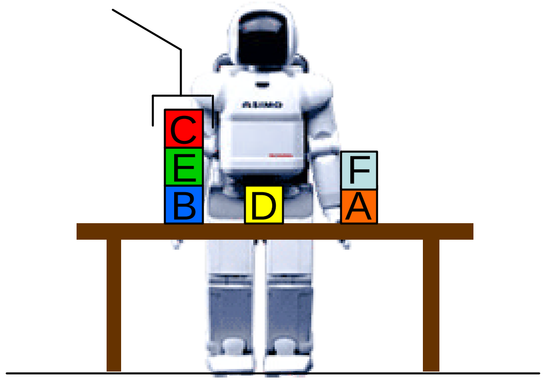


Outline

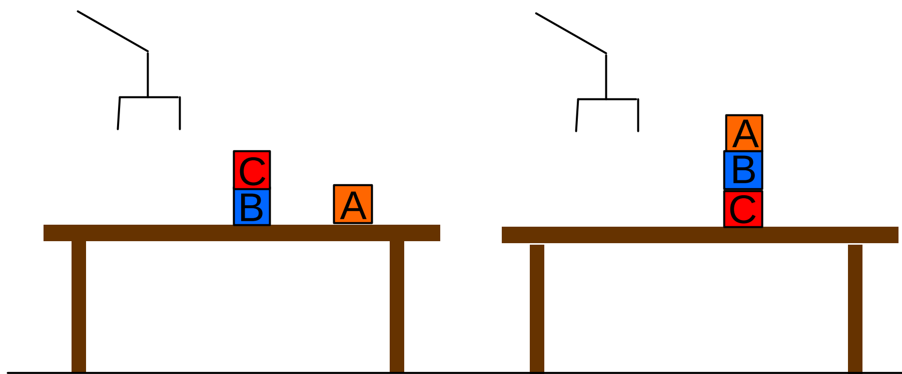
- Introduction
- Agents architectures
- **Planning in classical domains**
  - **The blocks-world domain**
  - **Classical planning and Classical planners**
  - **Planning by Abstraction**
- Planning in real domains
- Conclusions

The Blocks-World Domain

The domain



A sample problem

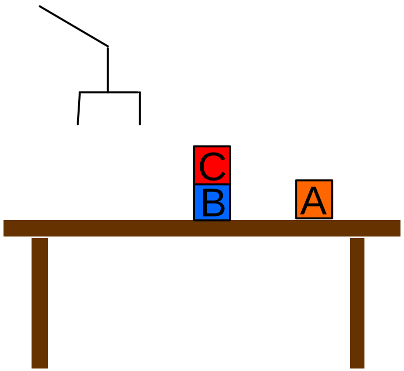


Initial state **Goal state**

The Blocks-World Domain

```
(define (domain BLOCKS)
  (:types block)
  (:predicates (on ?x - block ?y - block)(ontable ?x - block)
               (clear ?x - block)(handempty)(holding ?x - block))
  (:action pick-up
        :parameters (?x - block)
        :precondition (and (clear ?x) (ontable ?x) (handempty))
        :effect         (and (not (ontable ?x)) (not (clear ?x))
                             (not (handempty)) (holding ?x)))
  (:action put-down
        :parameters (?x - block)
        :precondition (holding ?x)
        :effect (and (not (holding ?x))(clear ?x)(handempty) (ontable ?x)))
  (:action stack
        :parameters (?x - block ?y - block)
        :precondition (and (holding ?x) (clear ?y))
        :effect (and (not (holding ?x))(not (clear ?y))(clear ?x)
                     (handempty) (on ?x ?y)))
  (:action unstack
        :parameters (?x - block ?y - block)
        :precondition (and (on ?x ?y) (clear ?x) (handempty))
        :effect (and (holding ?x)(clear ?y)(not (clear ?x))
                     (not (handempty))(not (on ?x ?y)))))
```

The Blocks-World Domain



Precondition:

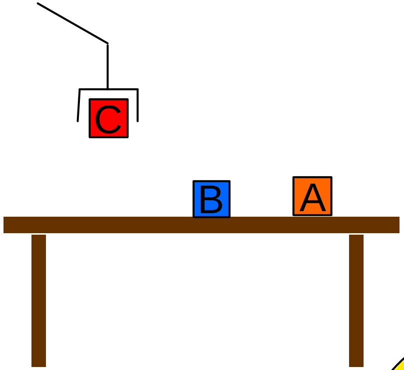
(on C B)
(clear C)
(handempty)

Action:

unstack (C, B)

Effect:

(holding C)
(clear B)
(not (handempty))
(not clear C)
(not (on C B)



Classical Assumptions

- **Atomic Time**
- **Deterministic effects**
- **Omniscience**
- **Sole cause of change**

Execution of an action is indivisible and uninterruptible, thus we need not to consider the state of the world while execution is proceeding. Simultaneously executed actions are impossible

Classical Assumptions

- Atomic Time
- **Deterministic effects**
- Omniscience
- Sole cause of change

The effect of executing any action is a deterministic function of the action and the state of the world when the action is executed.

Classical Assumptions

- Atomic Time
- Deterministic effects
- Omniscience
- Sole cause of change

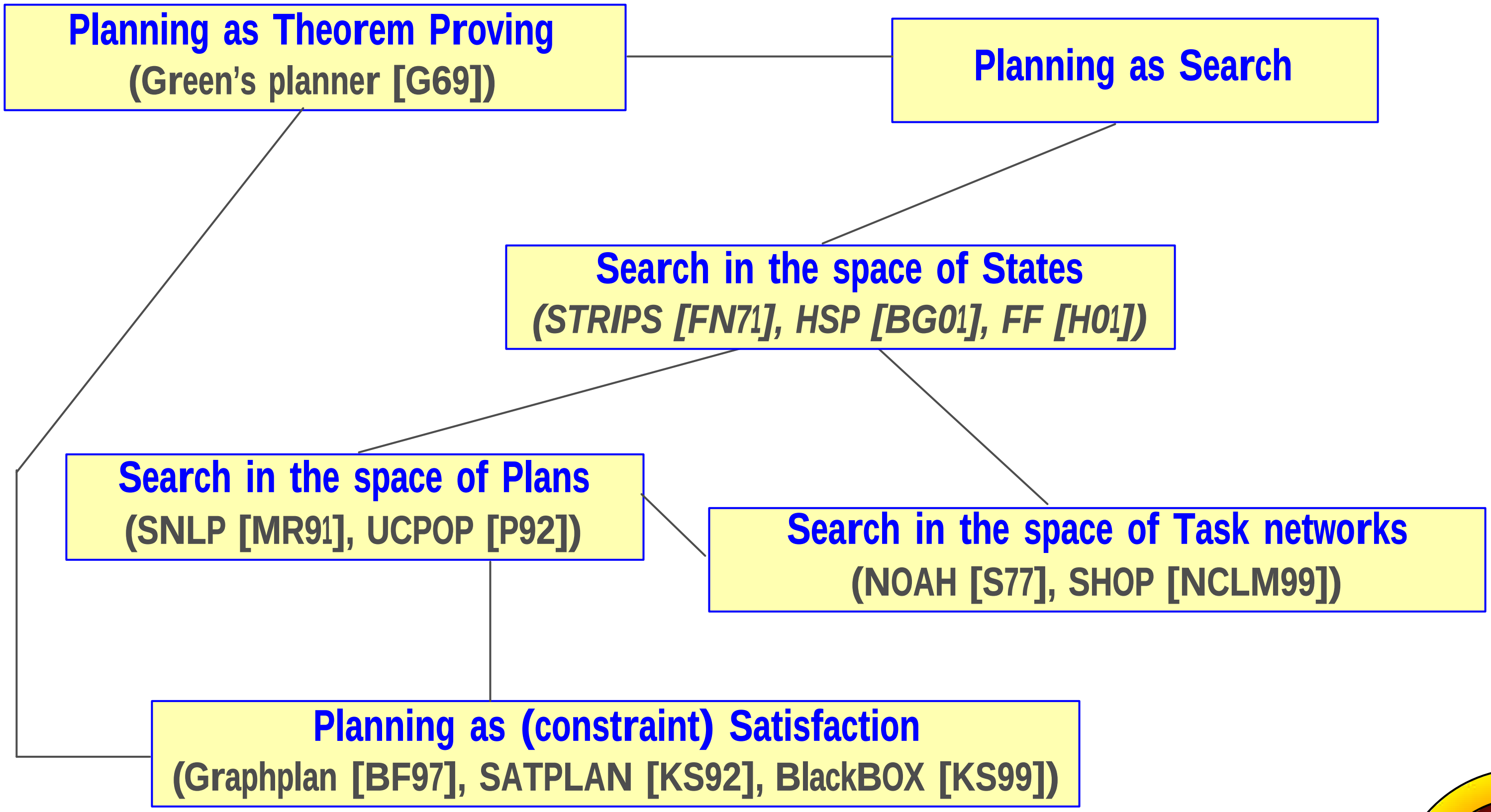
The agent has complete knowledge of the initial state of the world and of the nature of its own actions.

Classical Assumptions

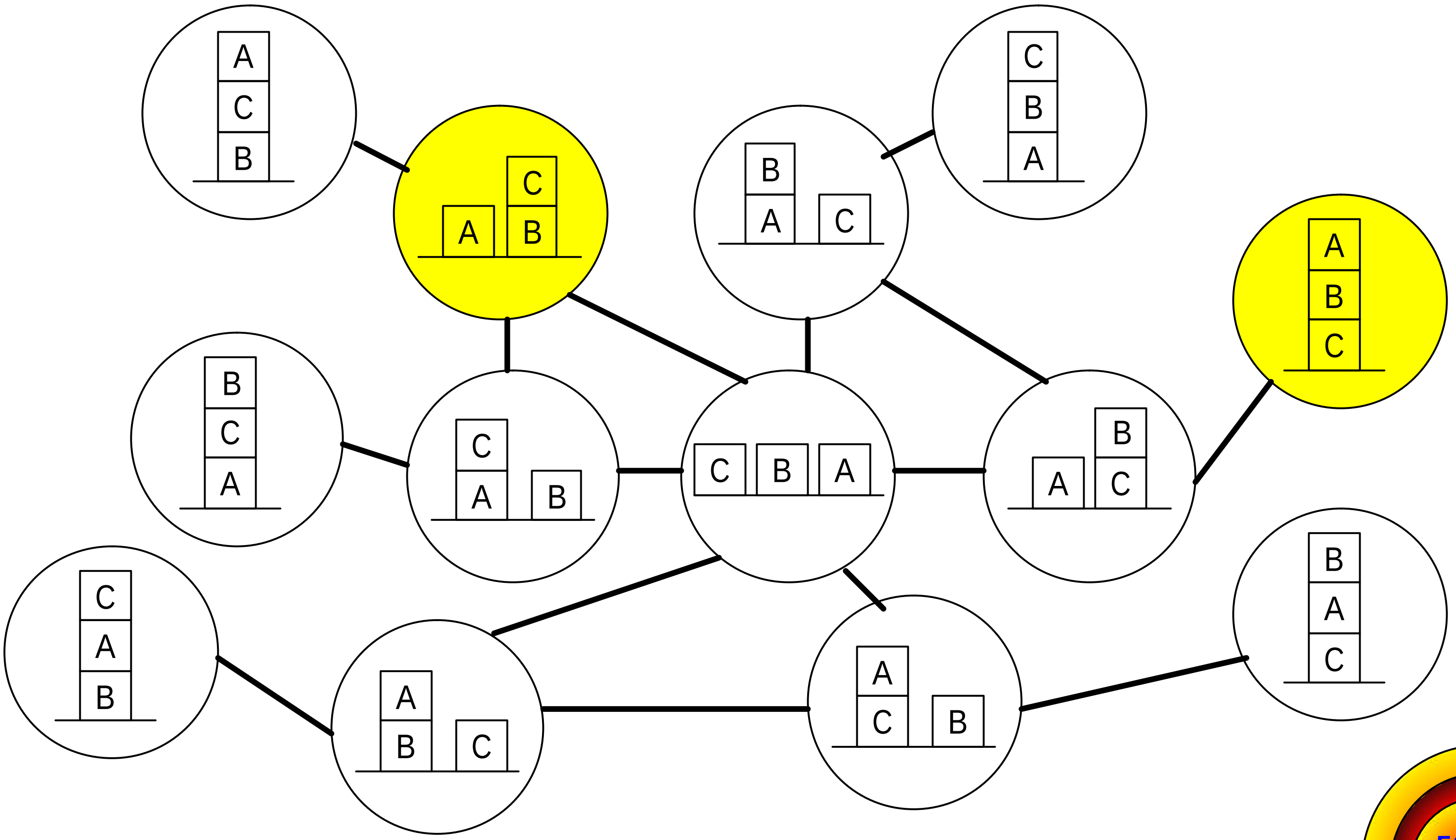
- Atomic Time
- Deterministic effects
- Omniscience
- Sole cause of change

The only way the world changes is by the agent's own actions. There are no other agents and the world is static by default.

Classical Planning: Evolution [k97]



Space of States



Space of States: HSP

- **The idea:**
  - **Select best action given current state after brief deliberation**
  - **Disregard delete list in actions effects**

Space of States: HSP

- **The heuristic function**
  - **Assume propositional rules *C* ® *p* for atoms *p* that can be established by some action *a* with precondition *C***
  - **Apply such rules in parallel to atoms true in *s***
  - **When each atom p is first derived by means of rules C_i ® p, assign to *p* the number *g(p; s)*:**

$$g(p,s) = 1 + \min_i \left[\sum_{q \in C_i} g(q,s) \right]$$

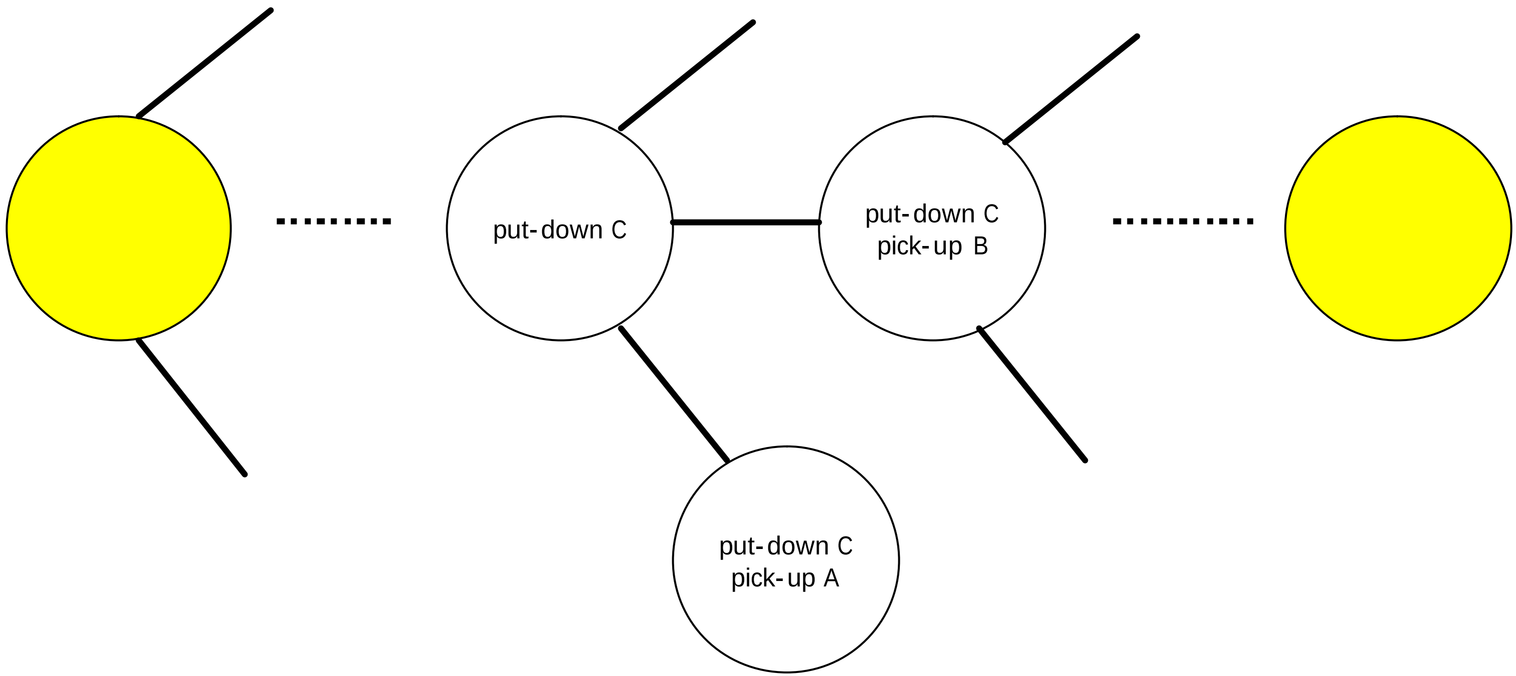
Space of States: HSP

- **The heuristic function**
  - **For atoms $p \in s$, $g(p; s) = 0$**
  - **Heuristic h(s) relative to goal G defined as:**

$$h(s) \overset{def}{=} \sum_{p \in G} g(p,s)$$

  - **Heuristic h(s) is not admissible**

Space of Plans

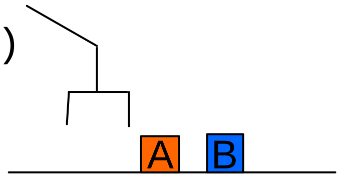


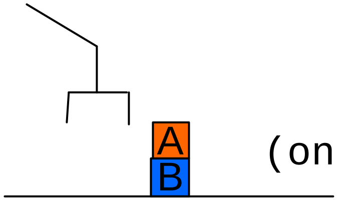
Space of Plans: UCPOP

- **UCPOP starts with an initial, dummy plan that consist solely of a “start” step and a “goal” step**

start

```
(ontable A)(ontable B)
(clear A)(clear B)
(handempty)
```





`(on A B)` ***goal***

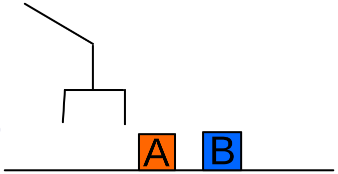
Space of Plans: UCPOP

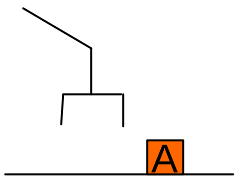
- **UCPOP then attempts to complete the initial plan by adding new steps and constraints until all preconditions are guaranteed to be satisfied**

Space of Plans: UCPOP

start

(ontable B)
(clear A)(handempty)(ontable A)
(clear B)

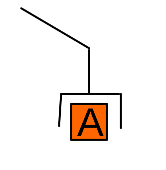


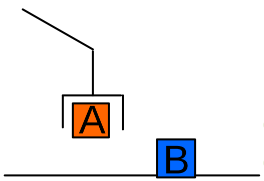


(ontable A)
(handempty)
(clear A)

pick-up B

(holding A)
(not (clear A))
(not (handempty))

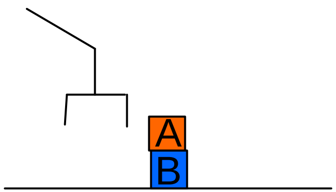


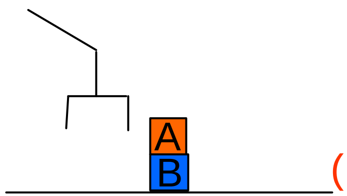


(holding A)
(clear B)

stack A B

(on A B)(handempty)
(clear A)
(not (holding A))
(not (clear B))





(on A B)

goal

Space of Task Networks

- **The planning system begins with an initial state-of-the-world and with the objective of creating a plan to perform a set of *tasks* (abstract representations of things that need to be done).**

Space of Task Networks

- **HTN planning is done by problem reduction**
  - **The planner recursively decomposes tasks into subtasks**
  - **It stops when it reaches *primitive* tasks that can be performed directly by *planning operators*.**

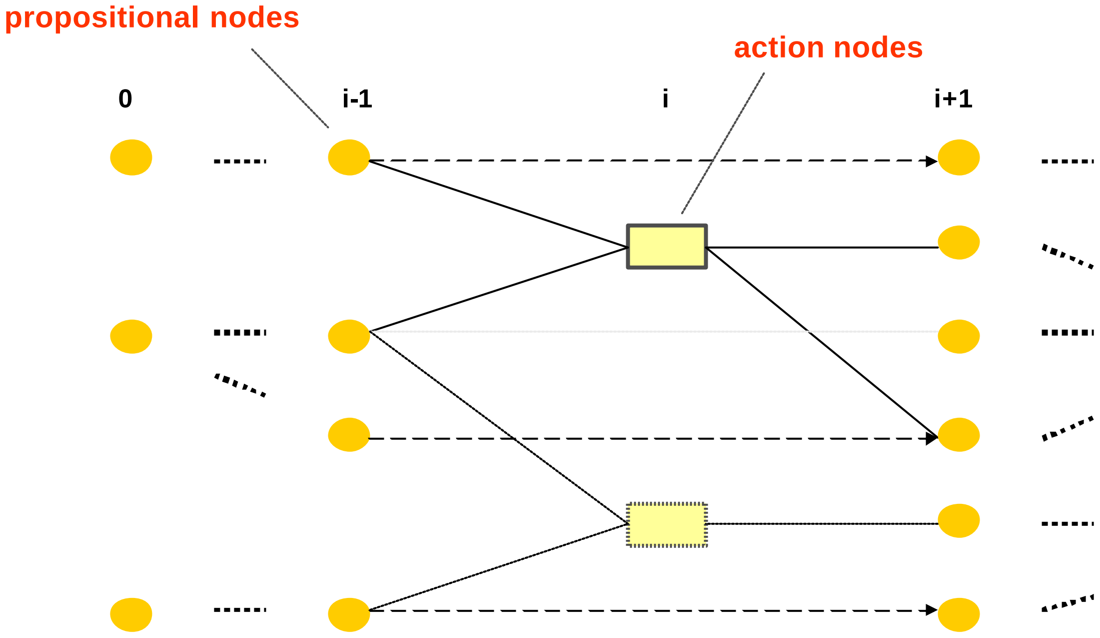
Space of Task Networks

- **To decompose nonprimitive tasks into subtasks, a set of *methods* is needed:**
  - **Each method is a schema for decomposing a particular kind of task into a set of subtasks.**
  - **For each task, there may be more than one applicable method, and thus more than one way to decompose the task into subtasks.**
  - **The planner may have to try several of these alternative decompositions before finding one that is solvable at a lower level.**

Space of Task Networks: SHOP

- **SHOP (Simple Hierarchical Ordered Planner) is a *domain-independent* implementation of ordered task decomposition was SHOP.**
- **SHOP can be configured to work in many different planning domains.**

Planning as Satisfatcion: GRAPHPLAN



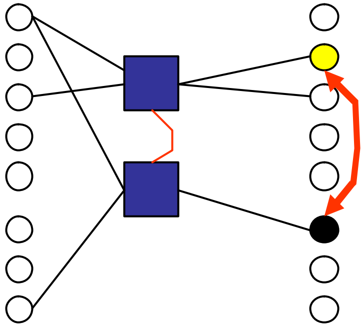
- graph expansion
- solution extraction

Planning as Satisfatcion: GRAPHPLAN

- **Two actions at level *i* are mutex if either:**
  - **inconsistent effects**
  - **interference**
  - **competing nodes**

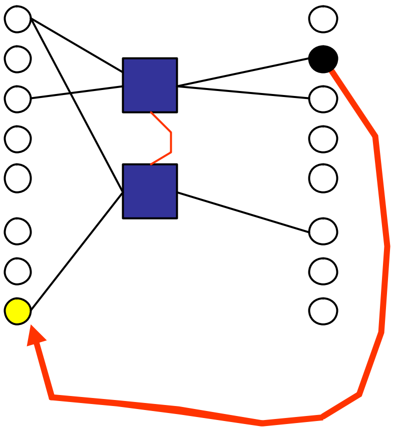
Planning as Satisfatcion: GRAPHPLAN

- Two actions at level *i* are mutex if either:
  - inconsistent effects
  - interference
  - competing nodes



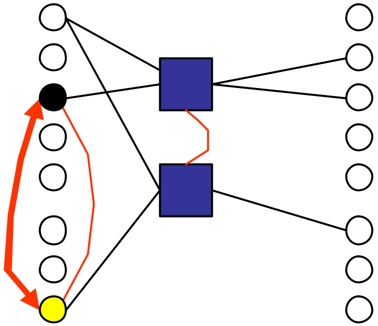
Planning as Satisfatcion: GRAPHPLAN

- Two actions at level *i* are mutex if either:
  - inconsistent effects
  - interference
  - competing nodes



Planning as Satisfatcion: GRAPHPLAN

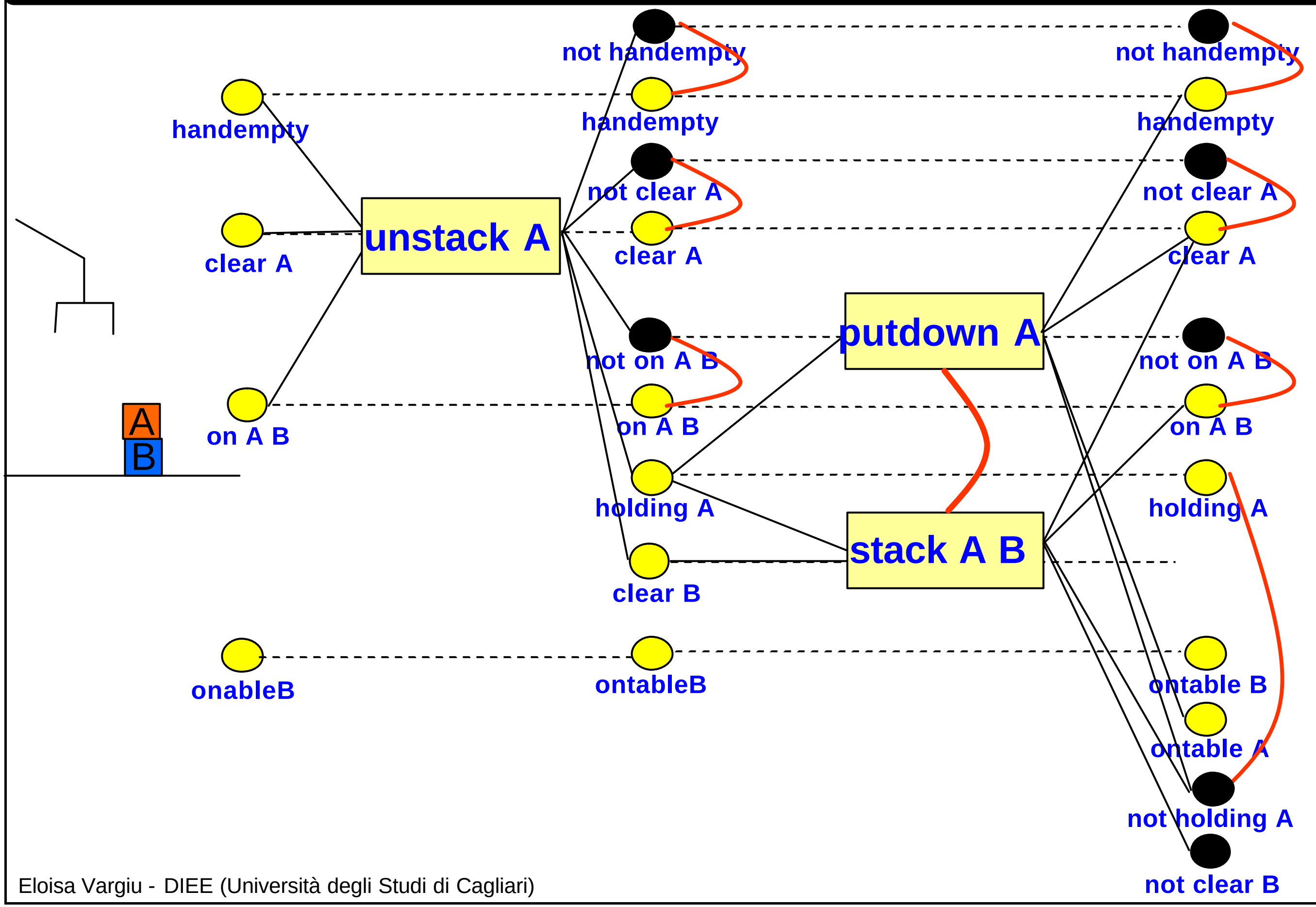
- Two actions at level *i* are mutex if either:
  - inconsistent effects
  - interference
  - competing nodes



Planning as Satisfatcion: GRAPHPLAN

- **Two propositions at level *i* (*Inconsistent support*):**
  - **it is the negation of the other**
  - **all ways of achieving the propositions are mutex**

Planning as Satisfatcion: GRAPHPLAN



Planning by Abstraction

- **An effective approach for dealing with the inherent complexity of planning tasks**
- **Exploits an ordered set of *abstractions* for controlling the search**
- **Under certain assumptions, it can reduce the size of the search space from exponential to linear in the size of the solution**

Abstraction as Control Heuristics

- ***Abstraction*** **is a technique aimed at providing some control heuristics (*)**
- **The original search space is mapped into corresponding abstract spaces, in which irrelevant details are disregarded at different level of granularity**

(*) abstract levels are used to control the search at the ground level

Abstraction Techniques

- Several abstraction techniques have been proposed in the literature, including:
  - Action-based [K87]
  - State-based [S74] [K94]
  - Hierarchical Task Networks [EHN94]
  - Case-based [BW95]

Abstraction: General Perspective

- An *abstraction* is a mapping between representations of a problem
- An *abstraction hierarchy* consists of an ordered set of *domains* and *mapping functions* between adjacent levels

Abstraction Techniques

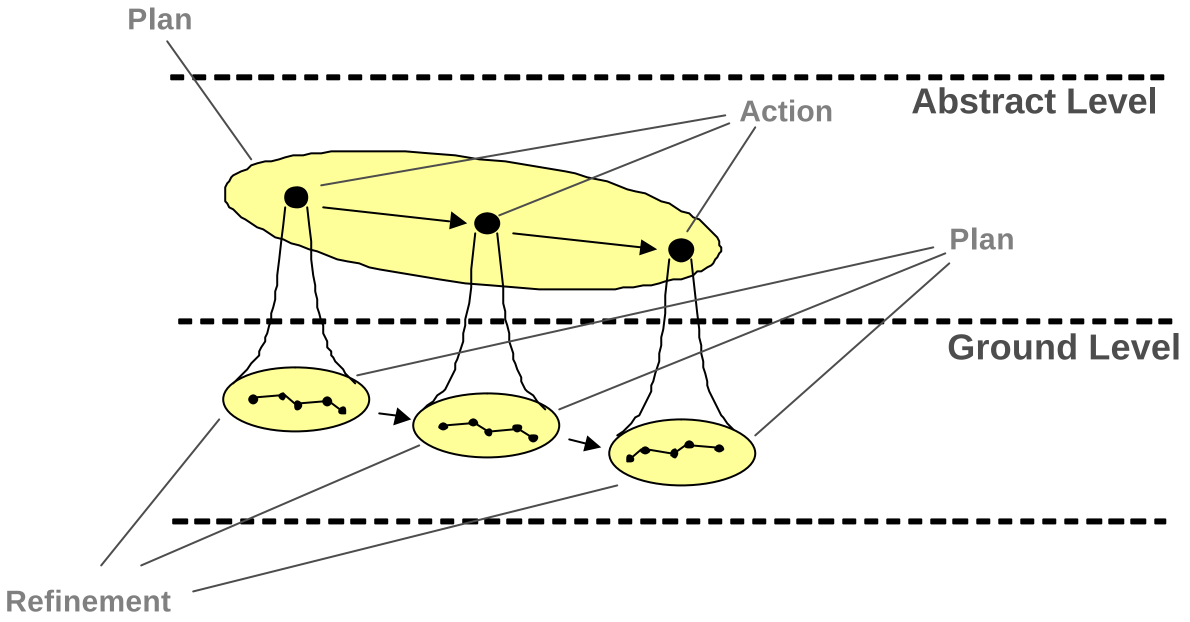
- In general, abstraction might occur on <u>predicates</u>, <u>types</u>, and <u>operators</u>.
- Typical examples of abstraction are:
  - Based on <u>Predicates/Types</u>
    - relaxed models [S74]
    - reduced models [K94]
  - Based on <u>Operators</u>
    - macro-operators [K87]

Abstraction: Drawbacks

- **Usually, abstraction weakens the original problem space, thus "false" solutions may be introduced at the abstract levels**
- **In presence of "false" solutions *backtracking* must be considered**
- **The effectiveness of the abstraction mechanisms strictly depends on the ratio between "true" and "false" solutions**

Planning by Abstraction at a Glance

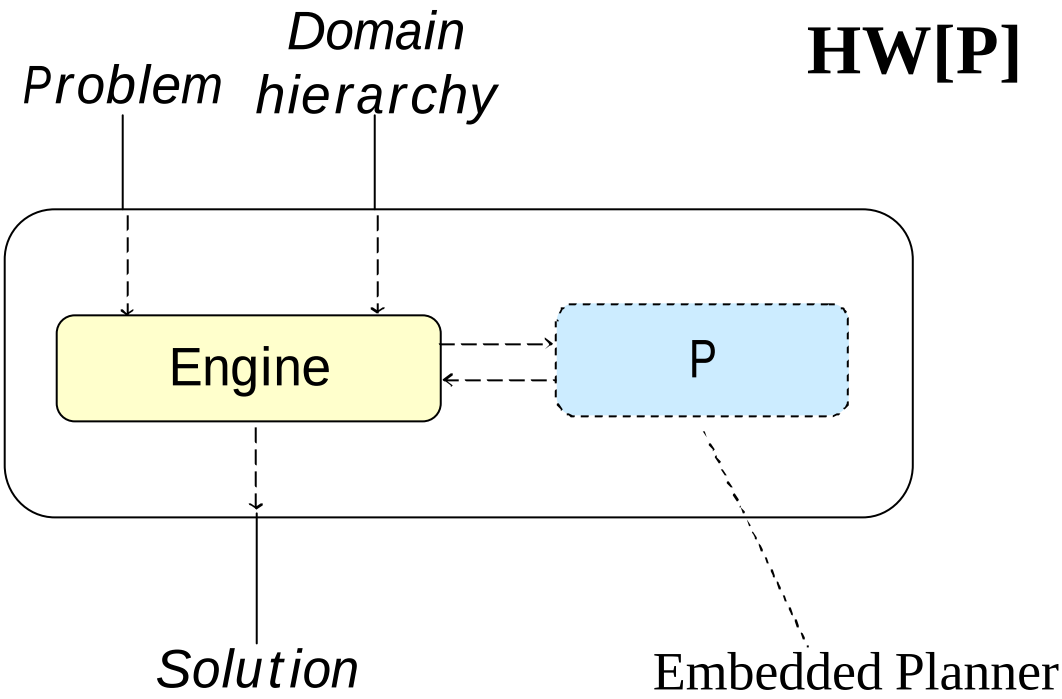
... on a two-levels abstraction hierarchy



Using Abstraction

- **What do we need to perform planning by abstraction?**
  - **An *engine* able to enforce hierarchical planning**
    HW[ ] [ACV03a]
  - **A *syntactic support* to express abstraction hierarchies (based on PDDL)**
    HPDDL [ACV03b]
  - **A *technique* to generate abstraction hierarchies**
    DHG [ACV04]

HW[]



HW[ ] = *Hierarchical Wrapper*

HW[]

- **HW[ ] embodies a generic planner *P***
- **HW[ ] delegates to *P* the search at any required level of abstraction**
- **HW[ ] performs a suitable switching between abstraction levels**
- **HW[P] is the resulting system**

HW[]

- **Input:**
  - **a ground-level *problem***
  - **a description of the *abstraction hierarchy***
    - **a set of *n* domains**
    - **a set of *n-1* mapping functions**
- **Output:**
  - **a *solution* to the given problem**

HW[]

- **To search for a solution:**
  - **HW[ ] translates the *init* and *goal* sections from ground to abstract level**
  - ***P* is invoked to search for an abstract solution**
  - **each abstract operator is refined by repeatedly invoking *P***

HW[]

- **Refinements are performed by:**
  - **activating *P*, at the ground level, on the *goal* obtained by translating downward its *effects***
  - **following the order specified by the abstract plan (the initial state of each refinement depends on all the previous refinements)**
  - **preserving *subgoals* attained during previous *refinements***

HW[]

- When an attempt to refine the current abstract solution fails…
  - *P* is invoked to find the next abstract solution (*)
- To ensure *completeness*…
  - if no abstract solution could be successfully refined, an overall search is performed at the ground level

(*) unless the number of abstract solutions found so far exceeds a given threshold

HPDDL

- To our knowledge, existing planning systems tailored for abstraction did not resort to a common notation
- To contrast this lack of a standardization, we have proposed an extension to PDDL able to represent *abstraction hierarchies*

HPDDL

- **An abstraction hierarchy is represented by:**
  - **a set of domains (*n*)**
    Each domain is expressed according to the standard PDDL notation (i.e., ‘*define domain*’)
  - **a set of mapping functions (*n - 1*)**
    Each mapping function is expressed by a new statement (i.e., ‘*define hierarchy*’)

HPDDL

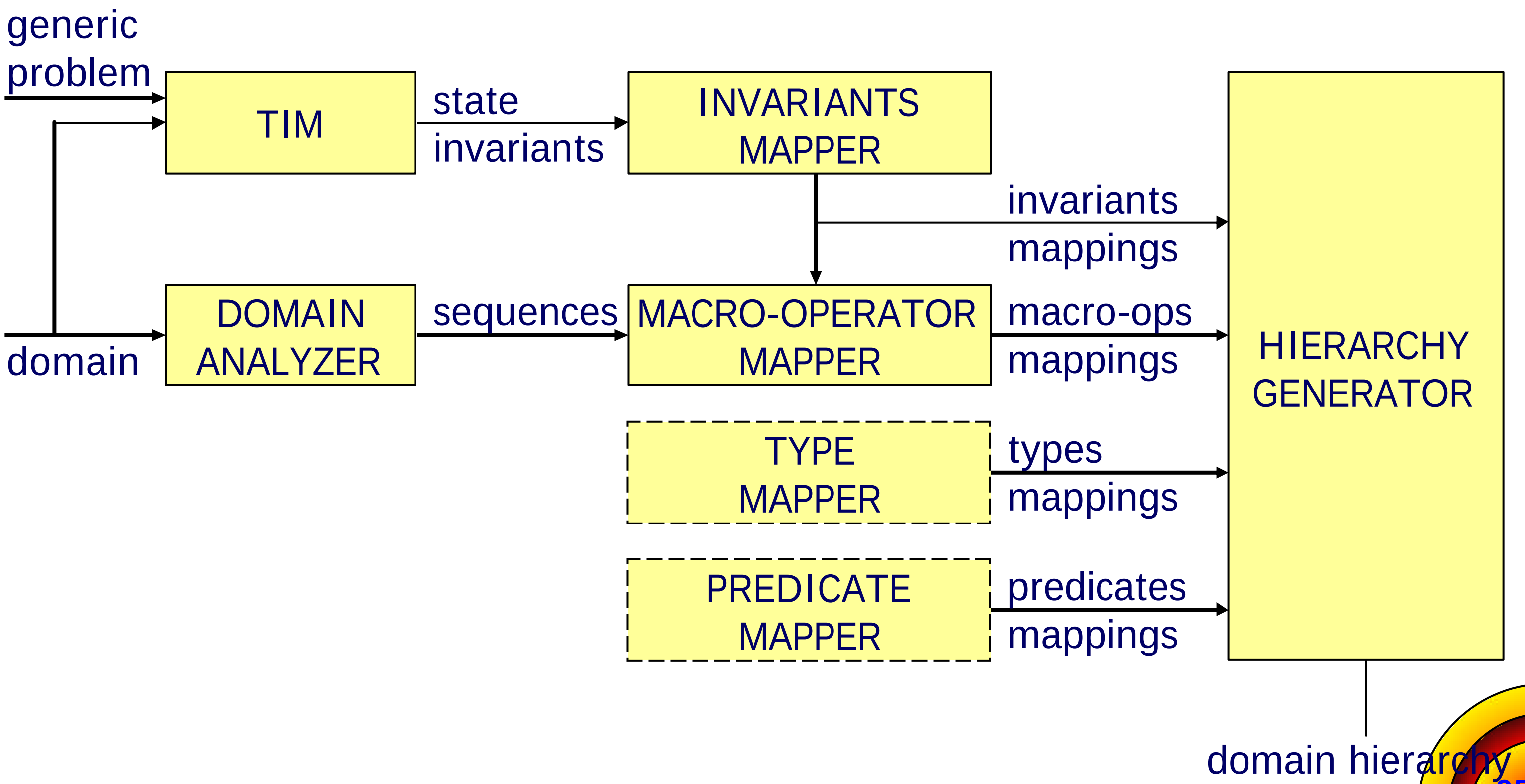
- **Defining an abstraction hierarchy using HPDDL:**

```
(define (hierarchy <name>)
  (:domains <domain-name>+)
  (:mapping (<src-domain> <dst-domain>)
   [:types <types-def>]
   [:predicates <predicates-def>]
   [:actions <actions-def>])*)
```

DHG

- **Abstraction hierarchies can be hand-coded by a domain engineer**
- **To facilitate the setting of the abstraction hierarchies automatic techniques can be considered**

DHG


generic problem
TIM
state invariants
INVARIANTS MAPPER
invariants mappings
domain
DOMAIN ANALYZER
sequences
MACRO-OPERATOR MAPPER
macro-ops mappings
TYPE MAPPER
types mappings
PREDICATE MAPPER
predicates mappings
HIERARCHY GENERATOR
domain hierarchy

DHG

- **The *domain analyzer* searches for macro-operators:**
  1) **building and**
  2) **pruning a directed graph (*)**

(*) where:

- ***nodes* represent *operators*, and**
- ***edges* represent *relations* between *effects* of the source node and *preconditions* of the destination node**

DHG

- **The macro-operator mapper builds macro-operators**
  1) **calculating pre- and post-conditions starting from a given sequence**
  2) **selecting only relevant macro-operators**

DHG

- **Generating pre- and post-conditions starting from a sequence of operators with variable parameters involves an unification process to avoid semantic inconsistencies**
- **State invariants (retrieved using TIM [FL98]) are used to solve semantic inconsistencies**
- **The information about the domain, enriched with state invariants, allows to correctly unify macro-operators**

DHG

- To automatically build the domain hierarchy, the *domain hierarchy* module requires a set of mapping functions
- Mapping functions contain the translation rules on types, predicates, operators, and invariants

DHG

- The mapping functions are expressed through the *:mapping* clause of the *define hierarchy* statement:

```
<mapping-def>::=
  (:mapping (<src-dom> <dst-dom>)
    [:types <types-def>]
    [:predicates <predicates-def>]
    [:actions <actions-def>]
    [:invariants <invariants-def>])
```

DHG

- **Given the mapping functions, abstract operators and predicates can be generated:**
  - **an abstract operator is generated from each macro-operator**
  - **predicates at the abstract level are the same of the ground level**

Part III

Real World

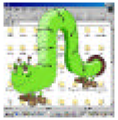
Outline

- Introduction
- Agents architectures
- Planning in classical domains
- **Planning in real domains**
  - **A Case Study: the SFZ Game**
  - **SFZ Agents**
  - **Planning by Abstraction in SFZ**
- Conclusions

Case Study: the SFZ Game

- **Mission:**
  - **try to protect the virtual city from external attacks and to take over its critical gates.**
- **Environment:**
  - **a “virtual” city, equipped with buildings, streets, an Internet-like network, etc.**

SFZ Agents

- **Avatars**
  (i.e., players' representatives within the city) 
- **inhabitants of the city**
  (i.e., clerks, policemen, terrorists, etc.)  
- **inhabitants of the Internet-like network**
  (i.e., computer viruses, sniffers, etc.) 

Planning in SFZ

Planning in SFZ

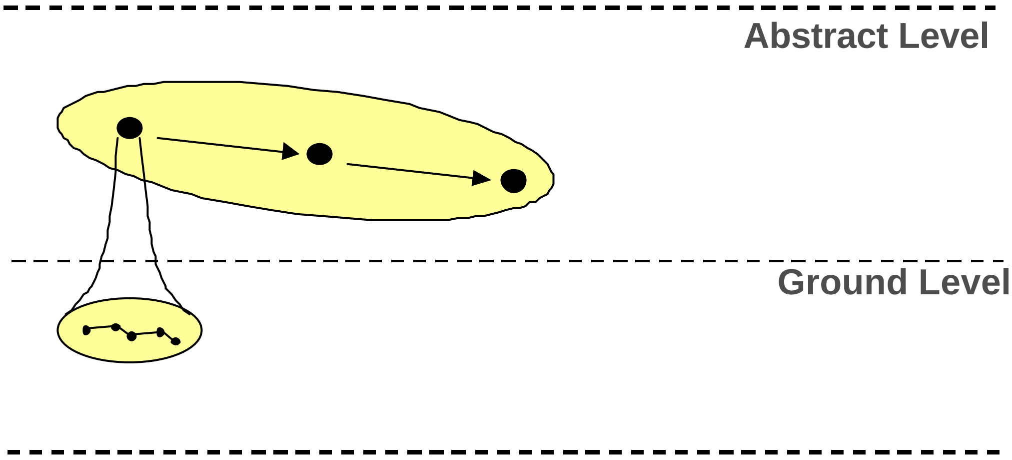
- **The underlying environment:**
  - **incomplete knowledge**
  - **dynamic environment**
  - **non-deterministic effects**

Planning in SFZ

- **Hierarchical Interleaving Planning and Execution (HIPE)** [AV01]
  - **Using abstraction hierarchies on top of a suitable hierarchical micro-architecture (the HIPE architecture)**
  - **Interleaving Planning and Execution**
  - **PDDL-compliant at each level of the hierarchy**

Planning in SFZ

- Strong assumption: the divide-and-conquer strategy can be applied ...
- Execution starts as soon as the first abstract operator has been refined ...



Abstraction in SFZ

- **Two problems at the abstract level ...**
  - **Completeness - can all possible plans be found by the abstract-level planner? If not, a search at the ground level (when needed) must be supported, too.**
  - **Soundness - does the abstract planner always find "true" plans? If not, backtracking should be supported (in this case, is interleaving still feasible?)**

Abstraction in SFZ

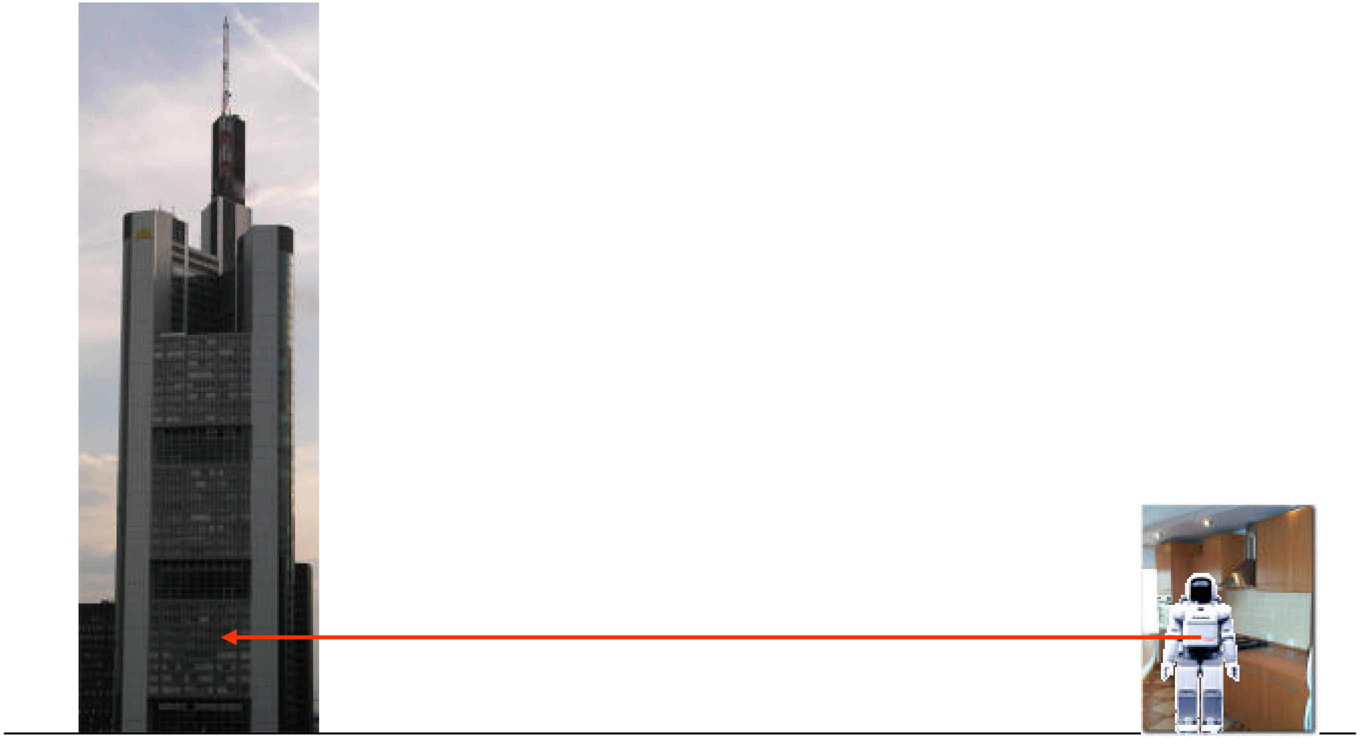
- **Fortunately, SFZ avatars can fail without compromising the user's odds to win ...**
- **Furthermore, the environment is dynamic (so that also fully refined ground-plans do not always guarantee that the goal can be attained)**

Abstraction in SFZ

- **Completeness ...**
  - **to be obtained as an asymptotic property (post-mortem analysis on ground-level solutions -in particular, the ones whose corresponding abstract solutions has not been found)**
- **Soundness ...**
  - **“hand-made”, by specifying a default behavior (ako near-DRP)**

Planning by Abstraction in SFZ

- **Abstract solution:** `(move home building)`



Planning by Abstraction in SFZ

- **Ground refinement:**
  - `(open door)`
  - `(go-outside home)`
  - `(move home building)`

Conclusions

Conclusions

- **Thinking in terms of agent-oriented technology, an agent devised to solve complex problems should exhibit a suitable planning capability and the ability of adapting itself to the given environment, while trying to achieve its own goals**
- **Abstraction techniques can help to design powerful agent architectures; in particular for solving real-world problems, which are inherently complex and difficult to cope with**

Acknowledgments

I wish to thank

Prof. G. Armano and Dott. G. Cherchi

for their useful suggestions and valuable help.

References

References

[AV01] G. Armano, E. Vargiu. An Adaptive Approach for Planning in Dynamic Environments. Proceedings of the International Conference on Artificial Intelligence (ICAI'01), pp.987-993, Las Vegas, Nevada, 2001.

[ACV01] G. Armano, G. Cherchi, E. Vargiu. *An Agent Architecture for Planning in a Dynamic Environment*. Proceedings of the Italian National Conference on Artificial Intelligence (AI*IA 2001), Italy, 2001.

[ACV03a] G. Armano, G. Cherchi, E. Vargiu. *A Parametric Hierarchical Planner for Experimenting Abstraction Techniques*. Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03), Acapulco, Mexico, 2003.

References

[ACV03b] G. Armano, G. Cherchi, E. Vargiu. *An Extension to PDDL for Hierarchical Planning*. Workshop on PDDL (ICAPS'03), Trento, Italy, 2003.

[ACV04] G. Armano, G. Cherchi, E. Vargiu. *Automatic Generation of Macro-Operators from Static Domain Analysis*. Proceedings of the European Conference on Artificial Intelligence (ECAI'04), 1994.

[BW95] R. Bergmann, W. Wilke. *Building and Refining Abstract Planning Cases by Change of Representation Language*. Journal of Artificial Intelligence Research (JAIR), 3:53-118, 1995.

References

[BF97] A. Blum, M. Furst. *Fast Planning Through Planning Graph Analysis.* Artificial Intelligence, 90:281-300, 1997.

[BG01] B. Bonet, H. Geffner. *Planning as Heuristic Search*. Artificial Intelligence, Special issue on Heuristic Search, 129(1-2), 2001.

[B86] R.A. Brooks. *A Robust Layered Control System for a Mobile Robot*. IEEE Journal of Robotics and Automatation, 2(1):14-23, 1986.

[BIP88] M.E. Bratman, D.J. Israel, M.E. Pollack. *Plans and resource-bounded practical reasoning*. Computational Intelligence, 4:349-355, 1988.

References

[EHN94] **K. Erol, J. Hendler, D.S. Nau. *HTN Planning: Complexity and Expressivity*. In Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94), 1123--1128, AAAI Press / MIT Press, Seattle, WA, 1994.**

[F92] **I.A. Ferguson. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents.* PhD thesis, Clare Hall, University of Cambridge, UK, 1992.**

[FN71] **R. Fikes, N. Nilsson. *STRIPS: A new approach to the application of theorem proving to problem solving*. Artificial Intelligence, 2:189-208, 1971.**

References

[FL98] M. Fox, D. Long. *The Automatic Inference of State Invariants in TIM*. Journal of Artificial Intelligence Research (JAIR), 9:367-421, 1998.

[GL87] M.P. Georgeff, and A.L. Lansky. Reactive Reasoning and Planning. In Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-87), pp.677-682, Seattle, WA, 1987.

[G69] G.C. Green. *Application of Theorem Proving to Problem Solving*. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'69), pp.219—239, 1969.

[K97] S. Kambhampati. *Refinement Planning as a Unifying Framework*. AI Magazine 18(2):67-97, 1997.

References

[KS92] **H. Kautz, B. Selman.** ***Planning as Satisfiability.*** **Proceedings of European Conference on Artificial Intelligence (ECAI-92), 1992.**

[KS99] **H. Kautz, B. Selman.** ***Unifying sat-based and graph-based planning*****. Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99), pp318–325. Morgan Kaufmann, 1999.**

[K94] **C.A. Knoblock.** ***Automatically Generating Abstractions for Planning*****. Artificial Intelligence, 68(2):243--302, 1994.**

[K87] **R.E. Korf.** ***Planning as Search: A Quantitative Approach*****. Artificial Intelligence, 33(1):65--88, 1987.**

References

[H01] J. Hoffman. *FF: The Fast-Forward Planning System*. AI Magazine, 22:57– 62, 2001.

[M91] P. Maes. *The agent network architecture (ANA)*. SIGART bulletin, 2(4):115-120, 1991.

[MR91] D. McAllester, D. Rosenblitt. *Systematic Nonlinear Planning.* Proceedings of AAAI-91, 1991.

[M97] J. Muller. *A Cooperation Model for Autonomous Agents*. In J.P. Muller, M. Wooldridge, and N.R. Jennings, eds., Intelligent Agents III (LNAI Volume 1193), pp.245-260, Springer-Verlag: Berlin, Germany, 1997.

References

[NCLM99] D. Nau, Y. Cao, A. Lotem, H. Muqoz-Avila. *SHOP: Simple Hierarchical Ordered Planner*. Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 99), pp.968-973, Morgan Kaufmann, 1999.

[P92] J.S. Penberthy. *UCPOP: A Sound, Complete, Partial Order Planner for ADL.* Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR92), 1992.

[RN04] S. Russel, and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2004.

References

[S77] E. Sacerdoti. *A Structure for Plans and Behaviour*. Elsevier, New York, 1977.

[W00] M. Wooldridge. *Intelligent Agents*. In G. Weiss, eds., Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, 1999.

[WJ94] M. Wooldridge, and N.R. Jennings. *Agent Theories, Architectures, and Languages: A Survey.* Workshop on Agent Theories, Architectures and Languages (ECAI'94), 1994.


Any
question?