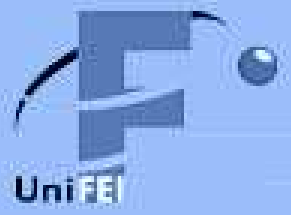


Centro Universitário da FEI

Reading PDDL, Writing Object-Oriented Model

Flavio Tonidandel
Centro Universitário da FEI
Tiago Vaquero
José Reinaldo Silva
Escola Politécnica USP

Planning Overview

There are many sophisticated planning systems

- Working with: Non-deterministic, numerical, probabilistic and many others planning domains

There is a language to describe planning domains and problems

- PDDL (Planning Domain Definition Language)

Many planning domains are very closed to real applications

Planning Overview

However….

- Describe real domains in PDDL is very difficult
- There are no tool for modeling support
- There are no method to validate or analyze planning domain models
- So… how could be possible to use planning systems in real life if we have the problems described above ?

PDDL is not a good language to describe domains or for a tool that aims to validate or analyze domain models.

We need to use a more intuitive, powerful, and general language to describe planning domains

Example of a PDDL file

```
(define (domain BLOCKS)
  (:requirements :strips :typing)
  (: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)))))
```

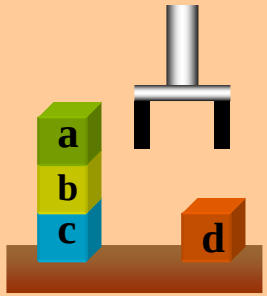
Classical Blocks World Domain - Example

- Action:

unstack(X,Y)

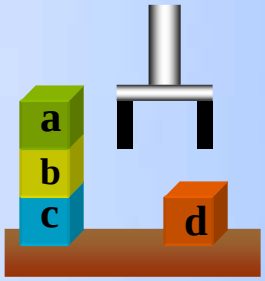
prec : on(X,Y) ∧ clear(X) ∧ handempty
delete list: on(X,Y) ∧ clear(X)) ∧ handempty
add list: holding(X) ∧ clear(Y)

Initial State S_0



on(a,b)
on(b,c)
ontable(d)
ontable(c)
clear(a)
clear(d)
handempty

S_0 — unstack (a,b) → S_1



Verify prec $\subseteq S_0$

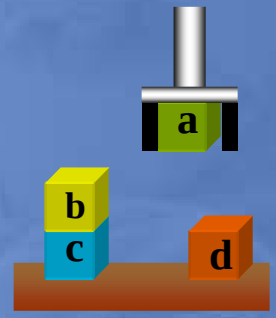
on(a,b)
on(b,c)
ontable(d)
ontable(c)
clear(a)
clear(d)
handempty

Delete List

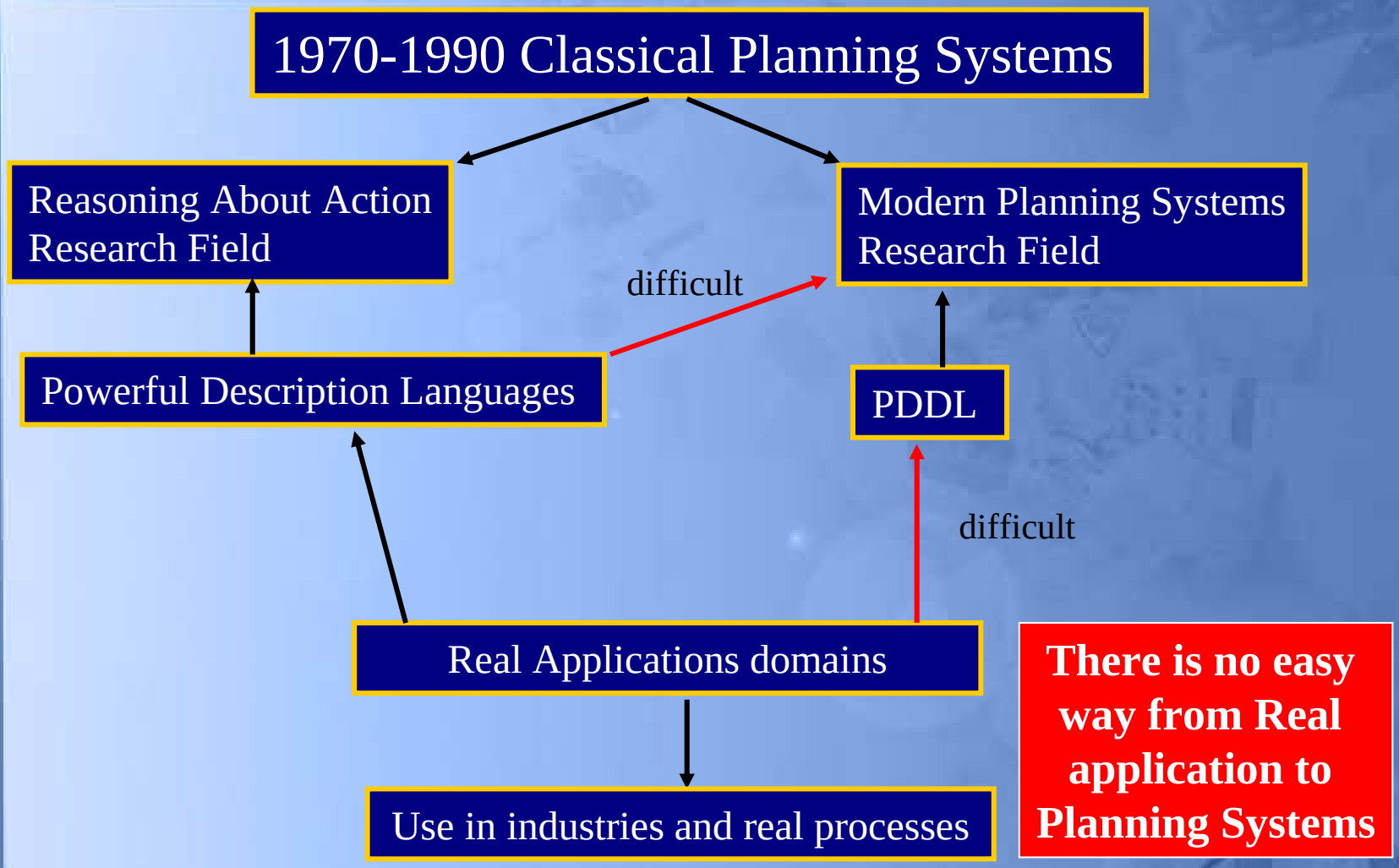
~~**on(a,b)**~~
on(b,c)
ontable(d)
ontable(c)
~~**clear(a)**~~
clear(d)
~~**handempty**~~

Add List

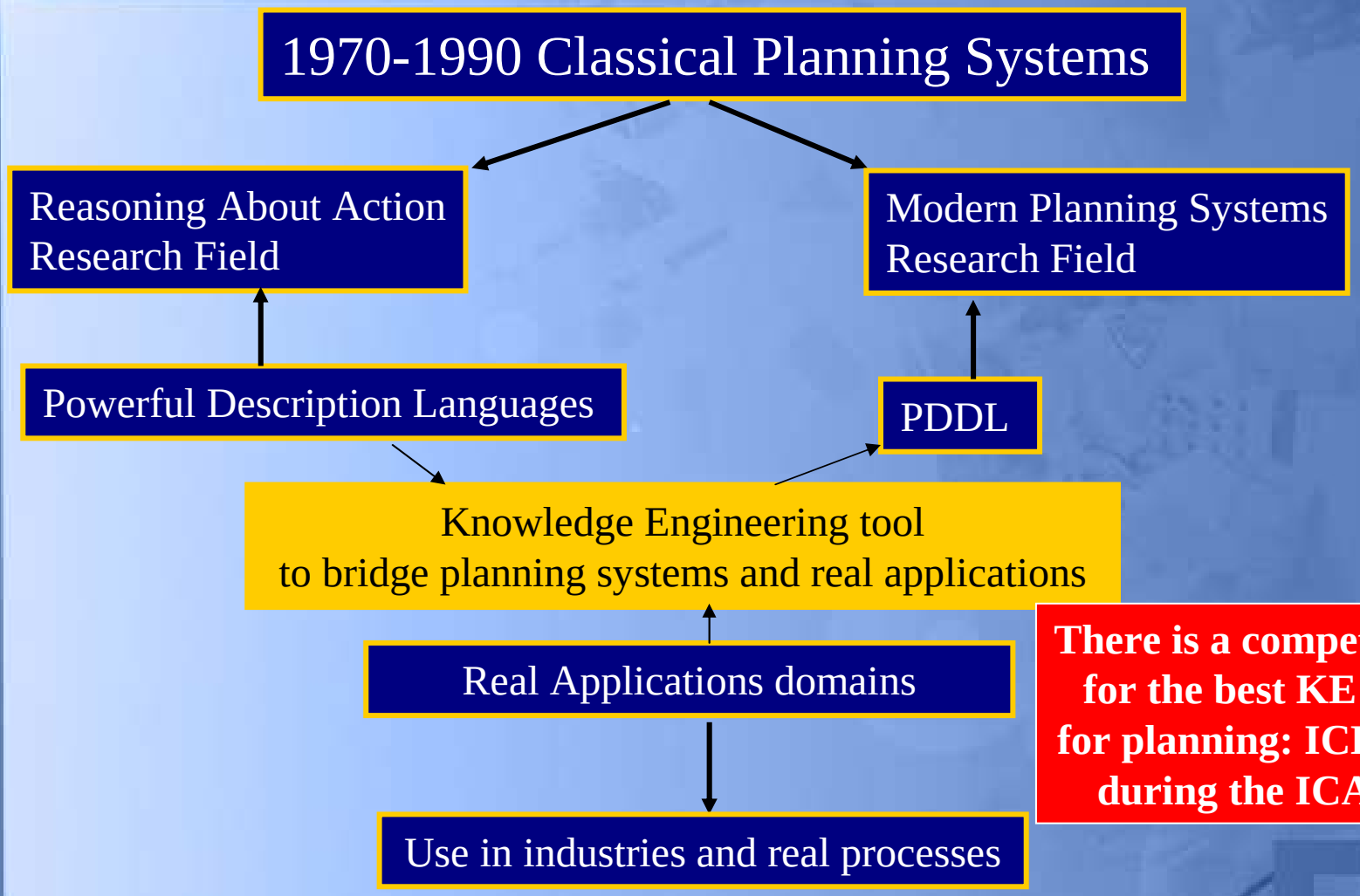
on(b,c)
ontable(d)
ontable(c)
clear(d)
holding(a)
clear(b)



Knowledge Engineering for Planning

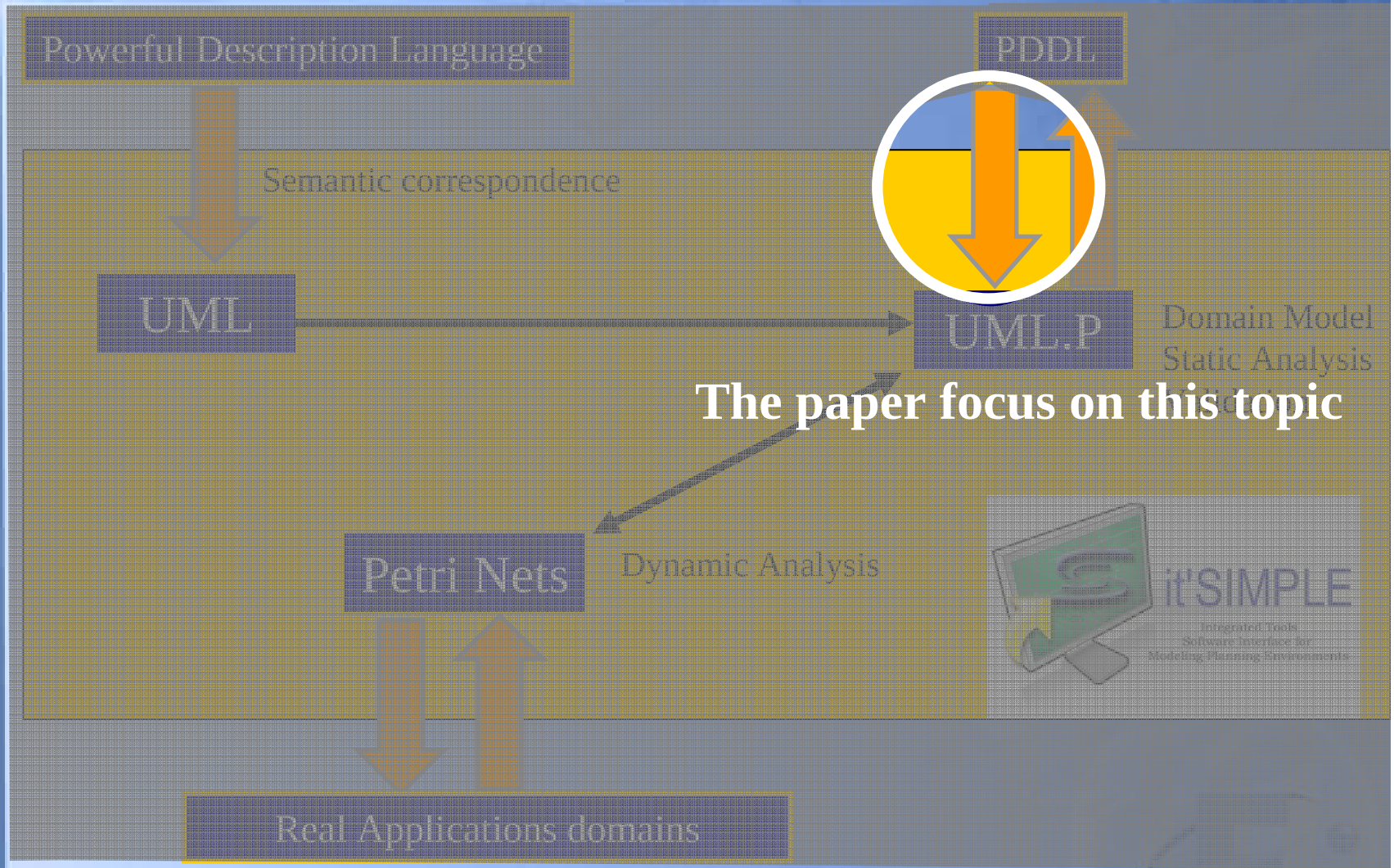


Knowledge Engineering for Planning



There is a competition for the best KE tool for planning: ICKEPS during the ICAPS

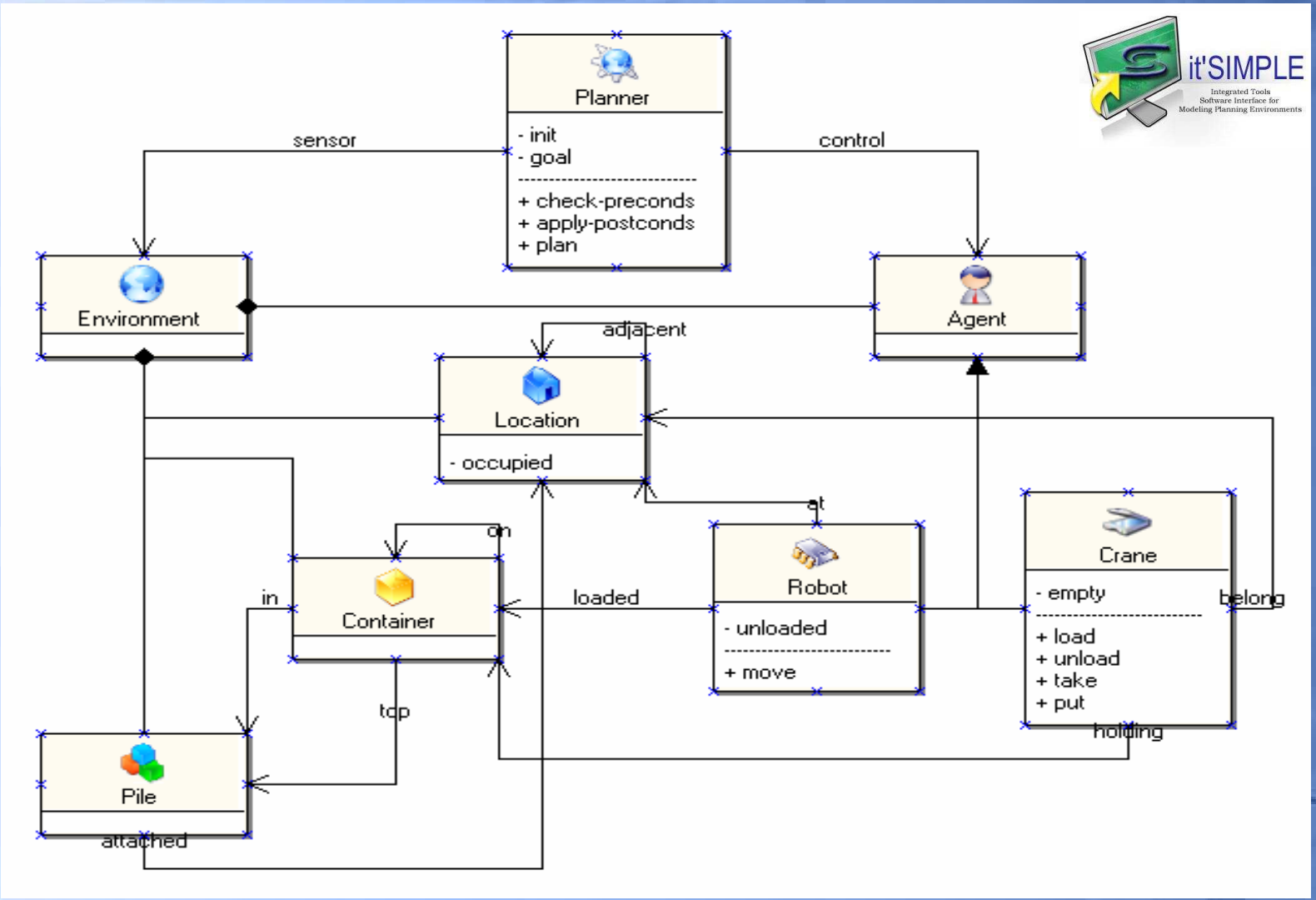
Knowledge Engineering tool



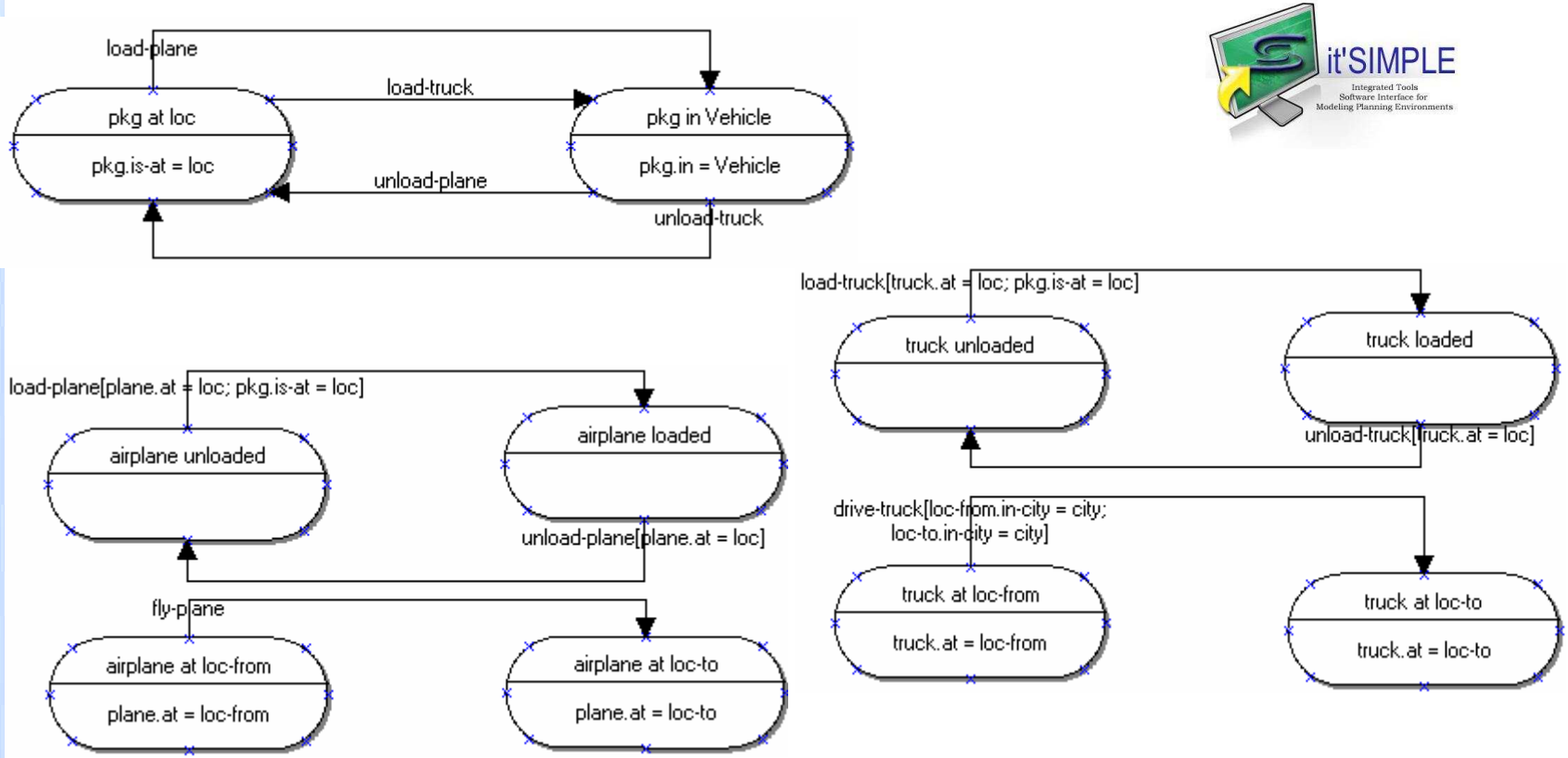
UML.P

- **UML.P → UML for Planning Approach**
  - Approximate UML to Planning
- **Class Diagram**
  - Describe objects and their relation
- **StateChart Diagram**
  - Describe Planning Actions
- **Object Diagram**
  - Describe Planning States

UML.P Class Diagram Example

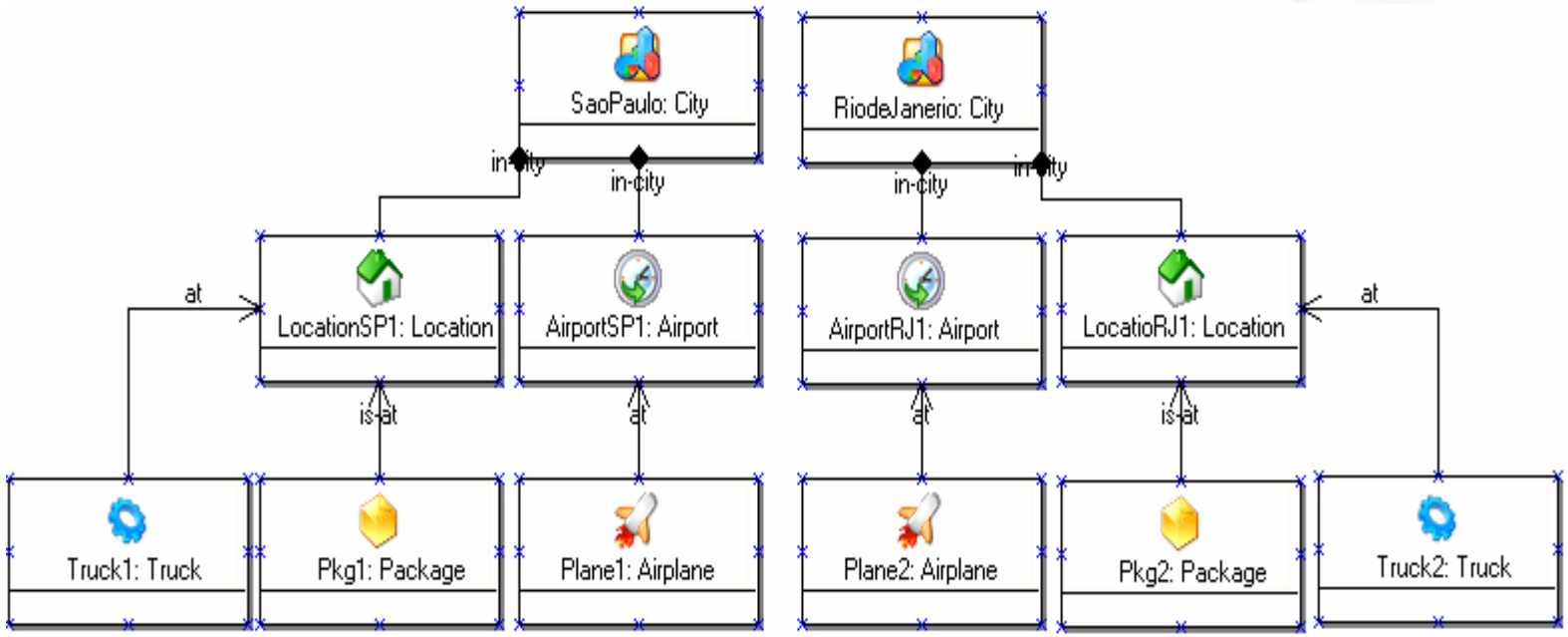


UML.P StateChart Diagram Example



Describe Planning Actions
Transition of States

UML.P Object Diagram Example

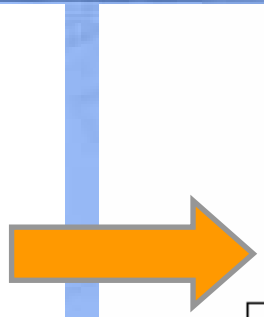


Describe Valid States
Class Diagram Correspondence

From PDDL to UML.P – Class Diagram

```
(:types Hand
        Block
        Table - object)

(:predicates (on ?x - block ?y - block)
             (ontable ?x – block ?t - table)
             (clear ?x - block)
             (handempty ?h - Hand)
             (holds ?h - Hand ?x - block)
            )
```



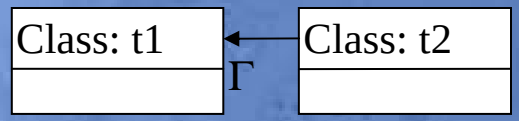
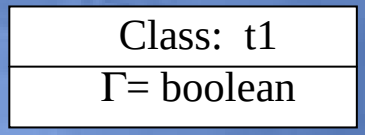
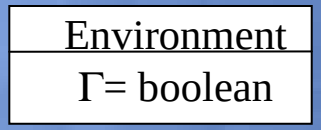
Translating PDDL → Class Diagram in UML.P

We can define Class Diagram from :types and :predicates

- **Classes (from :types),**
- **aggregations (from 2-arity predicate)**
- **Properties of classes (from 1-arity predicate)**
- **0-arity predicate → a property of Planner super-Class**

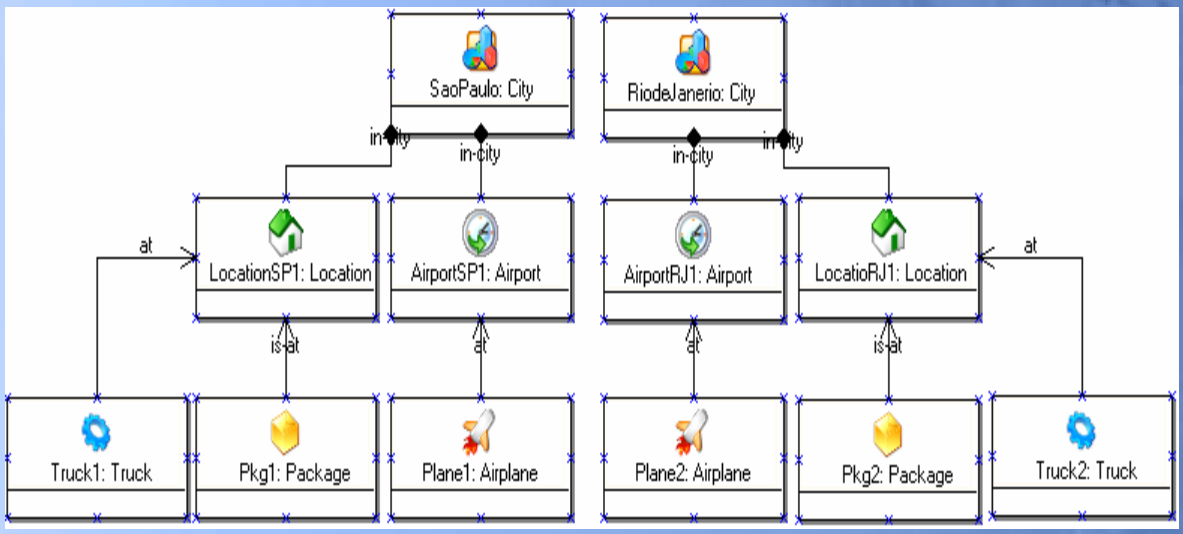
From PDDL to UML.P – Class Diagram

Table 1 – The correspondence between PDDL and UML for predicates

Predicate in PDDL	UML description	Graphical UML
(Γ ?v1 – t1 ?v2 –t2) Arity/2	Γ.v1 = v2	
(Γ ?v1 – t1) Arity/1	Γ.v1 = true or Γ.v1 = false	
Γ Arity/0	Γ = true or Γ = false	

From PDDL to UML.P – Object Diagram

- For the Object Diagram... PDDL has an specific file to describe problems
- The translation to UML is similar to Class Diagram (using table 1 as well)



From PDDL to UML.P – StateChart Diagram

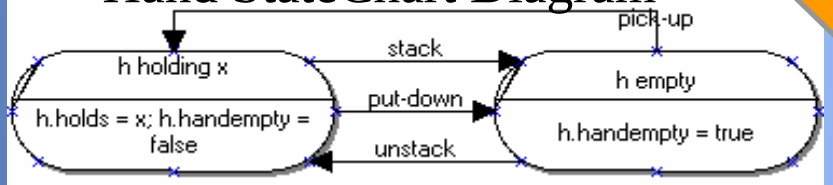
```
(:action pick-up
        :parameters (?x - block)
        :precondition (and (clear ?x) (ontable ?x)
                           (handempty))
        :effect (and (not (ontable ?x)) (not (clear ?x))
                     (not (handempty)) (holding ?x)))

(:action put-down
        :parameters (?x - block)
        :precondition (holding ?x)
        :effect (and (not (holding ?x)) (clear ?x)
                     (handempty)  (ontable ?x)))
```

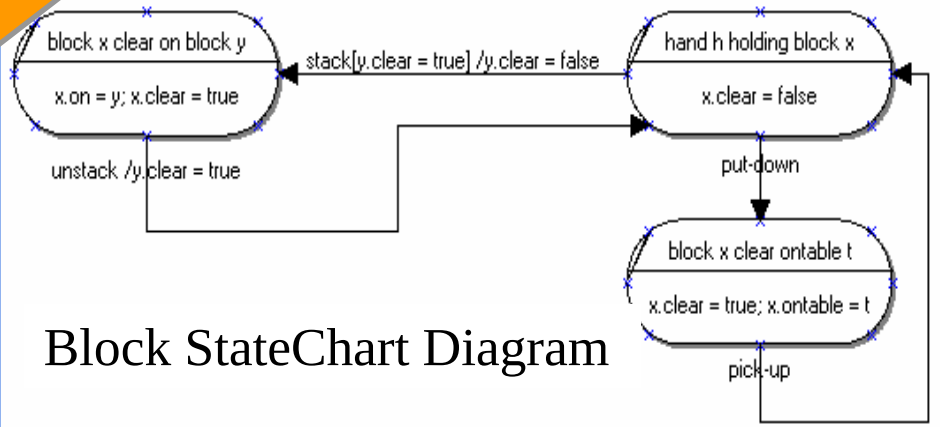
```
(:action stack
        :parameters (?x - block ?y - block)
        :precondition (and (holding ?x) (clear ?y))
        :effect  (and (not (holding ?x)) (not (clear ?y))
                      (clear ?x)  (handempty)  (on ?x ?y)))

(: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)))))
```

Hand StateChart Diagram

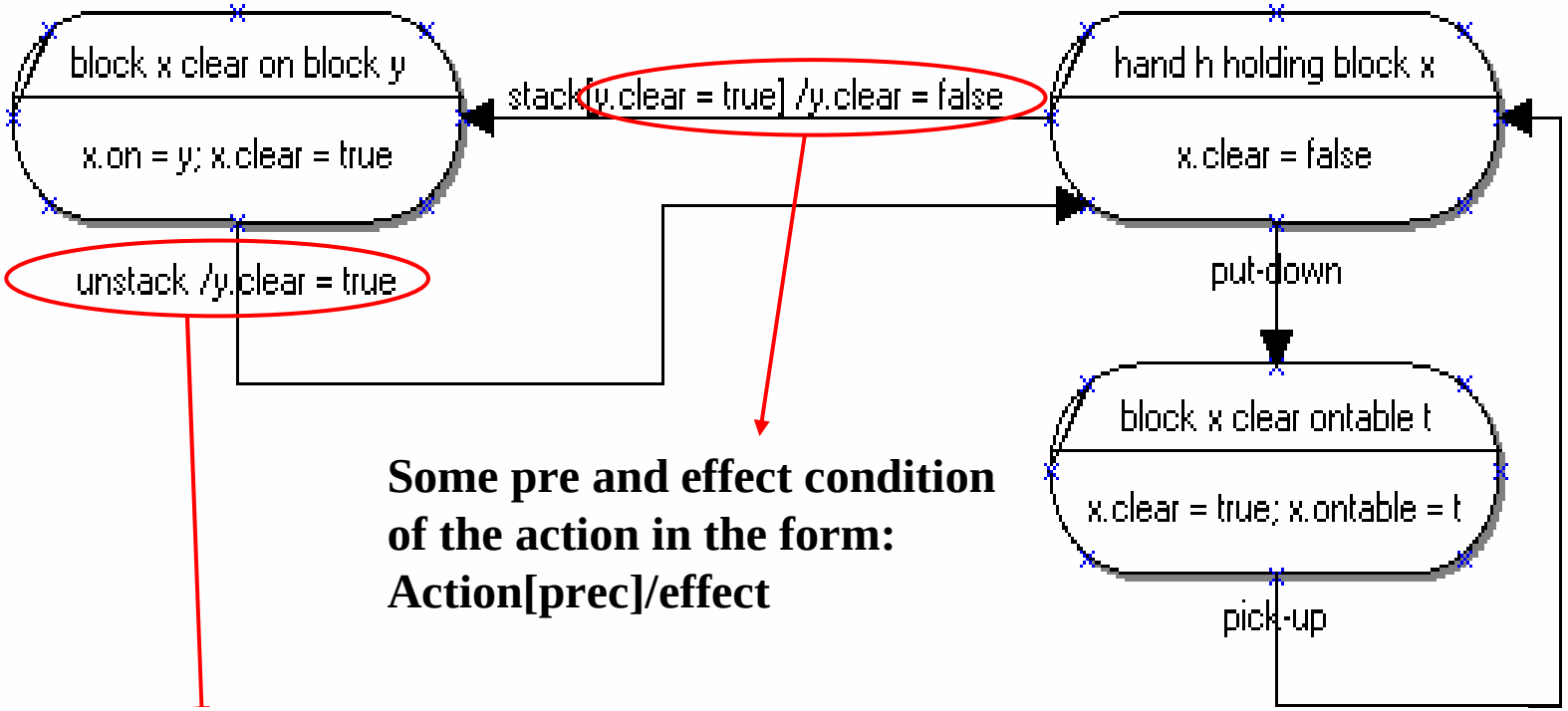


A StateChart Diagram:
-Defines States and Transitions (action)
-Relies on one object (class)
- One action can appear in many StateChart diagrams



Block StateChart Diagram

From PDDL to UML.P – StateChart Diagram



Some pre and effect condition of the action in the form: Action[prec]/effect

For the action unstack:
x.on=y and x.clear=true → Precondition and effect (will be deleted)
x.clear=false →effect
A state before an action tells us about Precondition and effects.
A state after an action tells us about only the effects

From PDDL to UML.P – StateChart Diagram

In order to define some StateChart Diagrams from PDDL actions, we have to state some definitions:

Predicate arity

Pi = predicate with arity i / $0 \leq i \leq 2$

Consider an action of the form:

:action α

:parameters (?v1 – tp1 ?v2 – tp1 ?v3 – tp2 ?v4 – tp3 ?v5 – tp2)

Parameter List

PL = {(v1 :class1) (v2 : class1) (v3 : class2) (v4 : class3) (v5 : class2)}

Relevant Parameter List

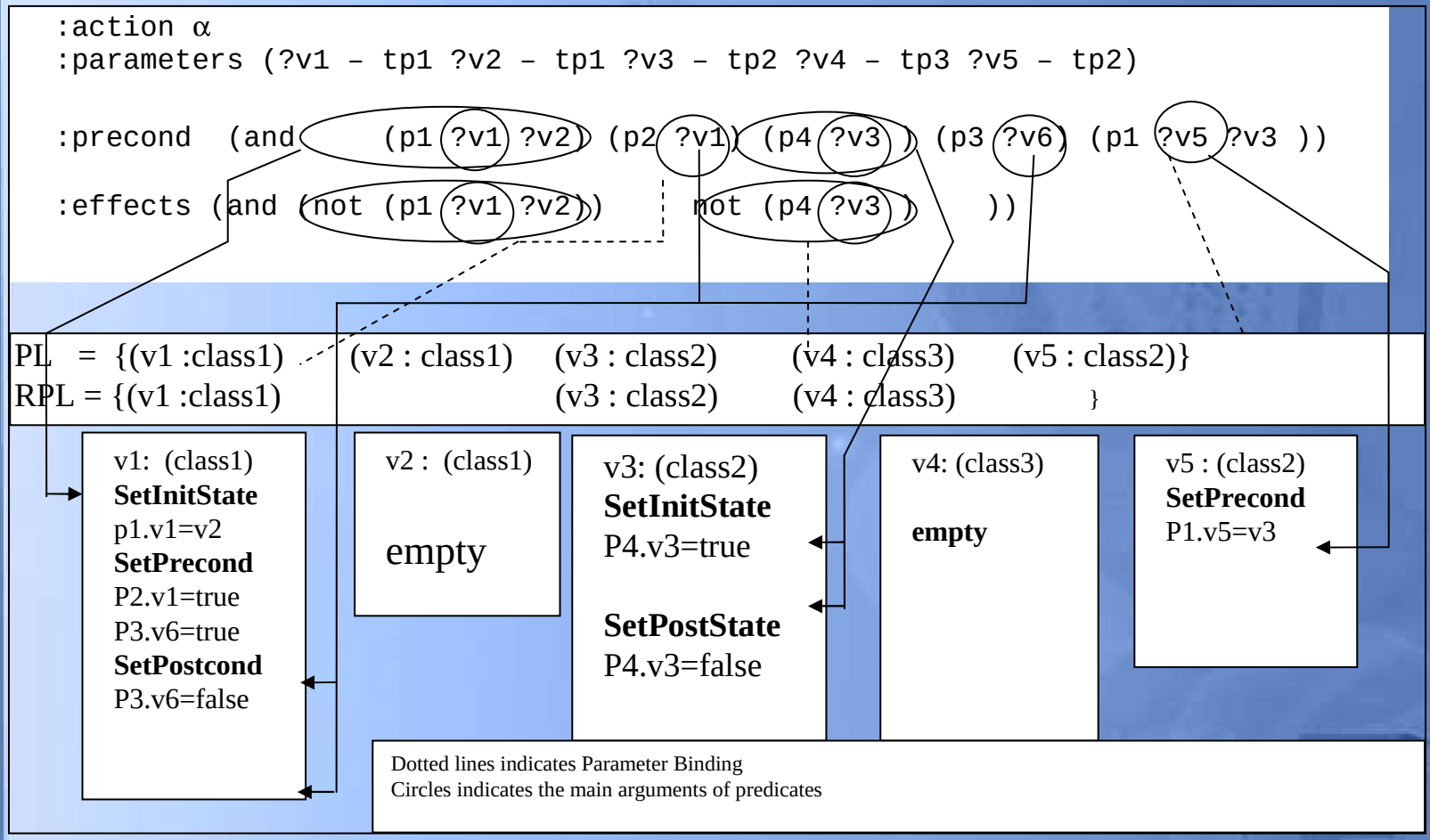
RPL = {(v1 :class1) (v3 : class2) (v4 : class3) }

→ The first occurrence of a class in the ordered parameter list … they are relevant to me !!!

→ Most relevant parameter for the action → v1:class1

From PDDL to UML.P – StateChart Diagram

Some details about formal definition and algorithms for this example can be seen in the paper.



From PDDL to UML.P – StateChart Diagram

Then... What we do now:

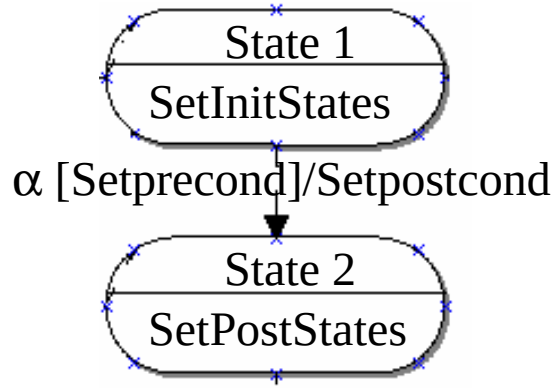
Which and how many StateChart Diagrams an action will appear ?

The action will have states in the StateChart Diagram of any class that the SetInitState or SetPostState are not empty for that action

For a class C and an action A

We join the SetInitStates, SetPostState, Setprecond and SetPostcond of all parameters of class C in action A

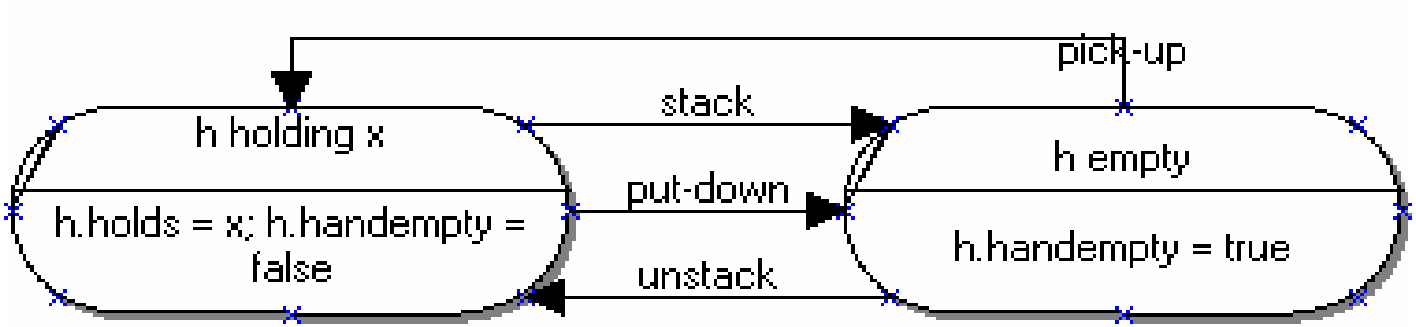
StateChart of the class C



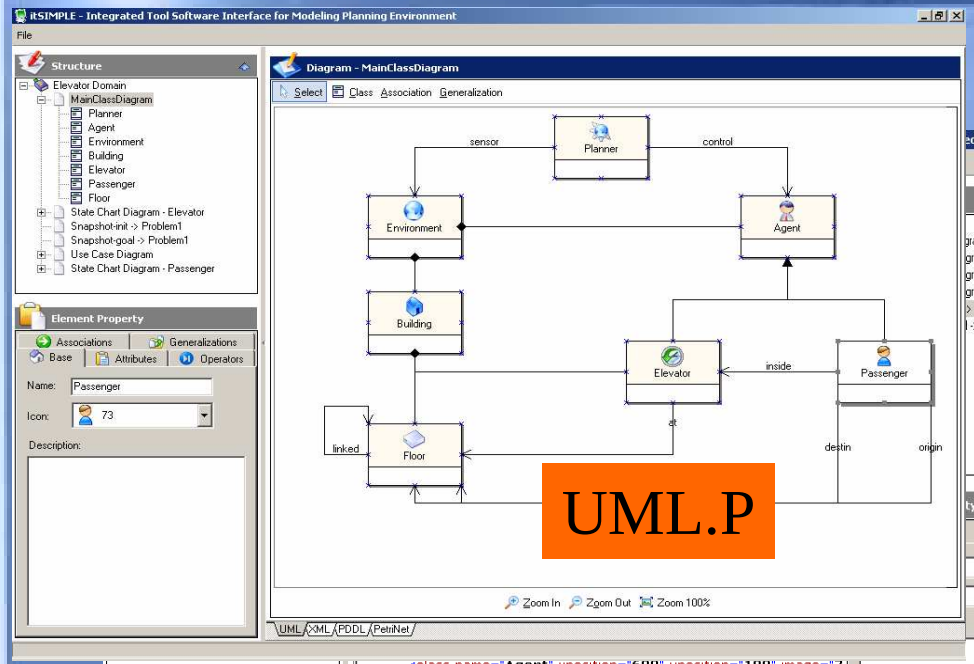
From PDDL to UML.P – StateChart Diagram

Finally…

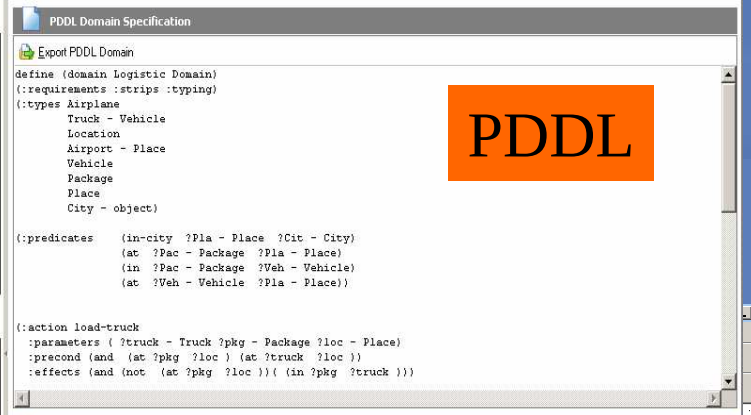
We will gather similar states in each StateChart Diagram in order to identify opposite or sequential actions:



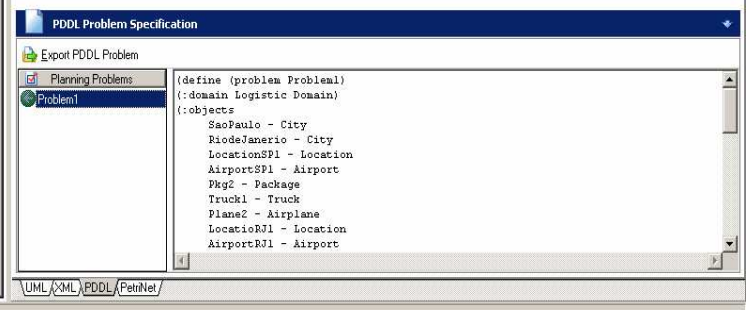
itSIMPLE tool

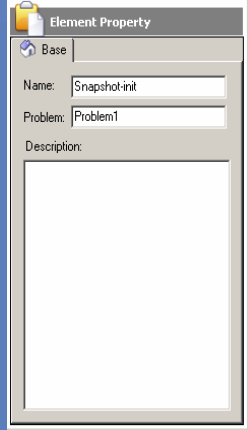


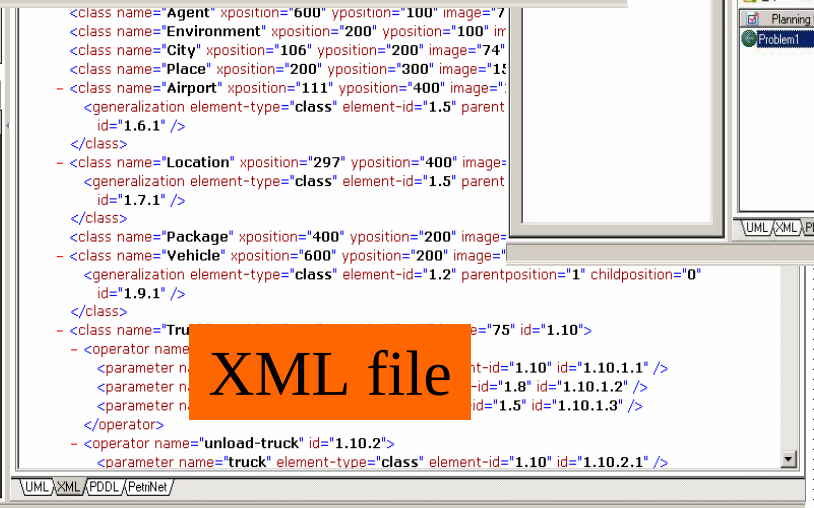
UML.P



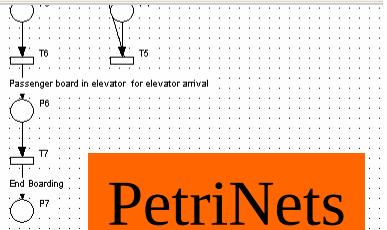
PDDL







XML file



PetriNets

Conclusion and Future Works

- This work defines a translation process of a STRIPS-like domain
- It is an important part of the entire work to define a KE tool for planning
- This paper will be extended to work with more complex PDDL descriptions
- It will be incorporated in the itSIMPLE tool for the next ICKEPS (International Competition on Knowledge Engineering for Planning and Scheduling) in 2007


UniFEI
Centro Universitário da FEI


Artificial Inteligence Applied to Automation


DESIGN LAB


USP
Universidade de São Paulo
BRASIL

That is it !!