Structuring processes into abilities : an information-oriented architecture for autonomous robots

Arnaud Degroote, Simon Lacroix
adegroot@laas.fr, simon@laas.fr

May 19, 2010

1 Context

2 Our proposal

3 Conclusion and future work

Presentation Plan

1 Context

2 Our proposal

3 Conclusion and future work

Work context

- The PEA project ACTION http://action.onera.fr

Work context

- The PEA project ACTION http://action.onera.fr
- In particular, controlling an UGV:

Work context

- The PEA project ACTION http://action.onera.fr
- In particular, controlling an UGV:
- Common tasks :
  - Exploring
  - Tracking target
  - Following target

Problem statement

Considering

- a set of robot capacities
- a (set of) mission(s)
- a more or less unknown environment

Problem statement

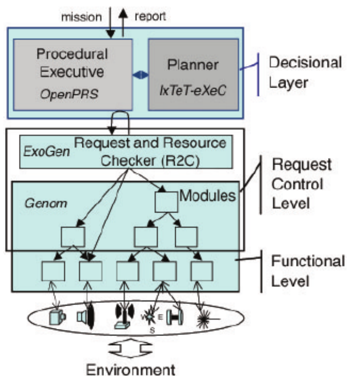
Considering

- a set of robot capacities
- a (set of) mission(s)
- a more or less unknown environment

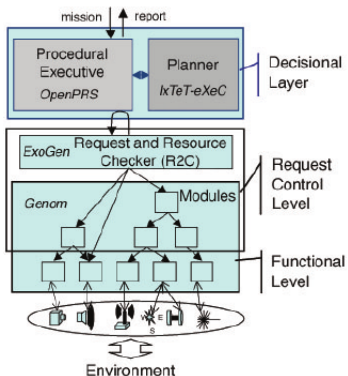
How to organize this different processes to achieve the mission ?

- in an efficient way
- in the most autonomous way

Related work : 3-layer architecture



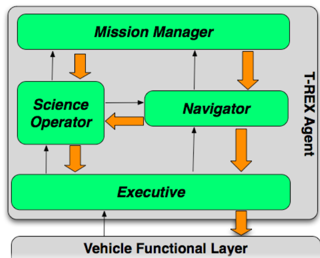
Related work : 3-layer architecture



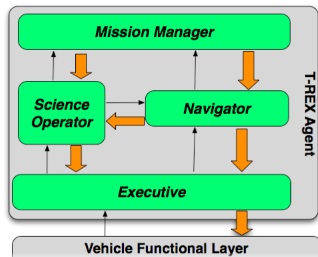
Issues with this approach

- Different representation between layers ⇒ difficult diagnostic
- Scalability issues
- Monolithic blocks

Related work : 2-layer architecture



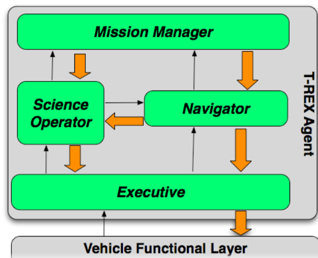
Related work : 2-layer architecture



Improvements

- Better interaction in decision layer
- Problem partition $\Rightarrow$ better reactivity

Related work : 2-layer architecture



Improvements

- Better interaction in decision layer
- Problem partition $\Rightarrow$ better reactivity

Issues

- non-composable decomposition

Requirements

Requirements

- Reactive to external event, "adaptable"

Requirements

Requirements

- Reactive to external event, "adaptable"
- Robust to single component failure

Requirements

Requirements

- Reactive to external event, "adaptable"
- Robust to single component failure
- Generic, modular, extensible

Requirements

Requirements

- Reactive to external event, "adaptable"
- Robust to single component failure
- Generic, modular, extensible
- Verifiable, "provable"

Presentation Plan

1 Context

2 Our proposal

3 Conclusion and future work

Splitting up into information

What is an information ?

- robot internal state
- possible futures of the robot (i.e. plan)
- data from the outside world, processed or not.

Splitting up into information

What is an information ?

- robot internal state
- possible futures of the robot (i.e. plan)
- data from the outside world, processed or not.

Why partitioning on top of information ?

- Information is a first-class object of the functional layer
- Competition on information acquisition
- Information is robot-agnostic

Ability notion

Definition

- Each piece of information is encapsulated in an *ability*
- This information can be constrained by some free variables

Ability notion

Definition

- Each piece of information is encapsulated in an *ability*
- This information can be constrained by some free variables

A more formal definition

$$Ability = < N_a, S_r, S_w, S_i, S_T, S_F, S_{Ta}, S_{rt} >$$

Ability notion

Definition

- Each piece of information is encapsulated in an *ability*
- This information can be constrained by some free variables

A more formal definition

$$Ability =< N_a , S_r, S_w, S_i , S_T, S_F , S_{Ta}, S_{rt} >$$

Identifier

- N_a : ability identifier

Ability notion

Definition

- Each piece of information is encapsulated in an *ability*
- This information can be constrained by some free variables

A more formal definition

$$Ability =< N_a , S_r, S_w, S_i , S_T, S_F , S_{Ta}, S_{rt} >$$

Context

- S_r : Readable variables, real information
- S_w : Writable variables, control ability behaviour
- S_i : Internal variables

LAAS

Ability notion

Definition

- Each piece of information is encapsulated in an *ability*
- This information can be constrained by some free variables

A more formal definition

$$Ability =< N_a , S_r, S_w, S_i , S_T, S_F , S_{Ta}, S_{rt} >$$

Programmation environment

- S_T : a set of types
- S_F : a set of functions, which works on previously defined S_T

Ability notion

Definition

- Each piece of information is encapsulated in an *ability*
- This information can be constrained by some free variables

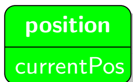
A more formal definition

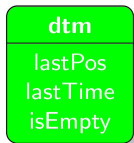
$$Ability =< N_a , S_r, S_w, S_i , S_T , S_F , S_{Ta}, S_{rt} >$$

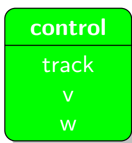
Tasks

- S_{Ta} : a set of tasks
- S_{rt} : a set of relation between tasks

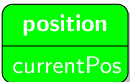
Abilities network

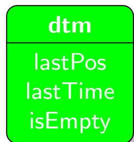


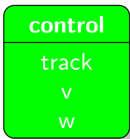




Abilities network





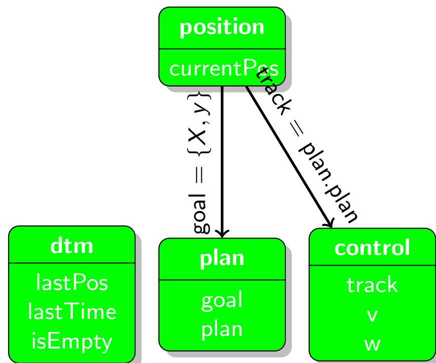


Run-time support

- Nameserver function
- Gives the list of running abilities
- Monitors the status of each ability

runtime

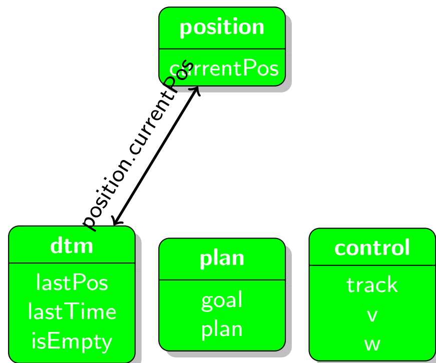
Abilities network



Control flow

- Asynchronous protocol
- Ends with
  - OK message
  - FAILURE message
  - KILLED message

Abilities network



Data flow

1. Fed using an asynchronous request / reply
2. Future update uses a publisher / subscriber interface

Tasks and recipes

Tasks

- Define how to pass from a symbolic state to another one
- Mainly defined by their pre and post-condition

LAAS

Tasks and recipes

Tasks

- Define how to pass from a symbolic state to another one
- Mainly defined by their pre and post-condition

Recipes

- Implement different strategies for one task
  - Recover from local error
  - Use different strategies for the same task

LAAS

Pre / Post-condition description

Pre / Post-condition

- Boole logic formula of clauses
- Clauses are comparison or function application on variables and constants

Pre / Post-condition description

Pre / Post-condition

- Boole logic formula of clauses
- Clauses are comparison or function application on variables and constants

```
        * (size < 42)
        * (pos::distance(pos::curentPosition, planning::goal) < threshold)
```

Recipe definition

Recipe definition

- **make** < constraint > : constrain a remote ability (one-shot)
- **ensure** < constraint > : constraint a remote ability (continuous)
- **wait** < condition > : wait until the condition is true
- classical loop and branch mechanisms

Recipe definition

Recipe definition

- **make** $<$ constraint $>$: constrain a remote ability (one-shot)
- **ensure** $<$ constraint $>$: constraint a remote ability (continuous)
- **wait** $<$ condition $>$: wait until the condition is true
- classical loop and branch mechanisms
- More work is required to define a clear semantic

Recipe definition

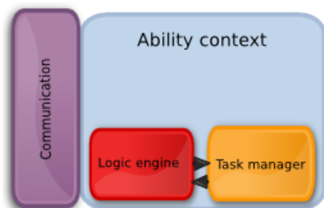
Recipe definition

- **make** $<$ constraint $>$: constrain a remote ability (one-shot)
- **ensure** $<$ constraint $>$: constraint a remote ability (continuous)
- **wait** $<$ condition $>$: wait until the condition is true
- classical loop and branch mechanisms
- More work is required to define a clear semantic

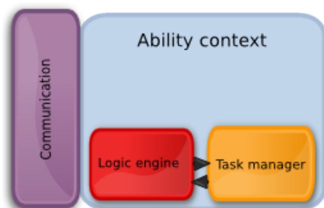
```
3DPath_handle_failure = recipe {
        Pre = {{ 3DPathFailure == true }}
        Body = {
        /* Get some new information */
        make ( pos::computeDistance(Dtm::lastMerged, pos::currentPostion) == 0.0))
        /* Restart the classic processing */
        ensure ( pos::computeDistance(Dtm::lastMerged, Pos::currentPosition) < threshold)
                 && 3DPath::goal == currentGoal
                 && Control::tracker == 3DPath::plan)
        wait(pos::computeDistance(goal, Pos::currentPosition) < goalThreshold))
        }
}
```

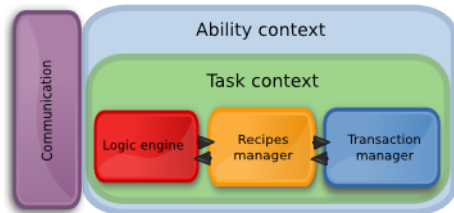
LAAS

Overview of ability and task implementation



Overview of ability and task implementation





Logic engine

Goal

Find a set of tasks to achieve new constraints

Logic engine

Goal

Find a set of tasks to achieve new constraints

When

- When receiving new constraints from outside
- to recover from a task failure

Logic engine

Goal

Find a set of tasks to achieve new constraints

When

- When receiving new constraints from outside
- to recover from a task failure

How

1. Rules selection on variable type

Logic engine

Goal

Find a set of tasks to achieve new constraints

When

- When receiving new constraints from outside
- to recover from a task failure

How

1. Rules selection on variable type
2. Classical unification / rules selection / backtracking

Logic engine

Goal

Find a set of tasks to achieve new constraints

When

- When receiving new constraints from outside
- to recover from a task failure

How

1. Rules selection on variable type
2. Classical unification / rules selection / backtracking
3. Heuristic used on the base of:
   - number of pre-condition not fulfilled
   - number of tasks involved
   - number of tasks "not runnable"

Transaction manager

Goal

1. Handle lazily complex constraints
2. Handle switch between recipes

Transaction manager

Goal

1. Handle lazily complex constraints
2. Handle switch between recipes

Handling complex constraints

- Register each constraints
- When receiving part of answer, reduce the global expression
- Terminate when expression is reduced to true or false

Transaction manager

Goal

1. Handle lazily complex constraints
2. Handle switch between recipes

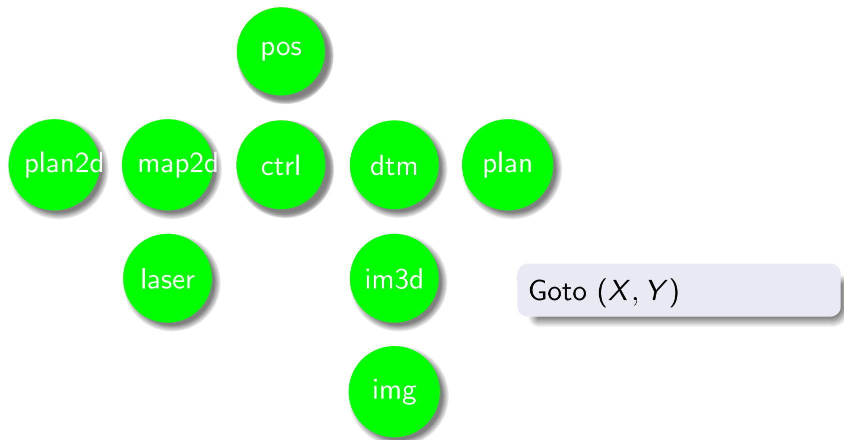
Handling complex constraints

- Register each constraints
- When receiving part of answer, reduce the global expression
- Terminate when expression is reduced to true or false

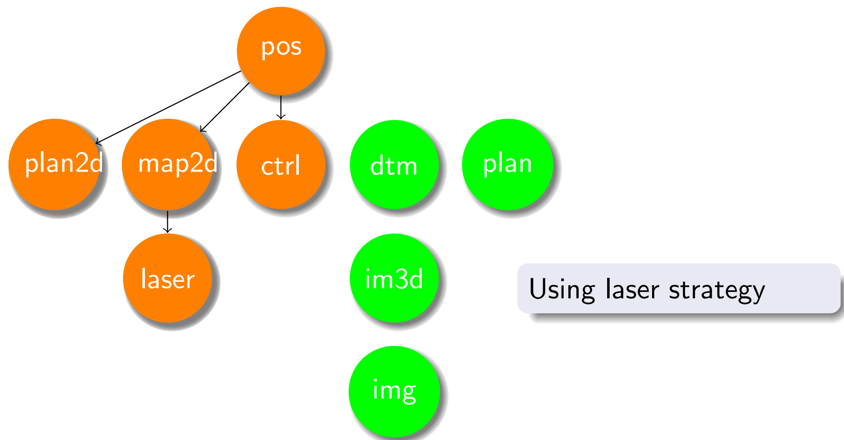
Recipes switch

- Recipes classification
- Cleaning task context or handling it transparently ?

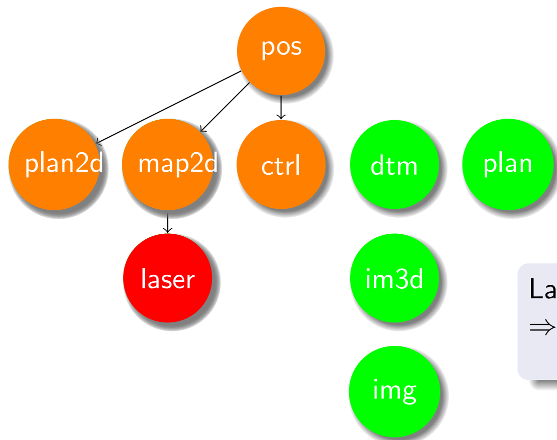
Illustration



Illustration

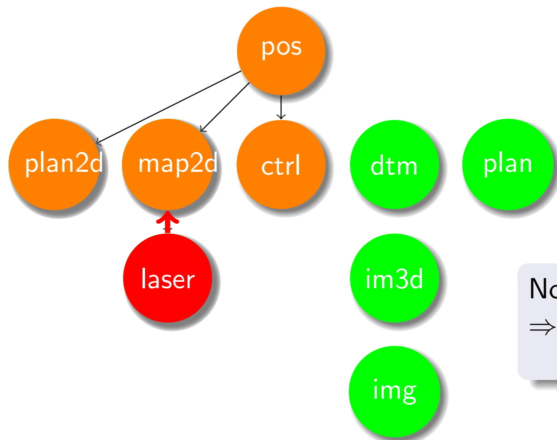


Illustration



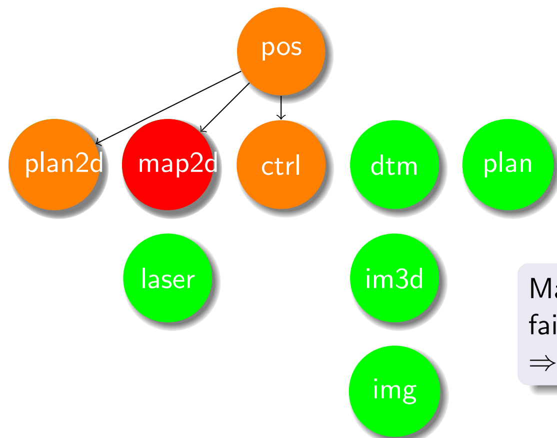
Laser fails
⇒ another recipe ?

Illustration



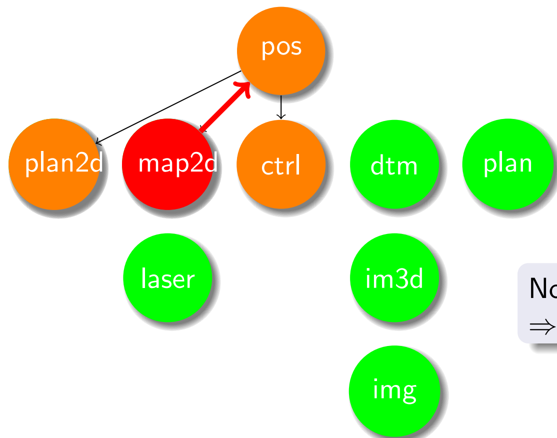
No other recipe
⇒ Another task set ?

Illustration



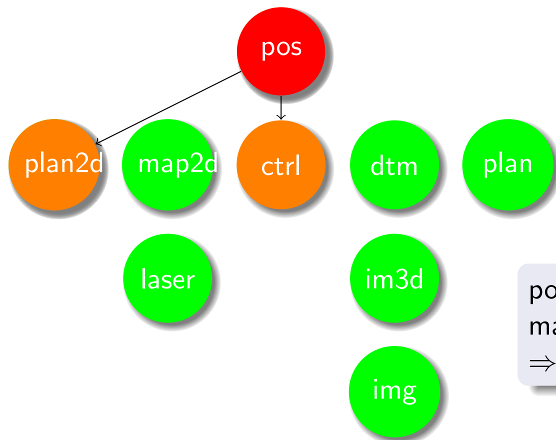
Map2d fails due to laser failure
$\Rightarrow$ another recipe ?

Illustration



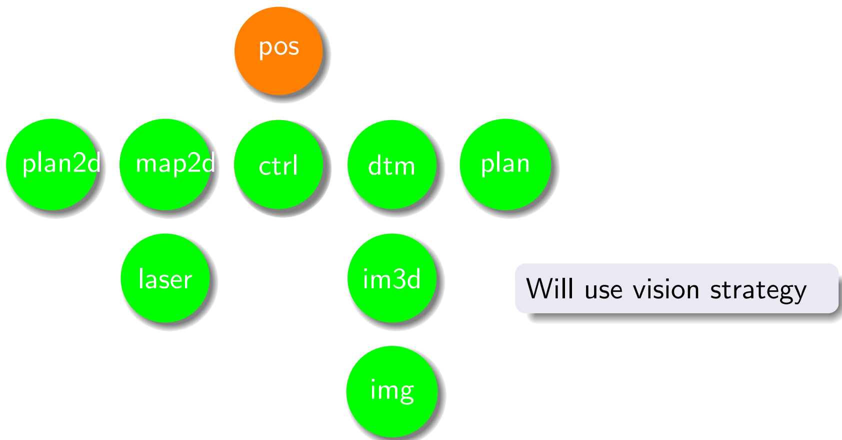
No other recipe
⇒ Another task set ?

Illustration

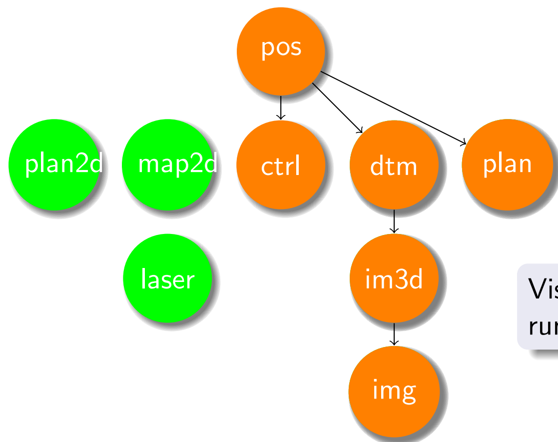


position fails due to map2d failure
$\Rightarrow$ another recipe ?

Illustration



Illustration



Vision strategy is running

Presentation Plan

1 Context

2 Our proposal

3 Conclusion and future work

Conclusion

Requirements

- Reactive to external event, "adaptable" :
  - logic engine

Conclusion

Requirements

- Reactive to external event, "adaptable" :
  - logic engine
- Robust to single component failure :
  - Separate process, run-time support

Conclusion

Requirements

- Reactive to external event, "adaptable" :
  - logic engine
- Robust to single component failure :
  - Separate process, run-time support
- Generic, modular, extensible :
  - Decomposition on information, decomposition on different layer

Conclusion

Requirements

- Reactive to external event, "adaptable" :
  - logic engine
- Robust to single component failure :
  - Separate process, run-time support
- Generic, modular, extensible :
  - Decomposition on information, decomposition on different layer
- Verifiable, "provable"
  - To Be Done :D

Inspiration

Various programming languages

- Erlang (concurrency)
- Datalog / Prolog (logic)
- Oz / Mozart, Alice (functional / concurrent / logic)

Inspiration

Various programming languages

- Erlang (concurrency)
- Datalog / Prolog (logic)
- Oz / Mozart, Alice (functional / concurrent / logic)

Processes algebras

- CSP (Communicating Sequential Process)
- CCP (Concurrent Constraint Programming)
- TCCP and UTCCP (Temporal Concurrent Constraint Programming)

Future work

- Implement completely the proposed solution, and test it on a real robot

Future work

- Implement completely the proposed solution, and test it on a real robot
- Improve the formalism : make some link with classic processes algebra like CSP, CCP or more recently UTCC

Future work

- Implement completely the proposed solution, and test it on a real robot
- Improve the formalism : make some link with classic processes algebra like CSP, CCP or more recently UTCC
- Enhance the framework to deal with multi-robot problematic