A Software Component for Plan Management in Robotics

Sylvain Joyeux

Thèse préparée au LAAS/CNRS

6 December, 2007

1. Introduction

Autonomous robotic systems

I can't define a robot, but I know one when I see one

J. Engelberger

Autonomous robotic systems

robotic systems electro-mechanical systems able to sense their environment and act in it

autonomous able to decide its course of action in order to perform a mission

i.e. embodied sense-plan-act loops

Robotic architectures

In software development, an architecture

- separates functions into components
- defines the interactions between those components

- solves the common problems
- must allow integration to solve the not-so-common ones
- in robotics, two main concerns:
  - modular integration of diverse data processing services
  - integration of decision-making

Robotic architectures

In software development, an architecture

- separates functions into components
- defines the interactions between those components

- solves the common problems
- must allow integration to solve the not-so-common ones
- in robotics, two main concerns:
  - modular integration of diverse data processing services
  - integration of decision-making

Decision-making

Two kinds of decision-making:

reactive given the situation and the goals, what should be the next step.

Decision-making

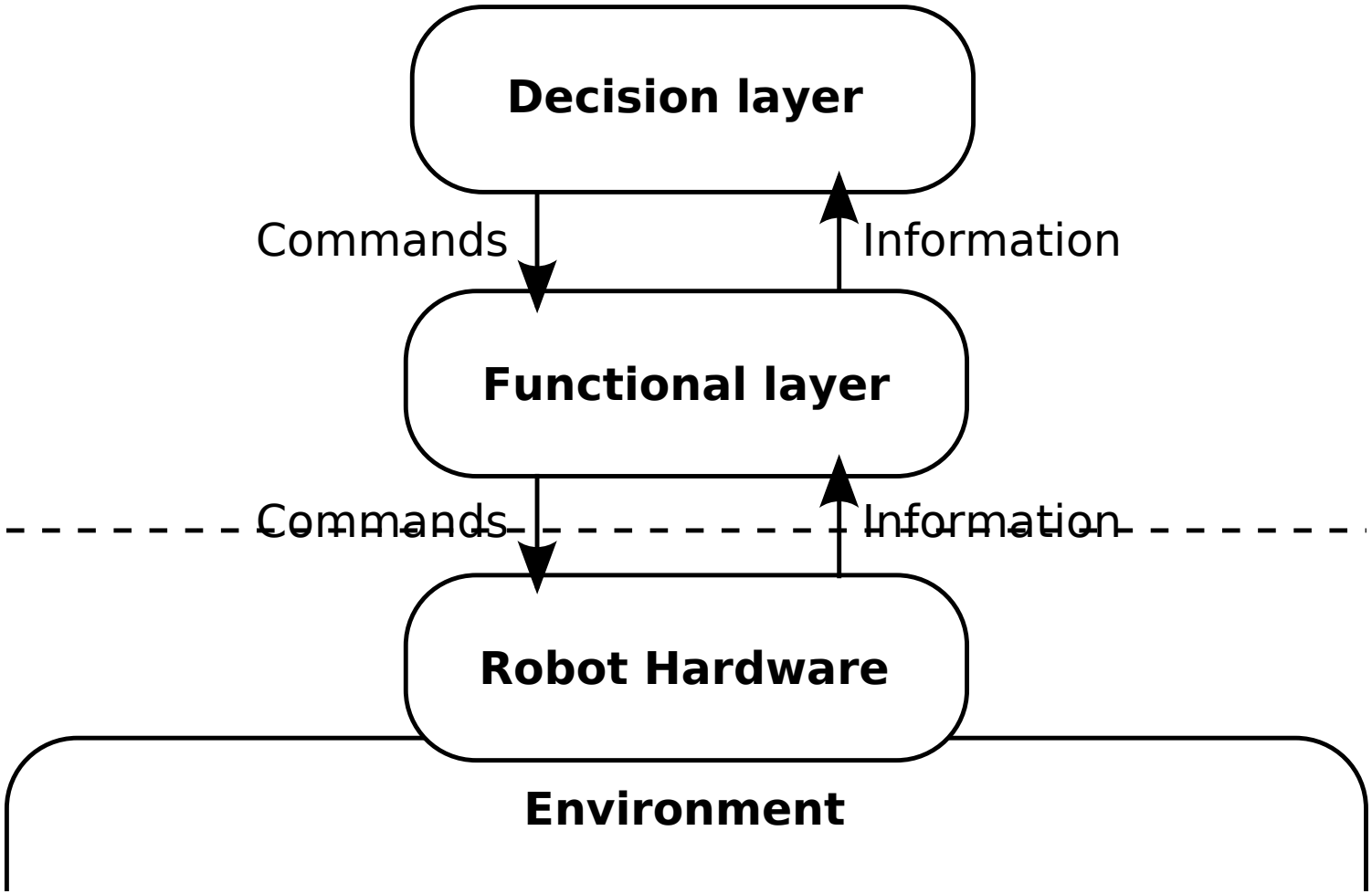
Two kinds of decision-making:

reactive given the situation and the goals, what should be the next step.

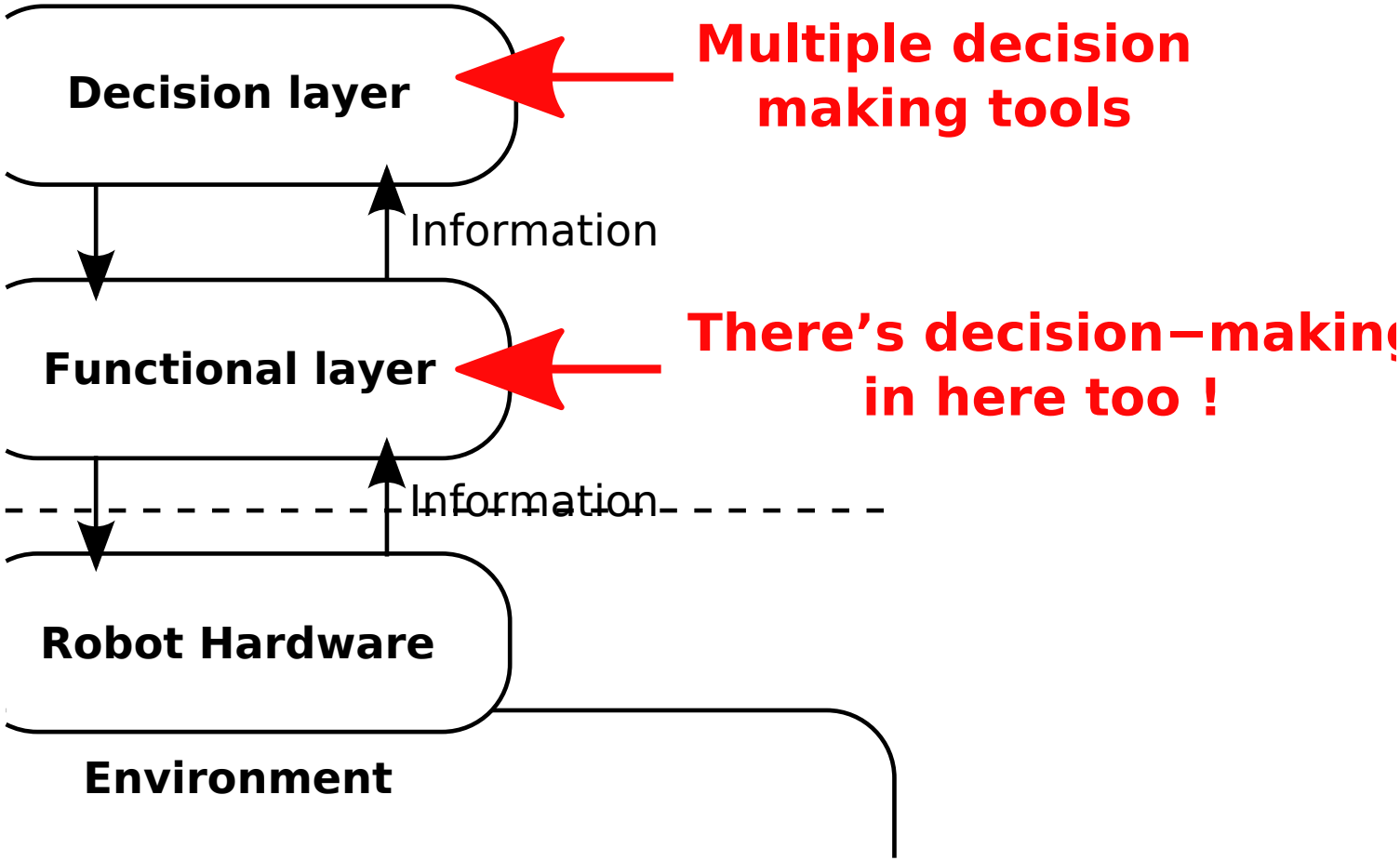
by anticipation given the situation and the goals,

- explore the possible futures
- determine the “desired” ones
- determine the set(s) of actions required to reach them
  = the plan(s)

The two-layer model



The two-layer model



Focus of this thesis
Integration of decision-making

Our focus

An architecture allowing . . .

the cooperative integration of heterogeneous decision-making tools

Focus of this thesis
Integration of decision-making

Our focus

An architecture allowing . . .
the cooperative integration of heterogeneous decision-making tools

heterogeneous — choose the right planner for the task

- because there is no one-fits-all planner
- because planning is complex
  - $\Rightarrow$ split big problems into smaller ones

Focus of this thesis
Integration of decision-making

Our focus

An architecture allowing . . .
the cooperative integration of heterogeneous decision-making tools

heterogeneous — choose the right planner for the task

- because there is no one-fits-all planner
- because planning is complex
  - ⇒ split big problems into smaller ones

cooperative integration — through planner-planner negotiation

⇒ one planner's decision must not overconstrain the others'

Integration through partial plans

Alami et al. – 1998
An Architecture for Autonomy

Muscettola et al. – 2002
IDEA: Planning at the Core of Autonomous Agents

Chouinard et al. – 2005
Intelligent rover decision-making in response to exogenous events

- multiple planners
- integration is done by exchanging partial plans
- no representation of the whole system plan

Concurrent Reactive Plans

Execution of plan libraries

- a plan library
- tools to adapt plans continuously
  - $\Rightarrow$ transformation planning
- state projection

Demonstrated the validity of a “plan management” scheme

Beetz – 2002
Concurrent Reactive Plans

GPGP/TAEMS
Plan-based control of multi-robot systems

- plan model based *only* on tasks and task graphs
  - ⇒ rich set of task relations
  - ⇒ designed for scheduling
- designed for multi-robot systems
- whiteboard-based approach of plan building

Lesser, Decker et al. – 2004
Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework

Our approach

General principle

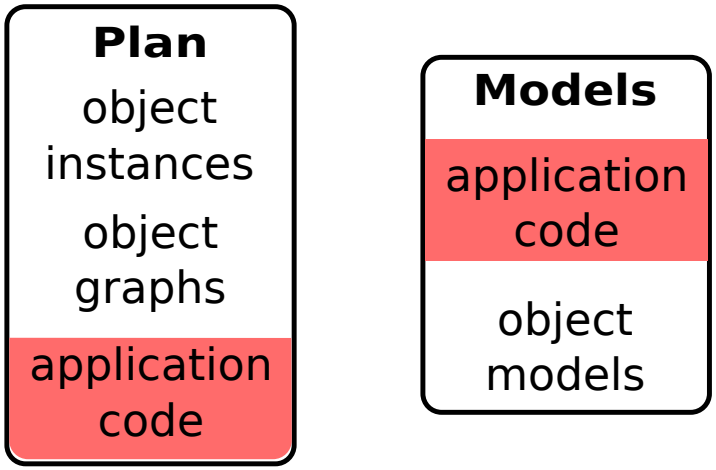
One central place where the whole system's plan is represented managed

Fit to be the integration place for

- planners
- reactive decision-making
- plan analysis tools: situation assessment, diagnostics, ...

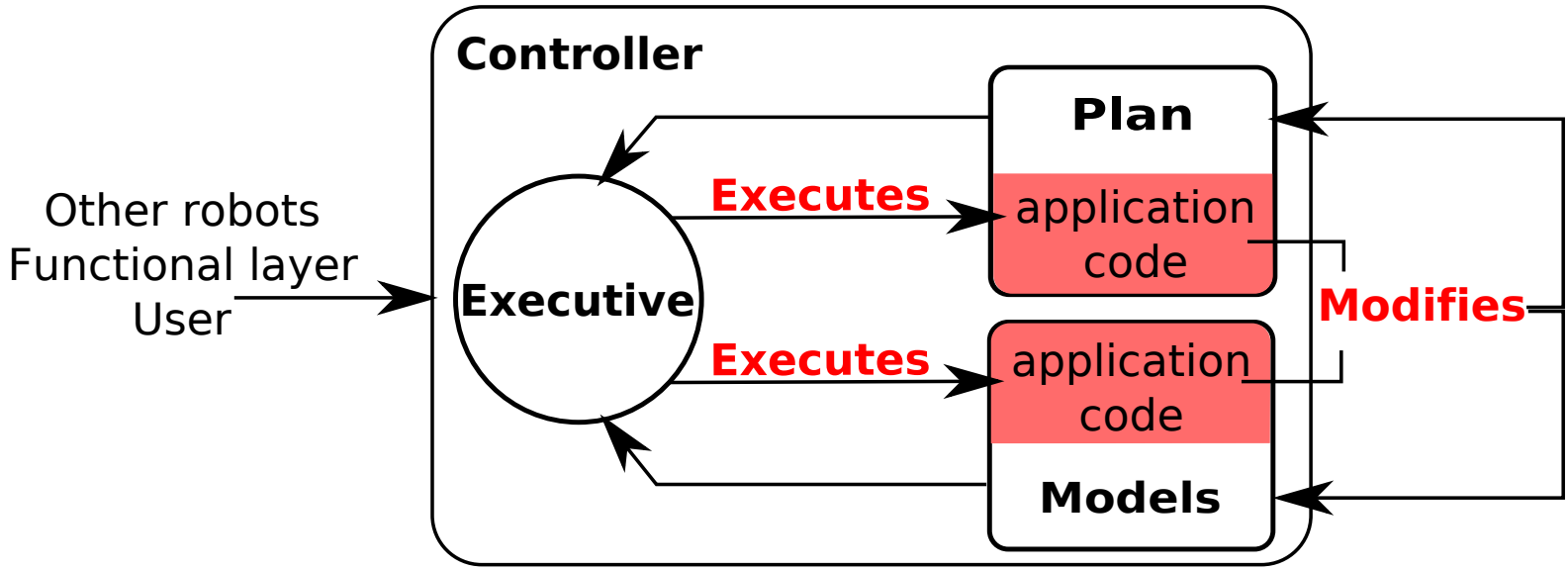
Our approach

1. A multi-robot plan model
   - as planner-agnostic as possible
     - ⇒ representation based on actions, not state
   - aimed at supervision, not planning
     - ⇒ represents from high-level actions to low-level ones
     - ⇒ rich situation representation
   - designed with multi-robot in mind



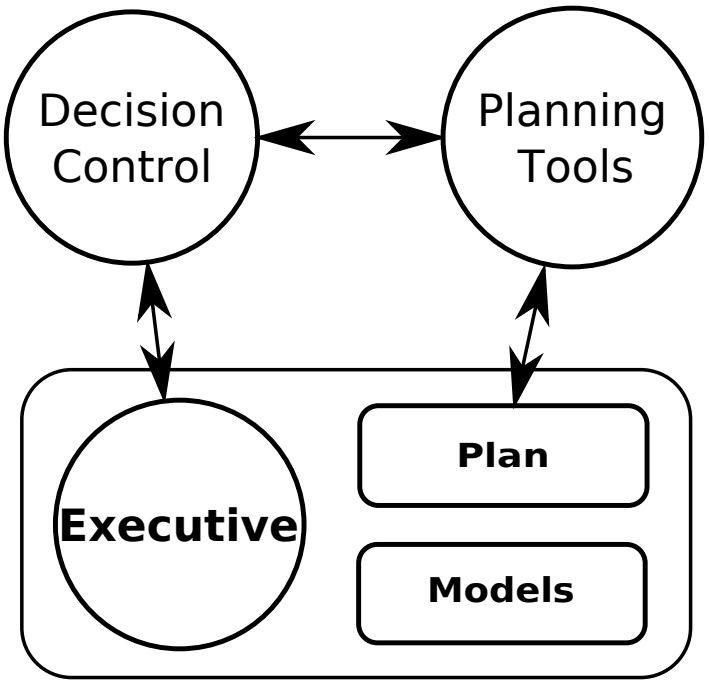
Our approach

1. A multi-robot plan model
2. A plan management scheme
   - execution of the said plan
     - ⇒ in particular, representation of errors and of their recovery process
   - adaptation as it is being executed
     - ⇒ dynamic plans



Our approach

1. A multi-robot plan model
2. A plan management scheme
3. Decision-making for plan execution
   - online plan fault resolution
   - planning-execution conflict resolution



Outline of the presentation

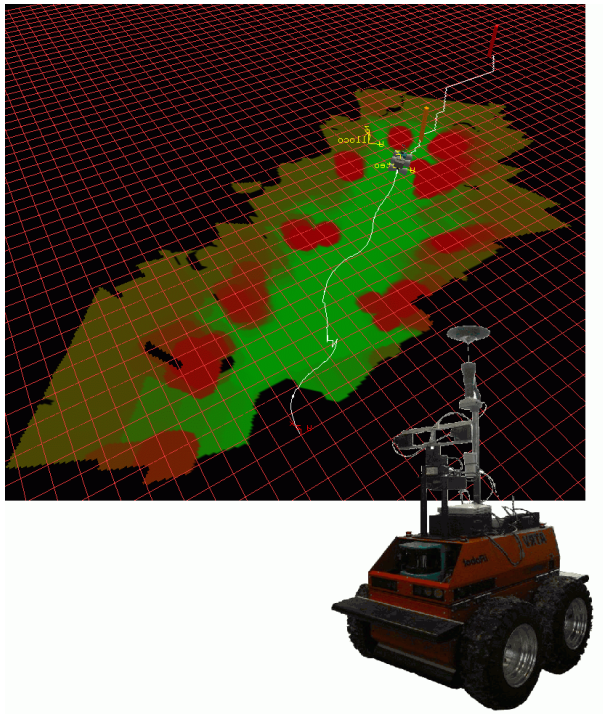
Illustration and validation of our approach

- through a complete robot
  - $\Rightarrow$ must manage a complex set of functions
  - $\Rightarrow$ “big enough” plan sizes to test the execution scalability
- through the management of a bi-robot plan
  - $\Rightarrow$ model fit to control multiple robots
  - $\Rightarrow$ building joint plans
  - $\Rightarrow$ joint plan execution

Mono-robot scenario: the Dala rover

Mission: navigation in unknown environment

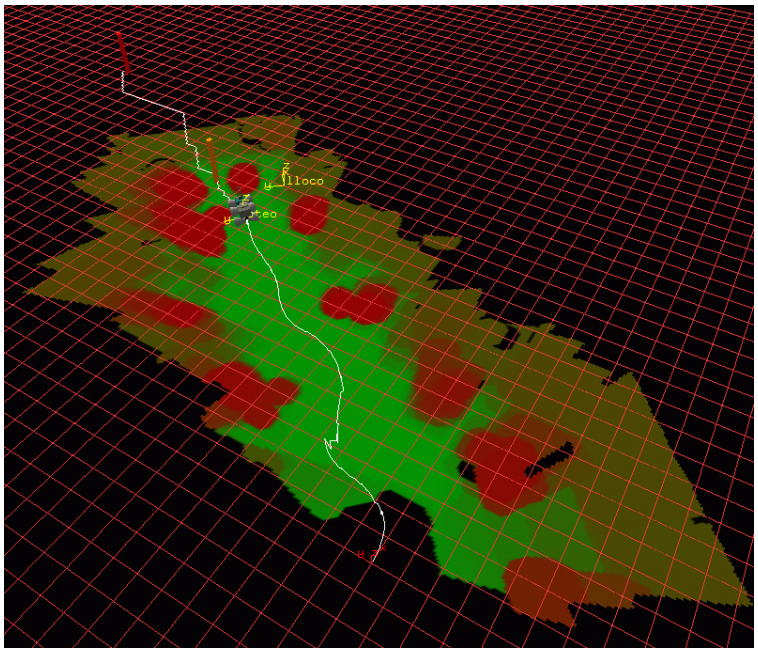
- **Bitmap** builds a global traversability map
- **Nav** long-range path planning in traversability map
- **Dtm** builds a local elevation map
- **P3d** short-range path planning in elevation map
- **Pom** position fusion



Bi-robot scenario: rover/UAV cooperation

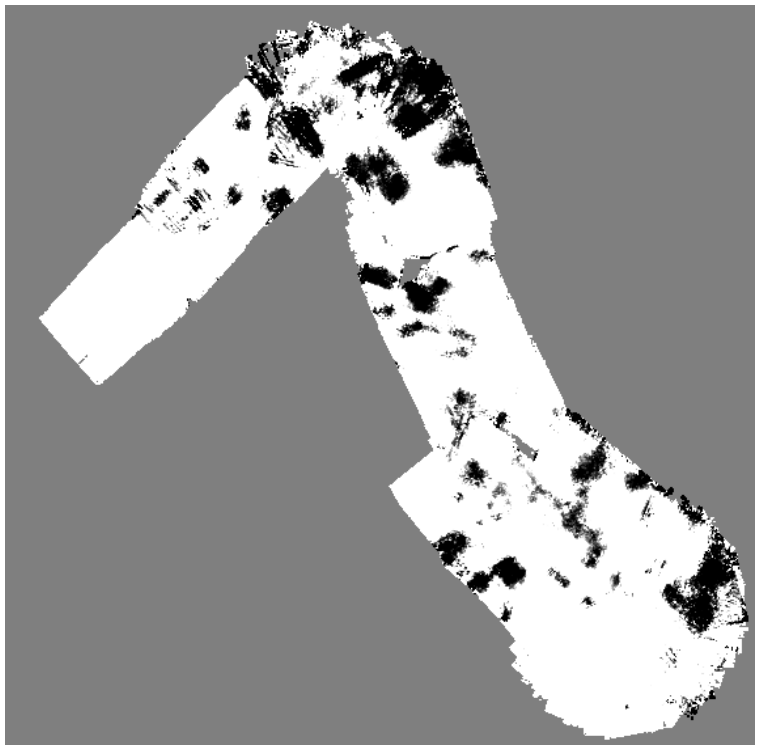
Rover

$\Rightarrow$ local perception



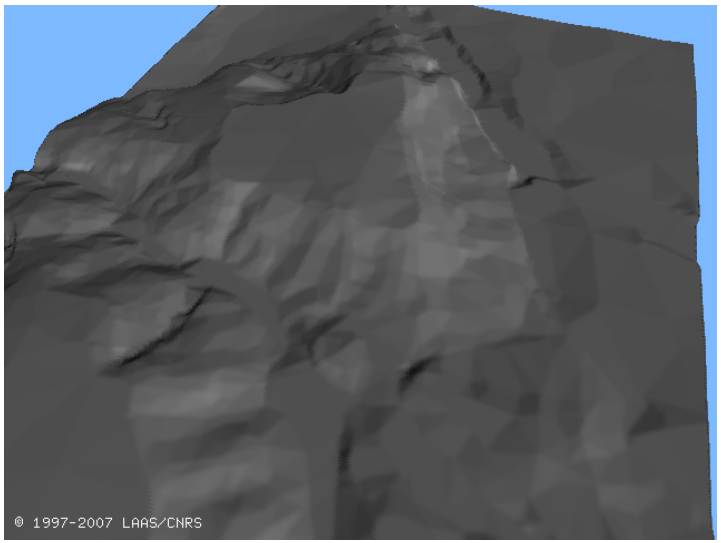
UAV

$\Rightarrow$ long-range perception



Simulated bi-robot setup

Terrain



+ rich simulation: the robot software is used almost as-is
+ fault injection to test error handling

Experimental mono-robot setup

The Dala rover

Fault injection devices

"Algeco", here at LAAS

Caylus, military training ground

RF emergency brakes

Esperce airfield

Thanks to Thierry, Matthieu and Arnaud

Experimental bi-robot setup

The Ressac UAV

Thanks to Roger, Vincent and Alain at ONERA

2. Plan Model

Overview

What do we need ?

- a representation of activities
  $\Rightarrow$ tasks
  The robot has already reached a given point, and is now moving towards another. It will update its map in the future.

Overview

What do we need ?

- a representation of activities
  $\Rightarrow$ tasks
- a representation of the activity interactions
  $\Rightarrow$ task graphs
  - how an activity influences another
    What impact the terrain mapping has on the movement ?

Overview

What do we need ?

- a representation of activities
  $\Rightarrow$ tasks
- a representation of the activity interactions
  $\Rightarrow$ task graphs
  - how an activity influences another
  - constraints on the activities
    Is it allowed for the localization to stop ?

Overview

What do we need ?

- a representation of activities
  - $\Rightarrow$ tasks
- a representation of the activity interactions
  - $\Rightarrow$ task graphs
    - how an activity influences another
    - constraints on the activities
- a representation of the activities evolution
  - $\Rightarrow$ events

The localization has already started. The movement has just blocked.

Overview

What do we need ?

- a representation of activities
  - $\Rightarrow$ tasks
- a representation of the activity interactions
  - $\Rightarrow$ task graphs
    - how an activity influences another
    - constraints on the activities
- a representation of the activities evolution
  - $\Rightarrow$ events
- a representation of the whole system evolution
  - $\Rightarrow$ event graphs

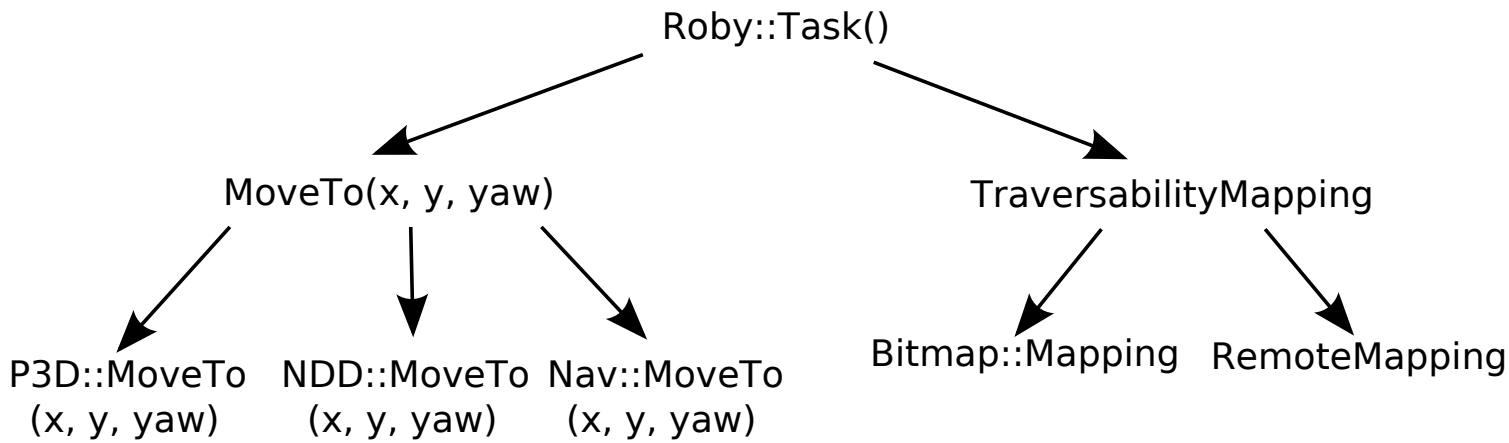
When the mapping is started and the path is computed, the robot can start moving

Tasks: representation of activities

- parametrized activity model
  - `MoveTo(x, y)`
  - `Pom::Localization()`
  - `MapZone(x_start, y_start, x_final, y_final)`
  - `TrackPath(path)`

Tasks: representation of activities

- parametrized activity model
- hierarchical organization of task models
  - $\Rightarrow$ allows to represent equivalence between activities



Tasks: representation of activities

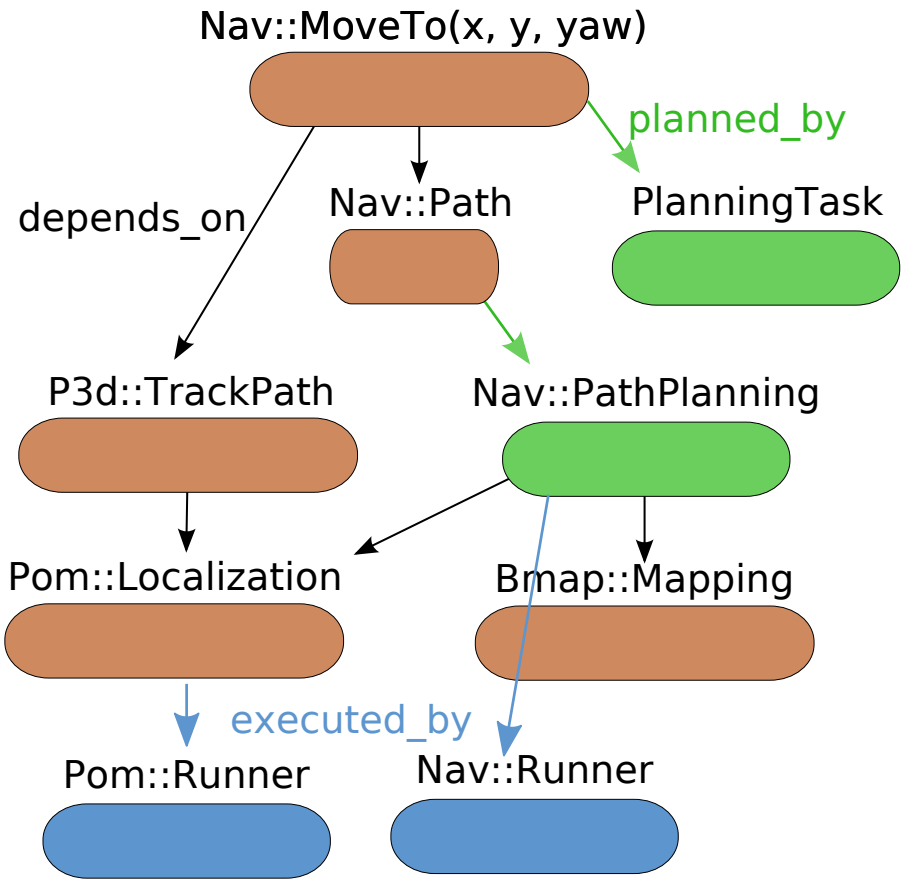
- parametrized activity model
- hierarchical organization of task models
- notion of non-executable tasks
  - $\Rightarrow$ partially instantiated activities
  - $\Rightarrow$ activities from abstract models

Task relations: activity interactions

Different kinds of relations

- hard dependency
- planning
- execution support

⇒ task graphs !

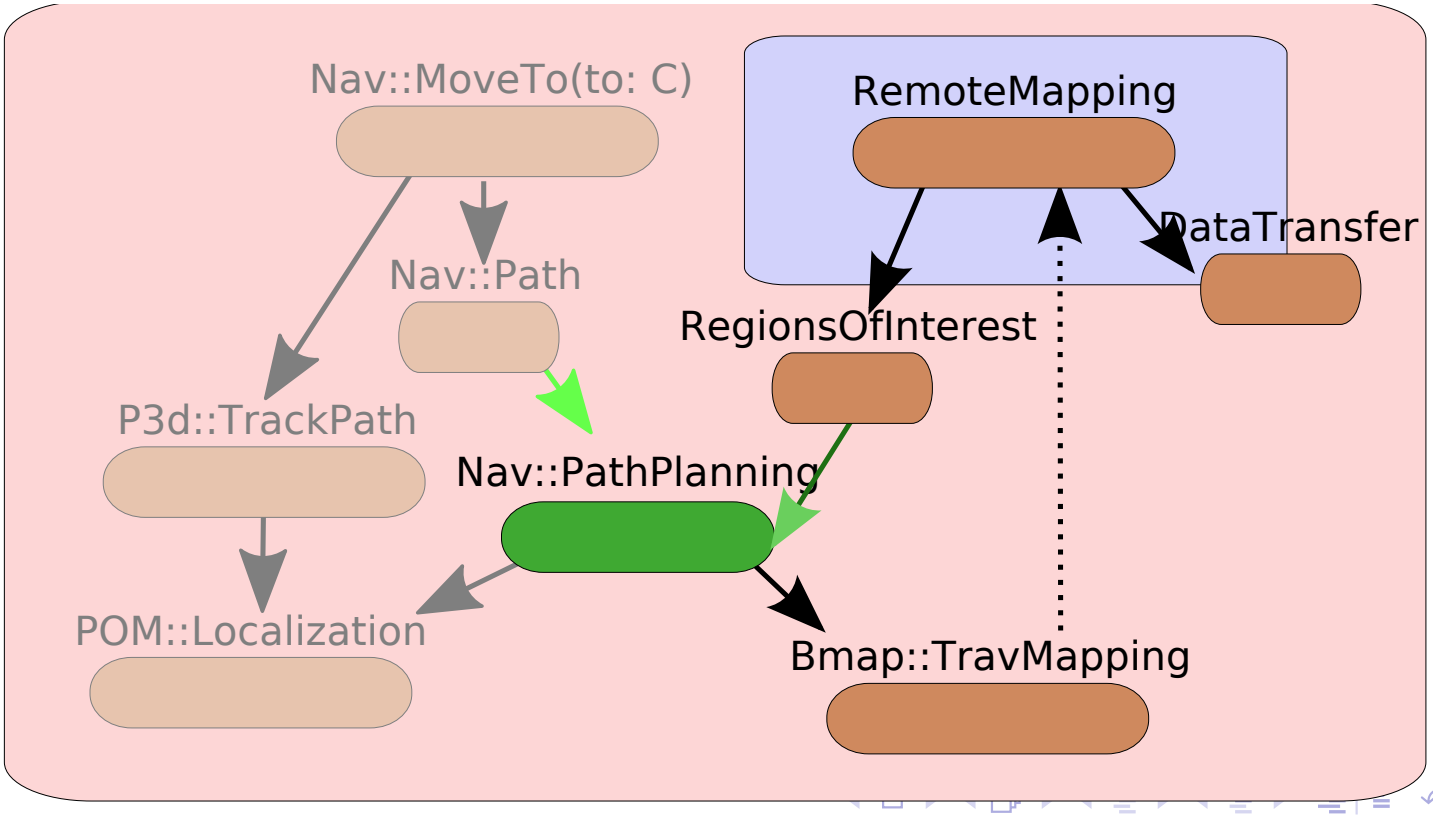


Multi-robot plans: ownership and roles

Notion of ownership and roles

ownership who is doing what ?

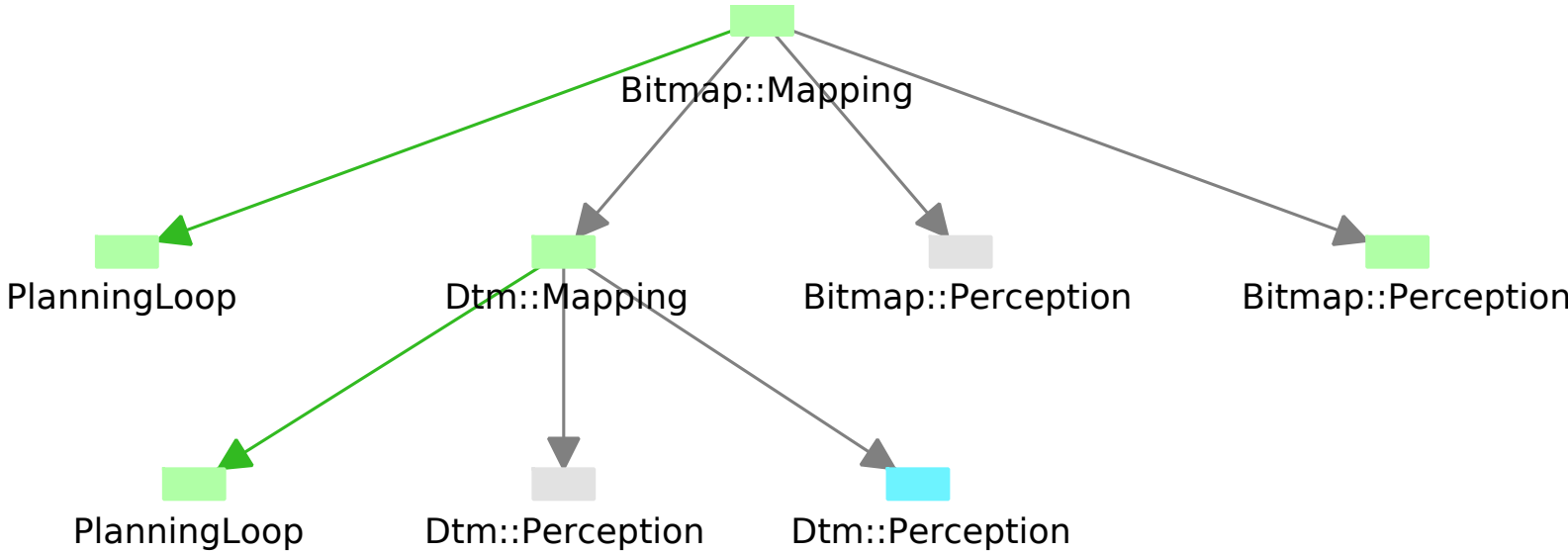
role who will be doing what in team interest ?



Execution flow: what for ?
Example: perception on the Dala rover

The needs

- trigger the updates at the appropriate times
- need to sequence the perception activities
  - when does a perception should start/stop ?
  - when are the maps updated ?
  - ...

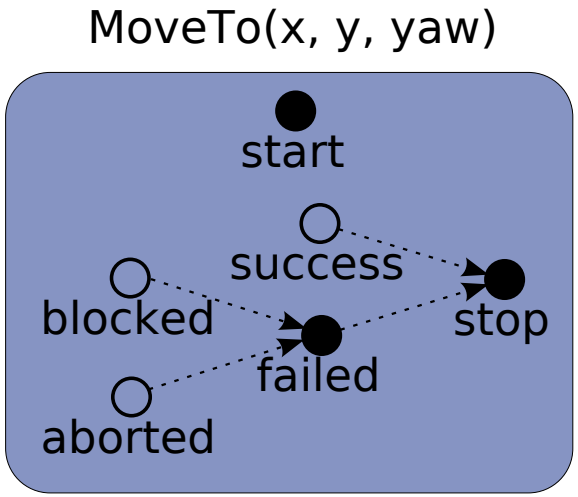


Events
A representation of milestones during execution

- represents a noticeable situation
  - $\Rightarrow$ the robot moved more than x meters
  - $\Rightarrow$ timeout of 5 seconds
  - $\Rightarrow$ ...

Events
A representation of milestones during execution

- represents a noticeable situation
  - ⇒ the robot moved more than *x* meters
  - ⇒ timeout of 5 seconds
  - ⇒ tasks are composed of events



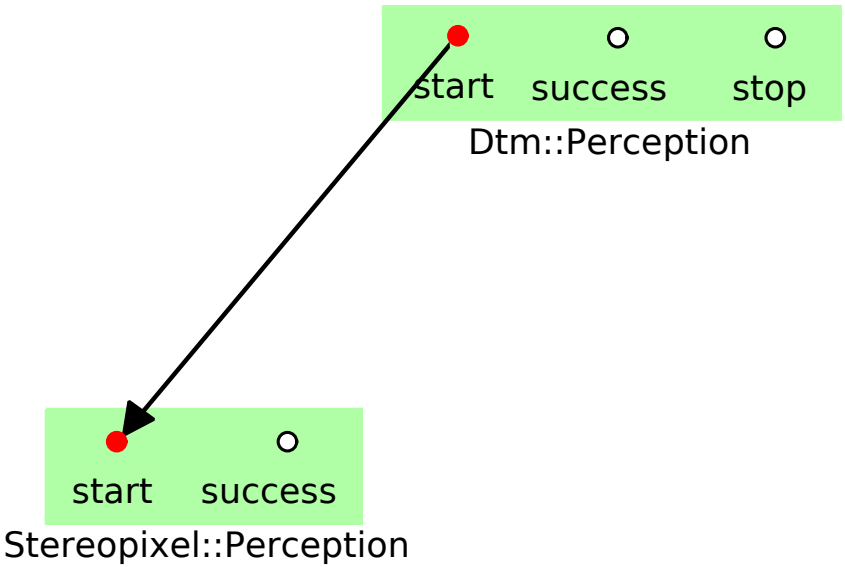
Events
A representation of milestones during execution

- represents a noticeable situation
  - ⇒ the robot moved more than x meters
  - ⇒ timeout of 5 seconds
  - ⇒ tasks are composed of events
- interface to control the system
  - ⇒ start an activity/stop it
  - ⇒ update the map
  - ⇒ ...

Reacting to events: signalling

Describes the reaction to events

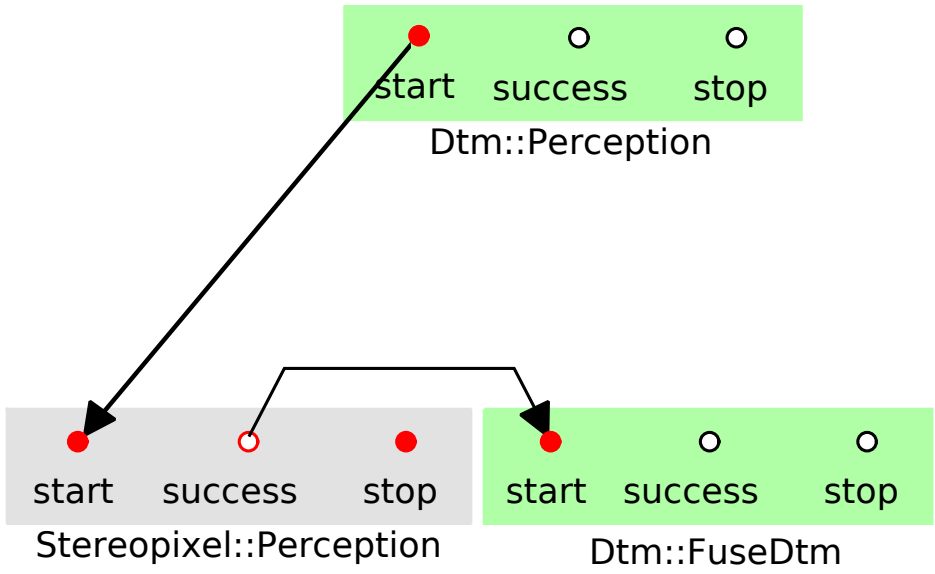
When there is a signal $e_1 \rightarrow e_2$ the command of e_2 is called when e_1 is emitted



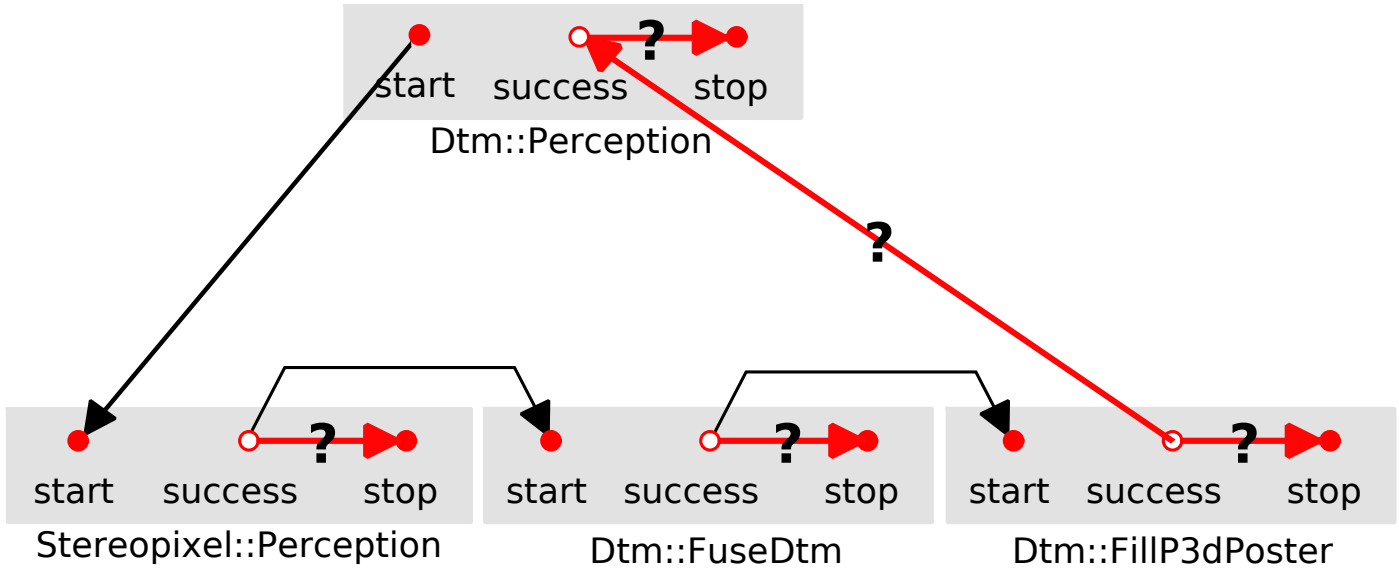
Reacting to events: signalling

Describes the reaction to events

When there is a signal $e_1 \rightarrow e_2$ the command of e_2 is called when e_1 is emitted



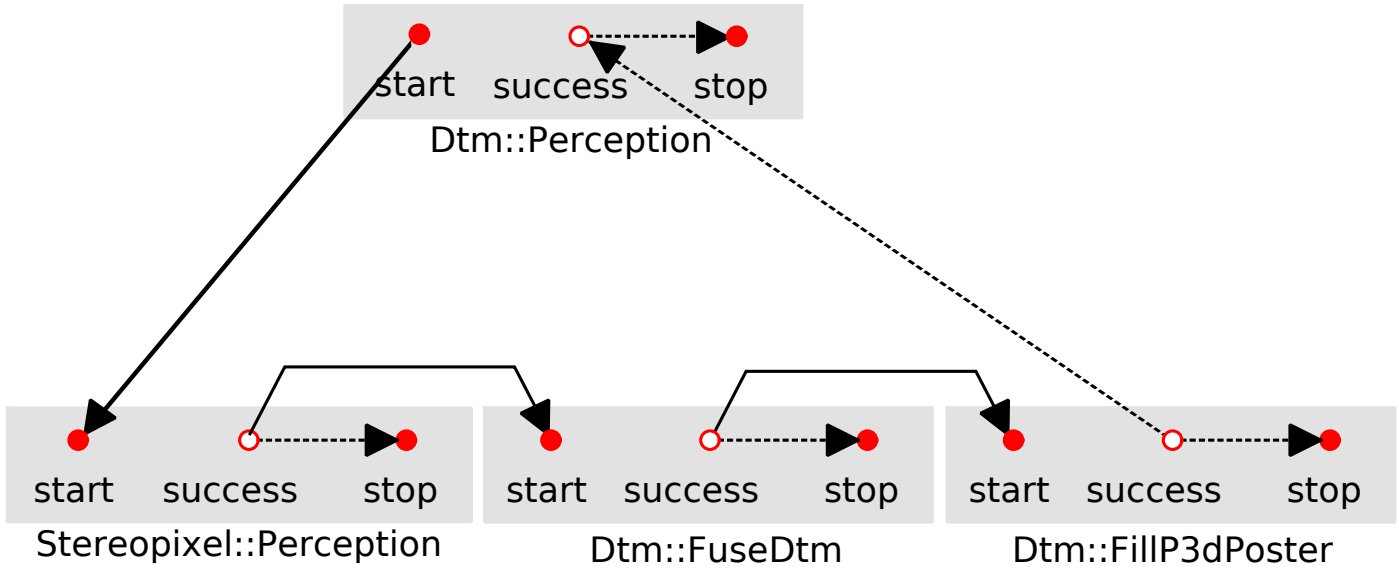
Signal is not enough !



Event generalization: forwarding

Generalization semantic between events

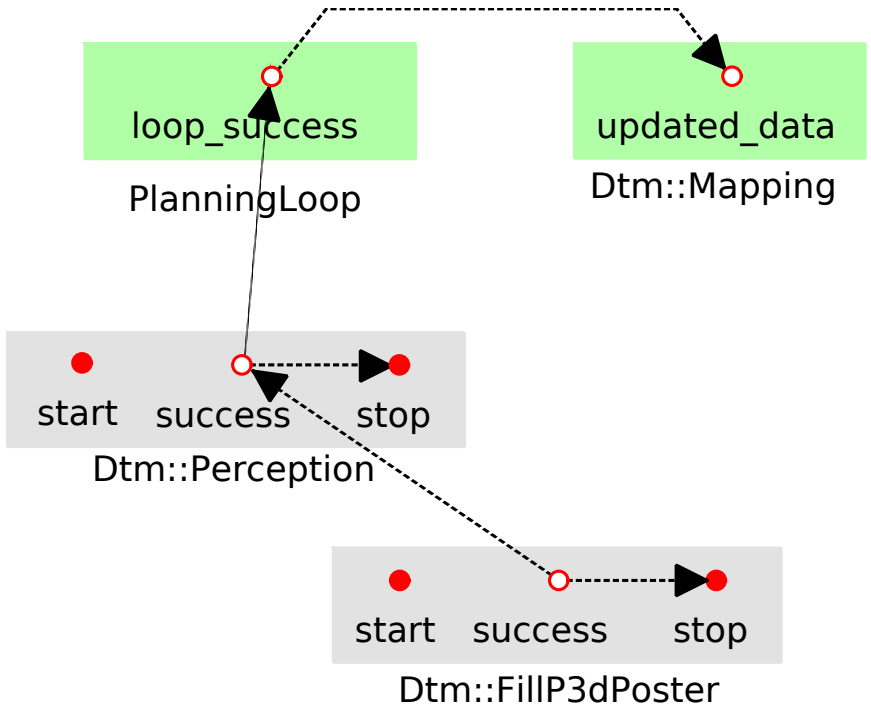
When e_1 is forwarded on e_2, e_2 is emitted when e_1 is. **Therefore** the situations represented by e_2 are a superset of those represented by e_1.



Event generalization: forwarding

Generalization semantic between events

When e_1 is forwarded on e_2, e_2 is emitted when e_1 is. **Therefore** the situations represented by e_2 are a superset of those represented by e_1.



Back to Dala perception loops

- implemented using two loop structures
  - ⇒ generic structures built with the standard model
- triggers are state events
  - ⇒ in this case, represent a delta in position, heading or time

▸ The loop construct

Summary

- with separated activity and temporal structures
- allows to represent progressive tasks and their interaction
- able to represent joint plans

3. Plan Execution

Overview

- event propagation
  - $\Rightarrow$ synchronous model
  - $\Rightarrow$ global propagation algorithm

Overview

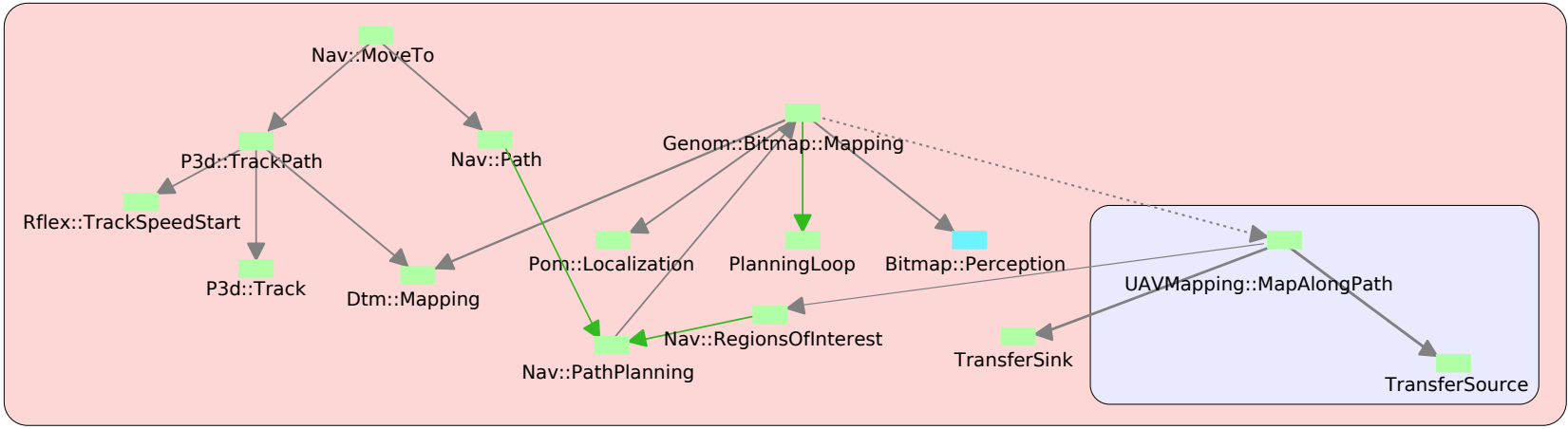
- event propagation
  - ⇒ synchronous model
  - ⇒ global propagation algorithm
- multi-robot execution
- rich error definition and handling
- automatic cleanup of the plan

Composition of plans in multi-robot context

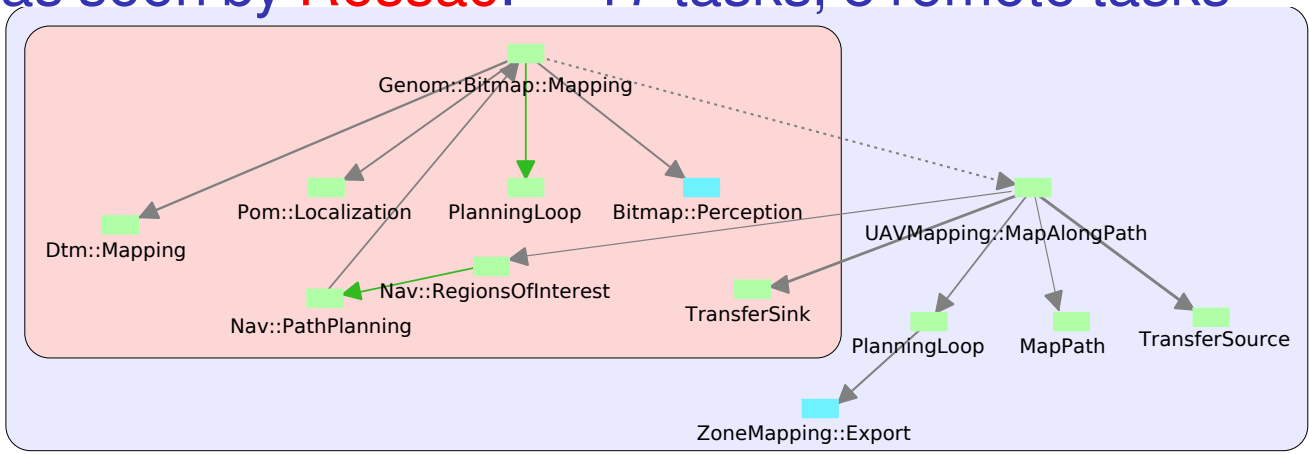
- cannot have all tasks of all robots in all plans
- selection of relevant tasks is based on:
  - direct relation to one's own plan
    - $\Rightarrow$ there is a relation between a local task and the remote task
  - an explicit subscription mechanism
    - $\Rightarrow$ a plan manager can explicitly ask another to send updates about its tasks

Example: partial views of Dala and Ressac
Live data from Caylus 25/10

Joint plan as seen by Dala: ∼ 65 tasks, 2 remote tasks



Joint plan as seen by Ressac: ∼ 17 tasks, 8 remote tasks

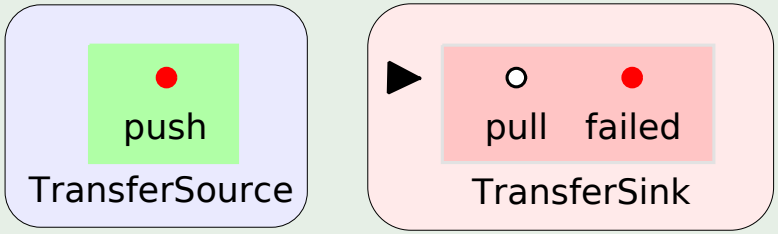


Synchronization problems in multi-robot

The protocol has synchronization guaranties

- either it guarantees synchronization
  - ⇒ example: plan building tools
- or it reports plan-related errors
  - ⇒ lack of synchronization in the plan

Example



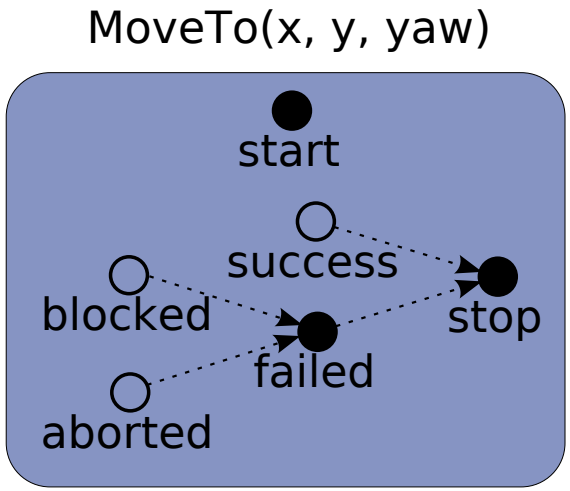
1. Ressac sends a notification to Dala
2. communication delays
   - ⇒ `TransferSink` fails in the meantime
3. Dala receives the notification but the target does not exist anymore
   - ⇒ `SynchronizationError` on common parent

Error handling

- a generic description of errors as
  - a type
  - a fault point
- means to constrain the system
  - $\Rightarrow$ to define what's allowed and what is not
- means to react to errors
- and a default mechanism for unhandled errors

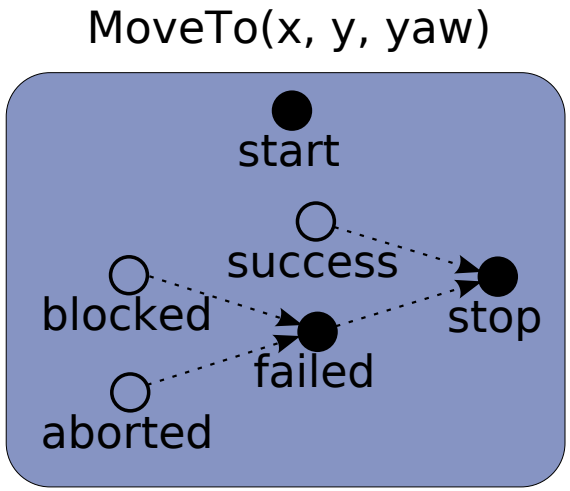
Managing fault modes

$\Rightarrow$ forwarding is used to represent fault modes



Managing fault modes

$\Rightarrow$ forwarding is used to represent fault modes



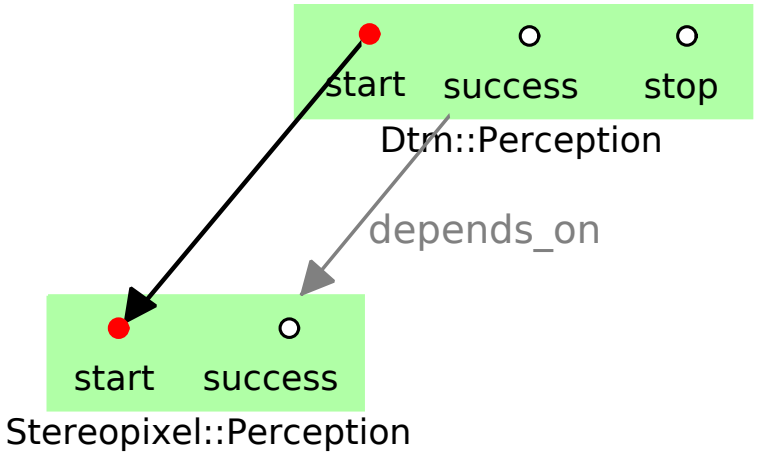
Forwarding and failure points

- a handler which applies to a given erroneous situation . . .
- . . . also applies to the subsets of this situation

Task relations as constraint definition

The link between events and task relations

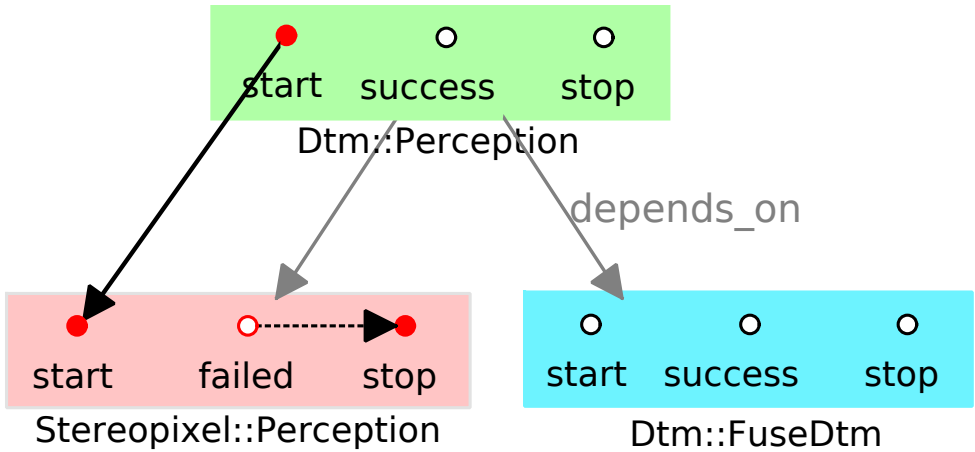
Task relations describe which situations are desired and which are forbidden



Task relations as constraint definition

The link between events and task relations

Task relations describe which situations are desired and which are forbidden



Type `DependencyFailedError`

failure point `failed` event of `Stereopixel::Perception`

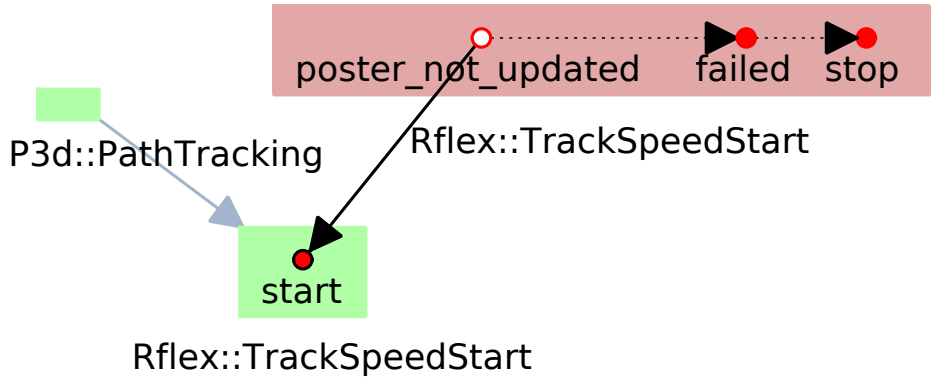
Error reparation

Three error handling methods

1. repair errors during event propagation
2. repair using subplans
3. exception propagation

Repairing errors during event propagation

An event handler of a specific erroneous event repairs the plan



Asynchronous repairs

- mark a task as repairing a failure point
  $\Rightarrow$ plan repair
- during the lifetime of the repair task and while a timeout is not reached
  $\Rightarrow$ the error is inhibited

The repair_with relation

This relation associates a task and an event which may be a failure point

$\Rightarrow$ activated when the failure point is the event

Example: P3D's `blocked` event

Cascade of error handlers

When the motion control reports "blocked"

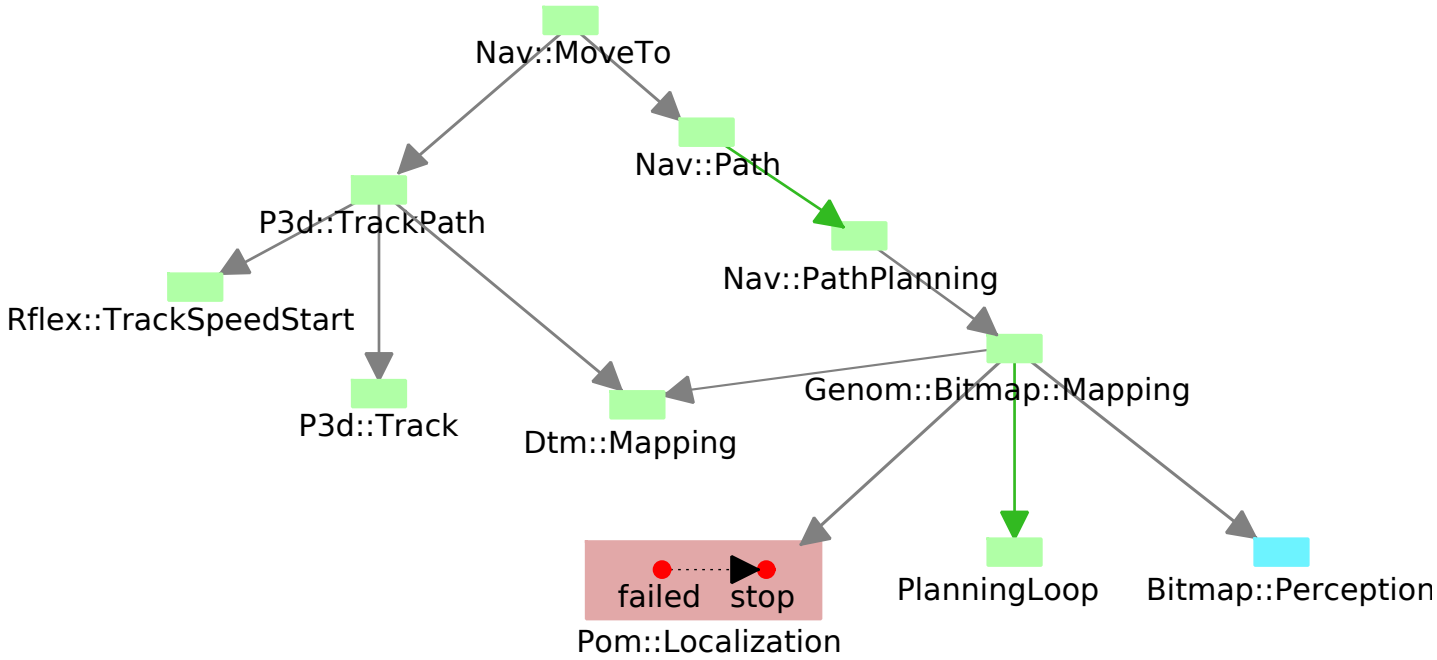
1. update the elevation map locally
2. if it still fails, reinitialize the elevation map
3. if it still fails, the robot is really blocked
   $\Rightarrow$ real error

+. if the robot moves after a repair, reinitialize to 1

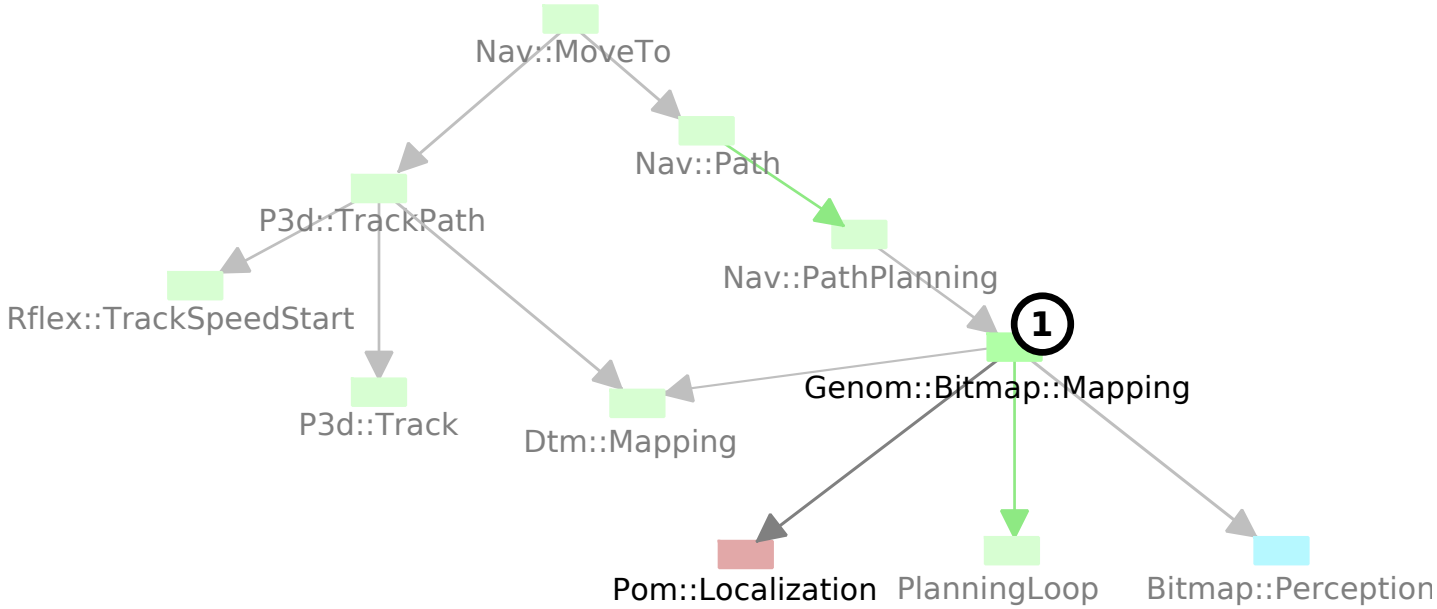
▸ Video ▸ `brakes_on` on Rflex

Exception handling

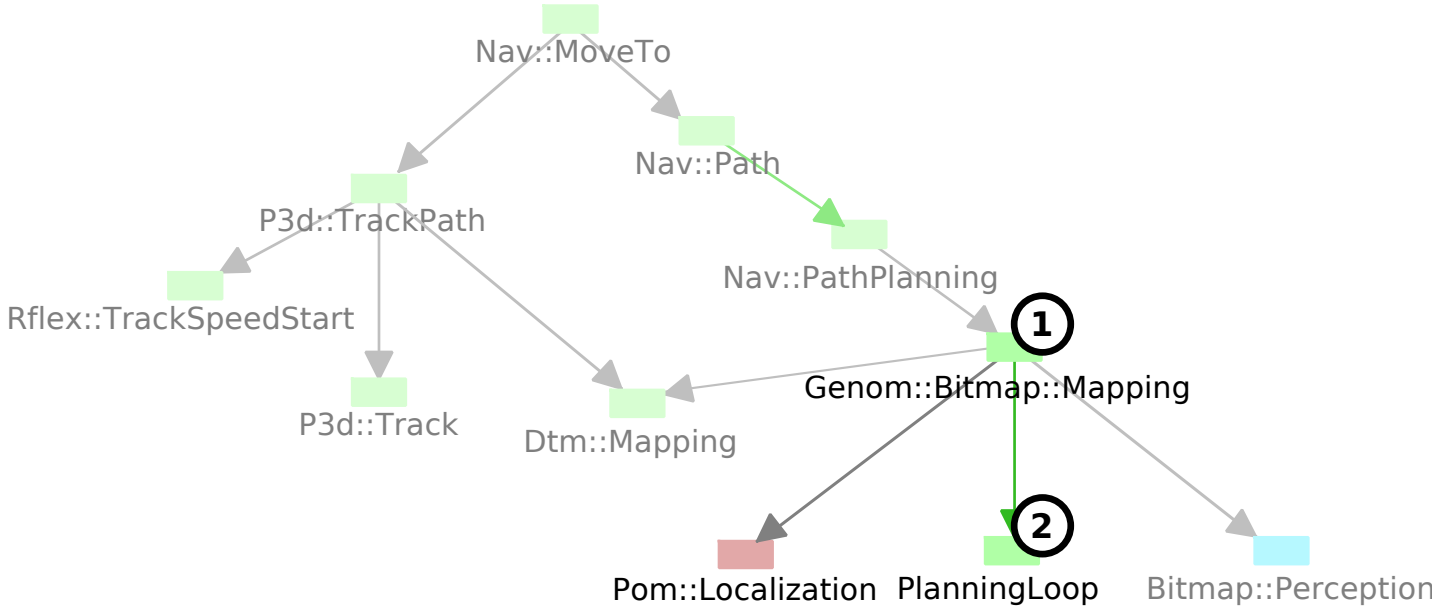
- a failure
  - is not handled by a repair_with relation
  - has no associated plan repair
- a repair task has failed



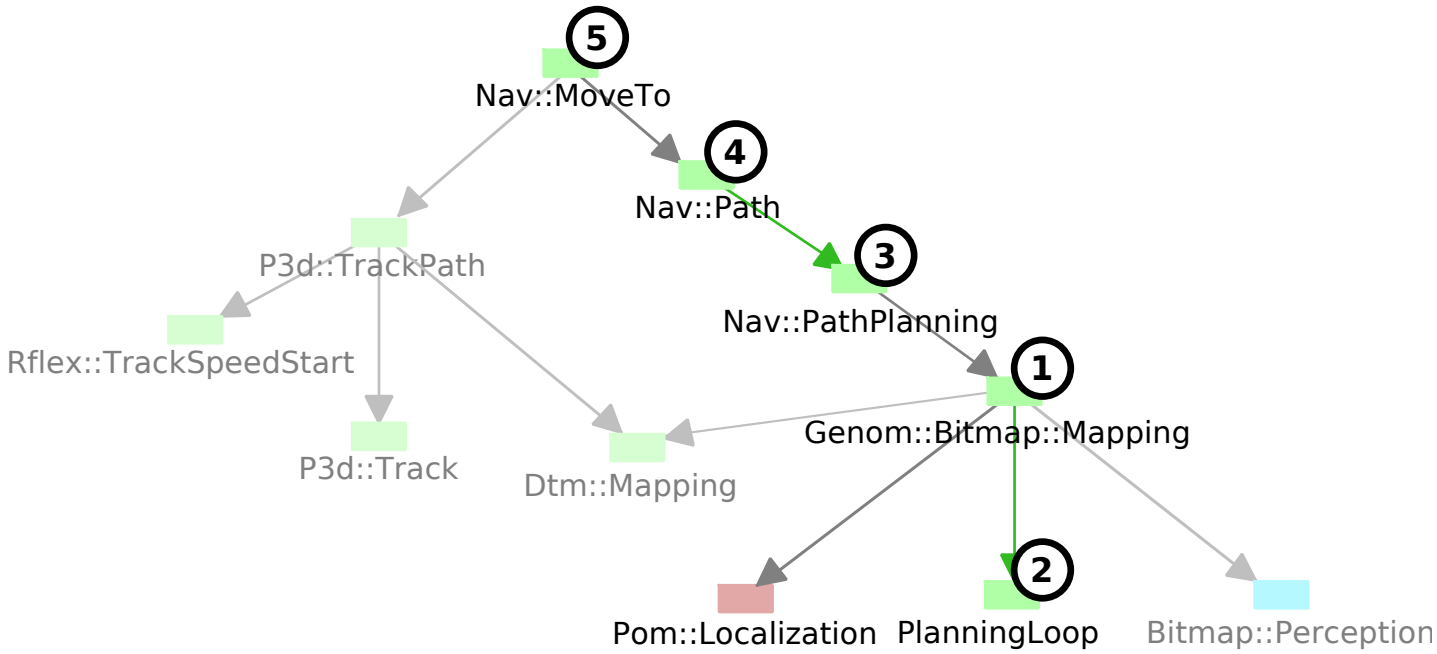
Exception handling call order



Exception handling call order



Exception handling call order



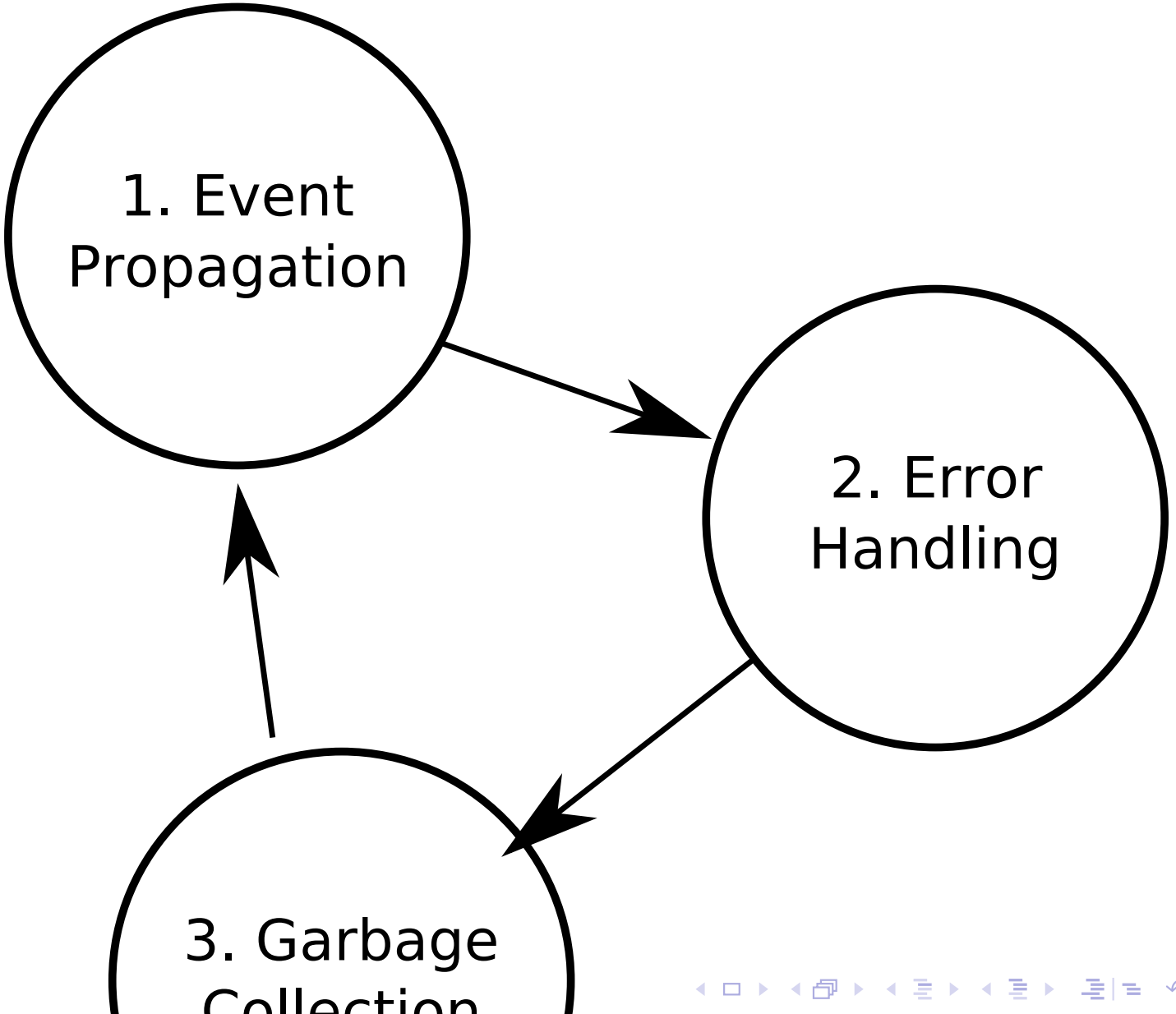
Garbage collection

- must remove tasks which are harmful
- must remove tasks that are not useful anymore
  - not used by any missions
  - not meaningful in the robot-robot interaction

Our plans are made of task graphs

$\Rightarrow$ global algorithm

Summary: the execution cycle



4. Plan Adaptation

Transactions: motivation

- our plan manager relies on task and event structure to manage the system
  - execution flow
  - determination of errors
  - garbage collection
  - ...

Transactions: motivation

- our plan manager relies on task and event structure to manage the system

$\Rightarrow$ we need a tool to change the old structure into the new one in a single step

Transactions: motivation

- our plan manager relies on task and event structure to manage the system

$\Rightarrow$ we need a tool to change the old structure into the new one in a single step

- plan generation is not a fast process
  - planning is a complex task
  - communication latency in multi-robot

Transactions: motivation

- our plan manager relies on task and event structure to manage the system

$\Rightarrow$ we need a tool to change the old structure into the new one in a single step

- plan generation is not a fast process

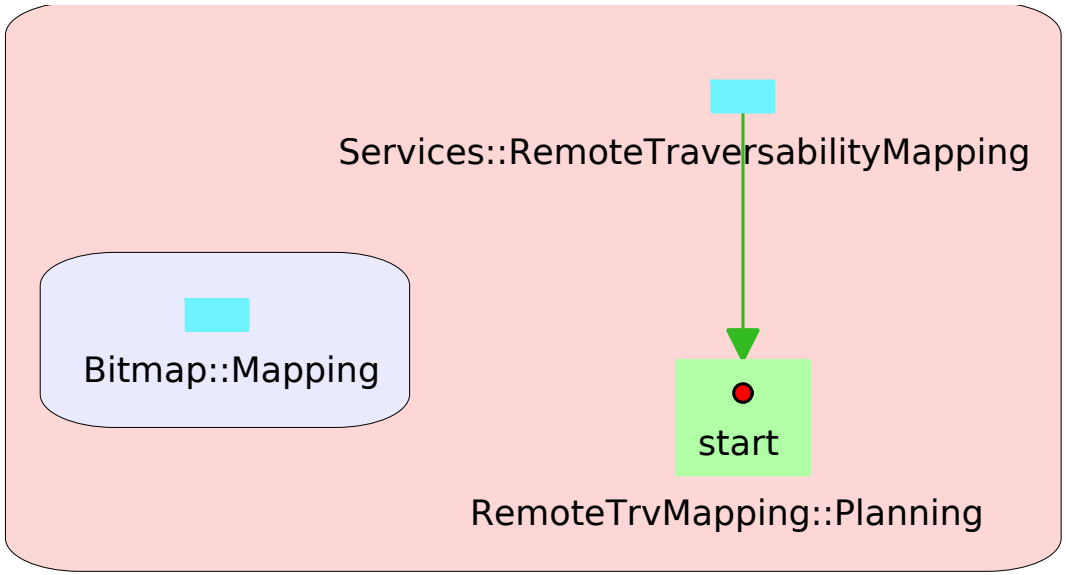
$\Rightarrow$ we need a tool to allow simultaneous plan generation and execution

Transaction

- originally a concept from databases
- represents the transformation
  - from the plan as it is being executed
  - into the new plan
- negotiation: allows to sandbox every kind of plan change
  - structure
  - ownership
  - roles
  - subscription
  - ...

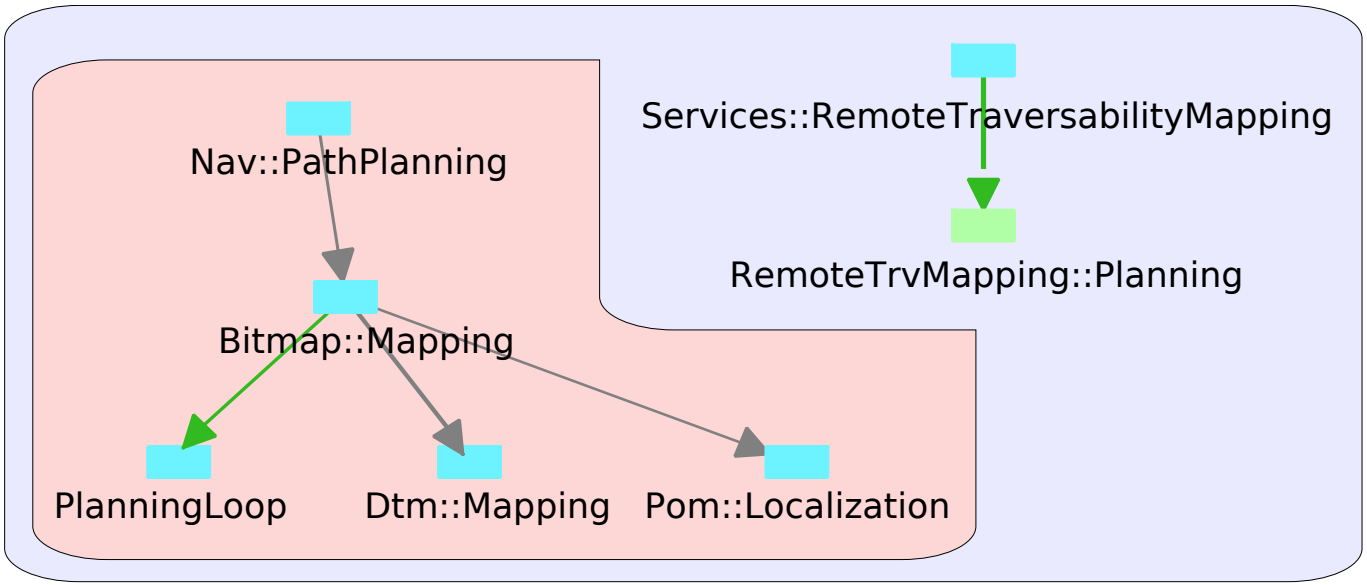
Example: Dala/Ressac negotiation
Live data from Caylus 25/10

Ressac's plan



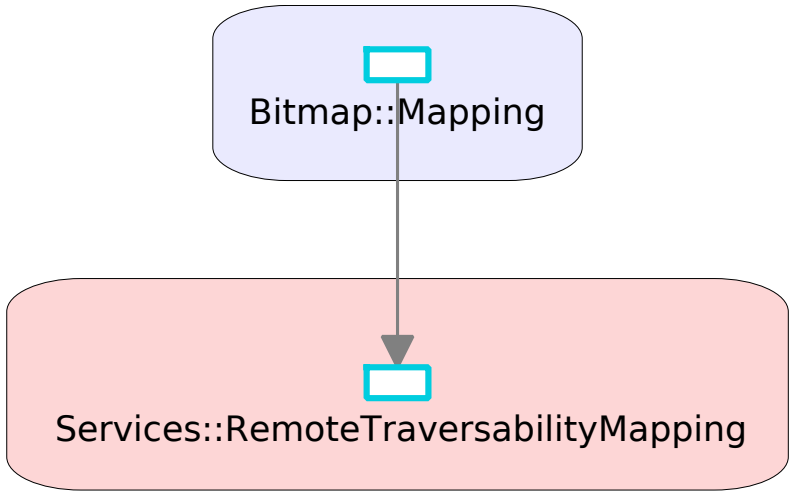
Example: Dala/Ressac negotiation
Live data from Caylus 25/10

Ressac's plan



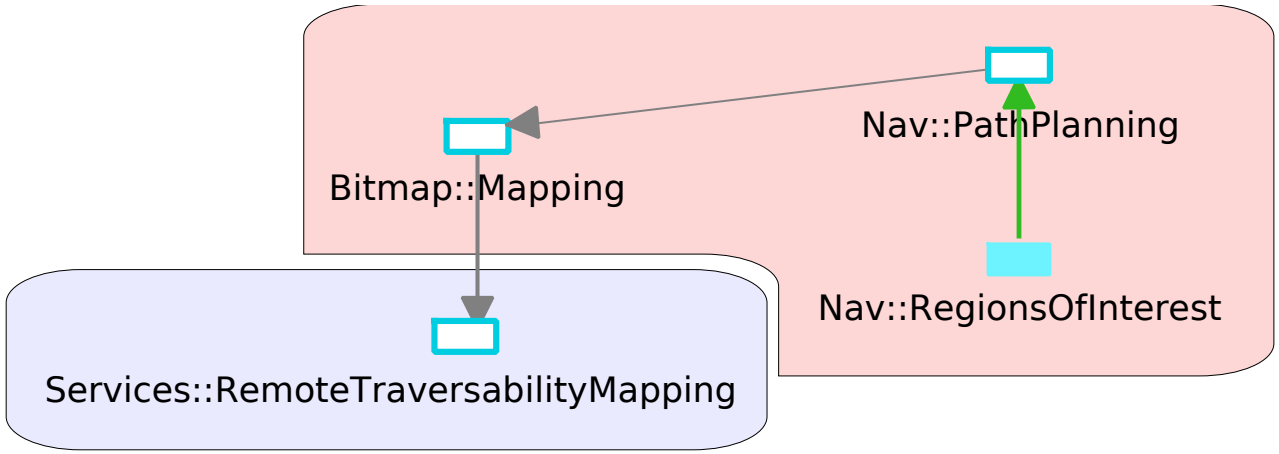
Abstract proposal for Dala

Transaction built by Ressac



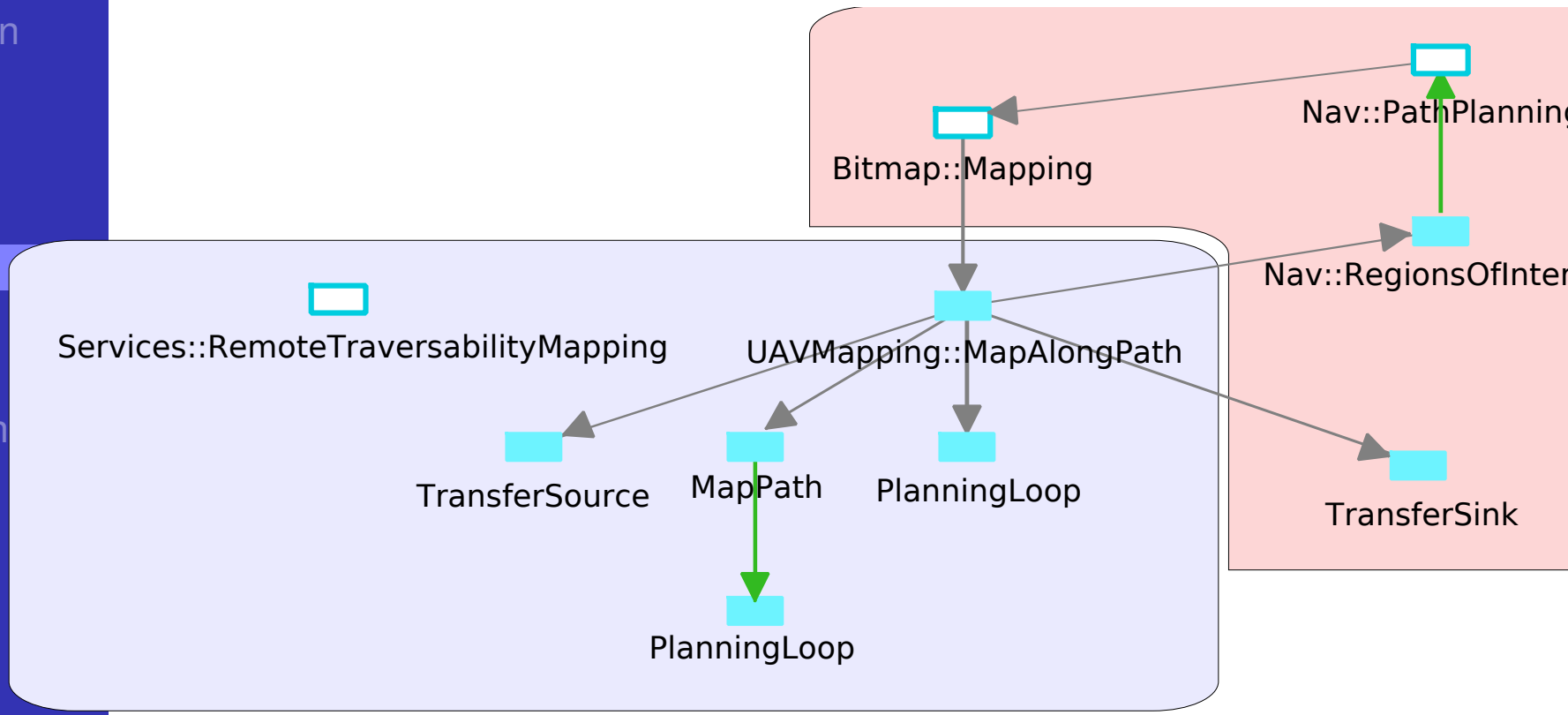
Dala's accepts and adds the `RegionsOfInterest` task

Transaction as modified by Dala



Example: Dala/Ressac negotiation
Live data from Caylus 25/10

Final state of the transaction



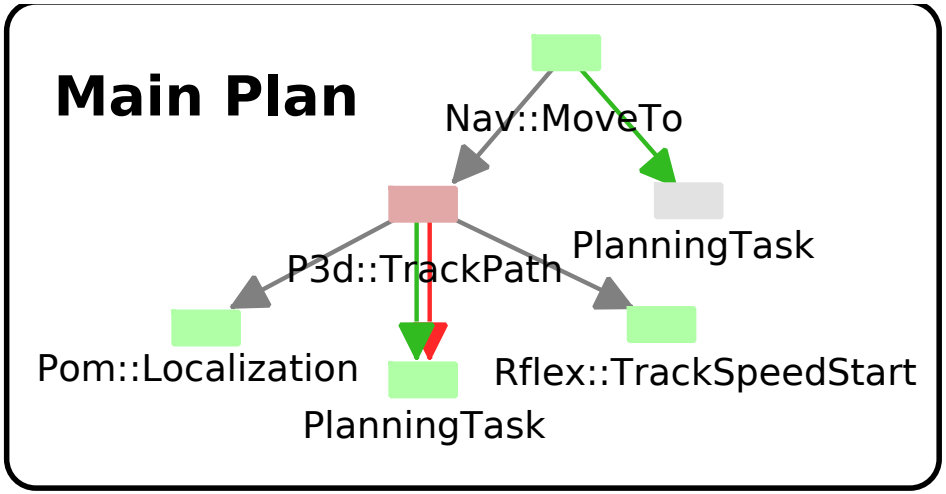
Planning/execution conflict

The plan is executed while the transactions are being built

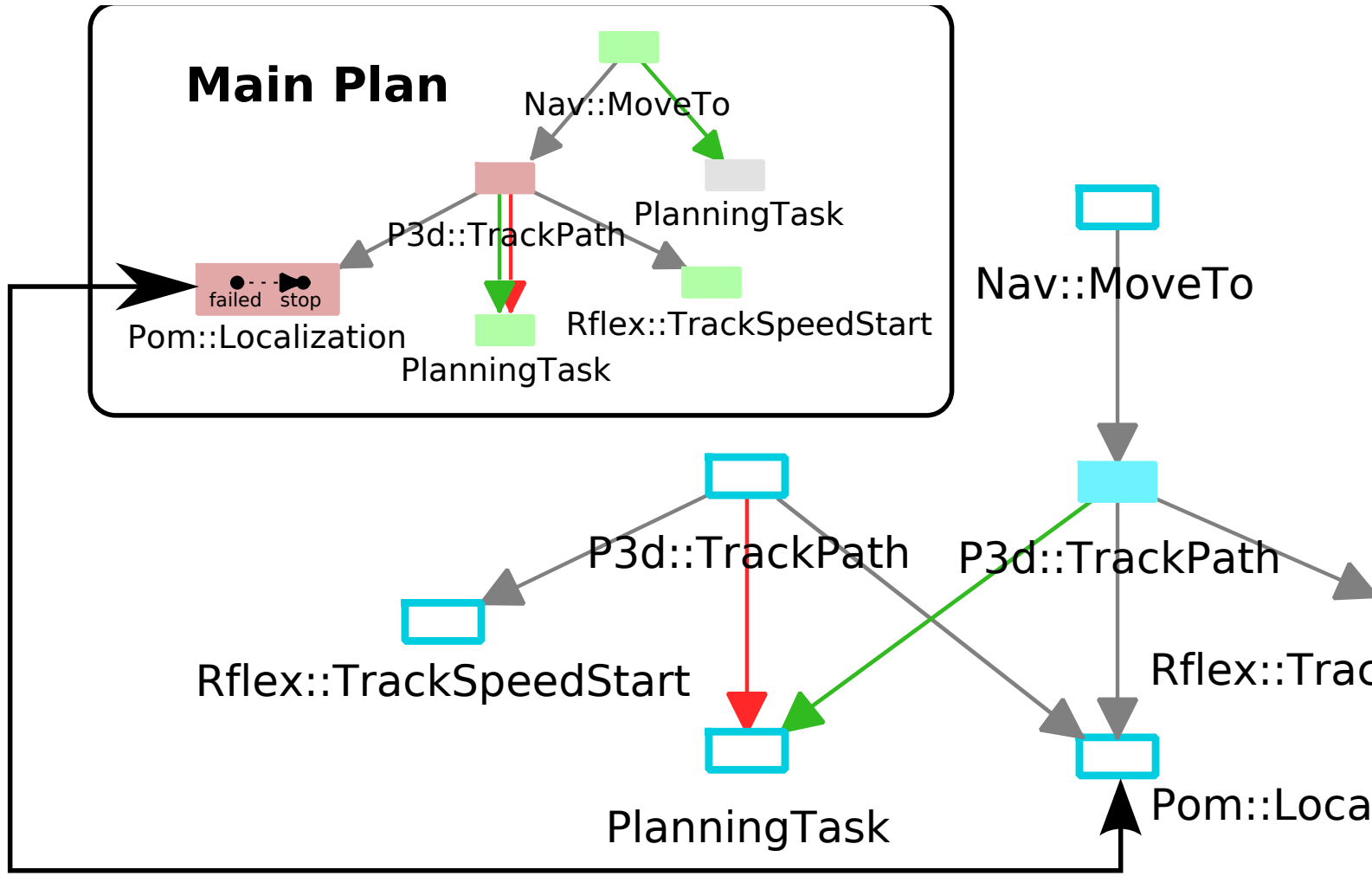
Every time the executed plan changes . . .

. . . compare the transaction with the new executed plan

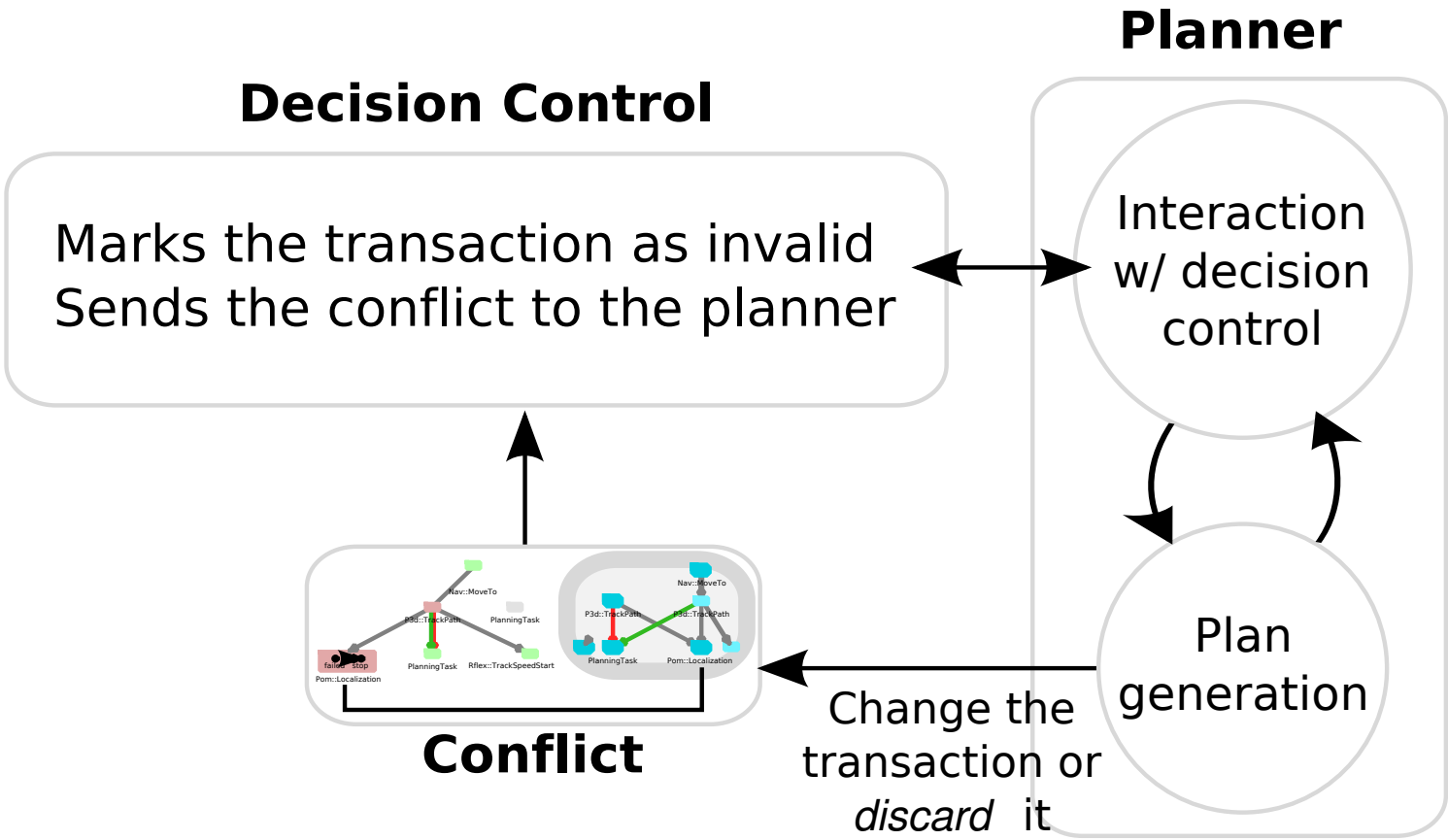
Example: replanning `P3d::TrackPath`



Planning/execution conflict



Edition cycle: handling invalid transactions



Planning/execution conflicts

- simplest solution: discard the transaction
- otherwise, adapt the transaction
- the decision control can also forbid a plan change brought by execution

Summary

Because of . . .

- the way our system works
- need for simultaneous planning & execution
- need for a plan-centric negociation tool

we defined transactions

$\Rightarrow$ sandbox to prepare plan transformations

5. Results

Implementation: the Roby software

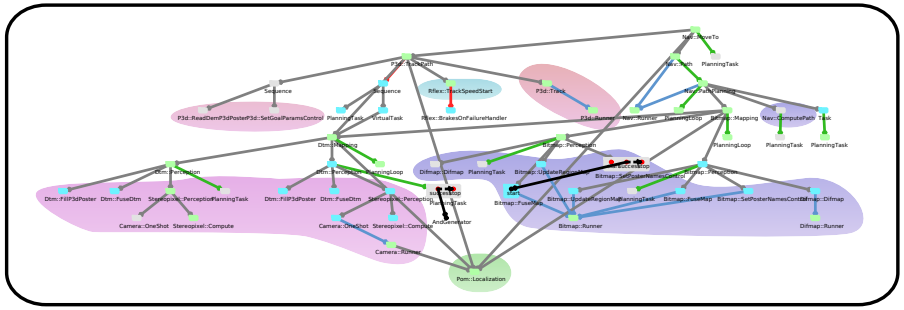
A robotics integration framework through plan management

- specification and development of tasks/events
- small plan generation tool
- integrated applications
- integrated simulation
- test-driven development

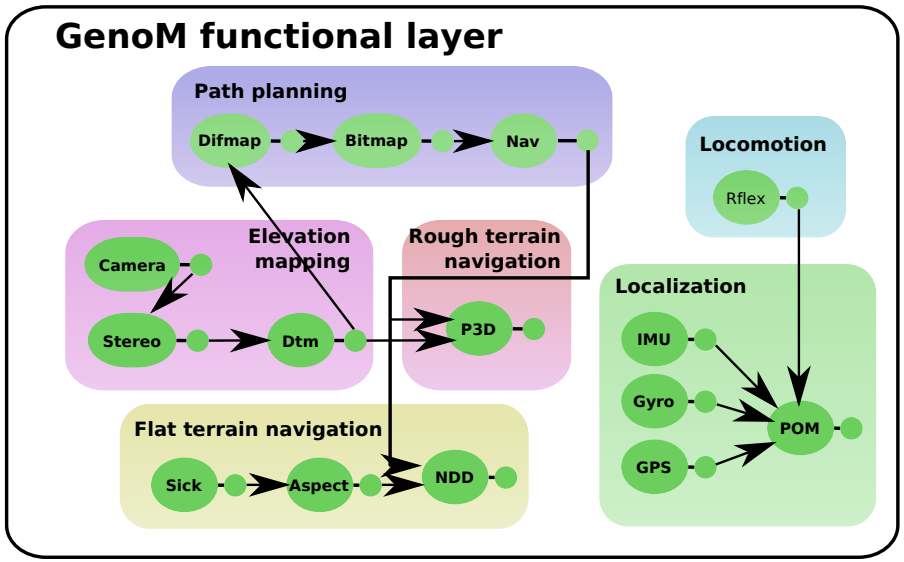
Mono-robot on Dala

Integration and validation on top of an existing functional layer framework (GenoM)

Roby plan



GenoM functional layer



Mono-robot experiment

duration 3-4 minutes per run for 15 meters

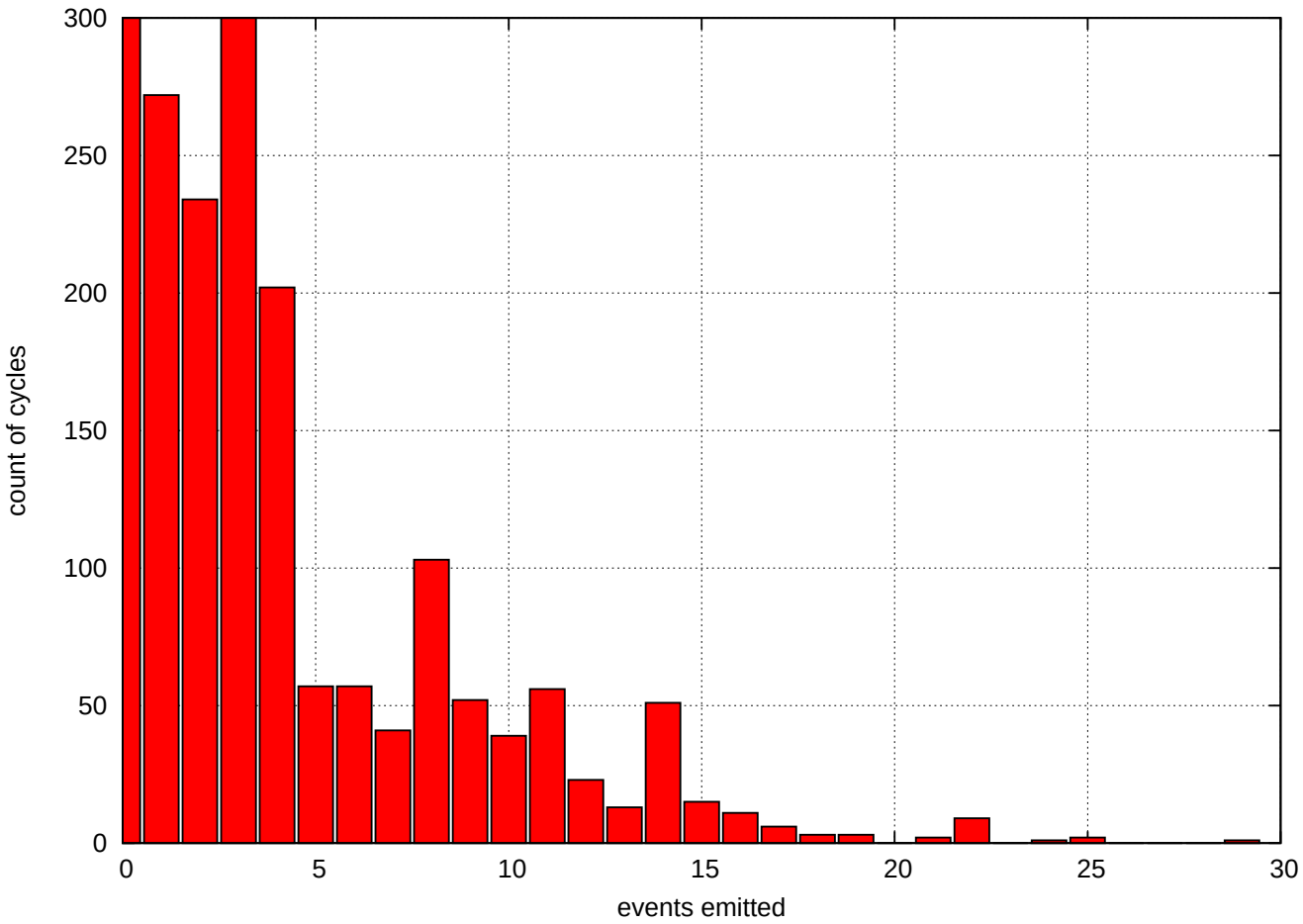
nominal cycle size 100 milliseconds

global execution statistics

- mean execution cycle size: 105
- 7776 emissions. **max**: 29/cycle
- 3604 commands called. **max:** 18/cycle
- 281 transaction commits. **max:** 3/cycle

Mono-robot experiment

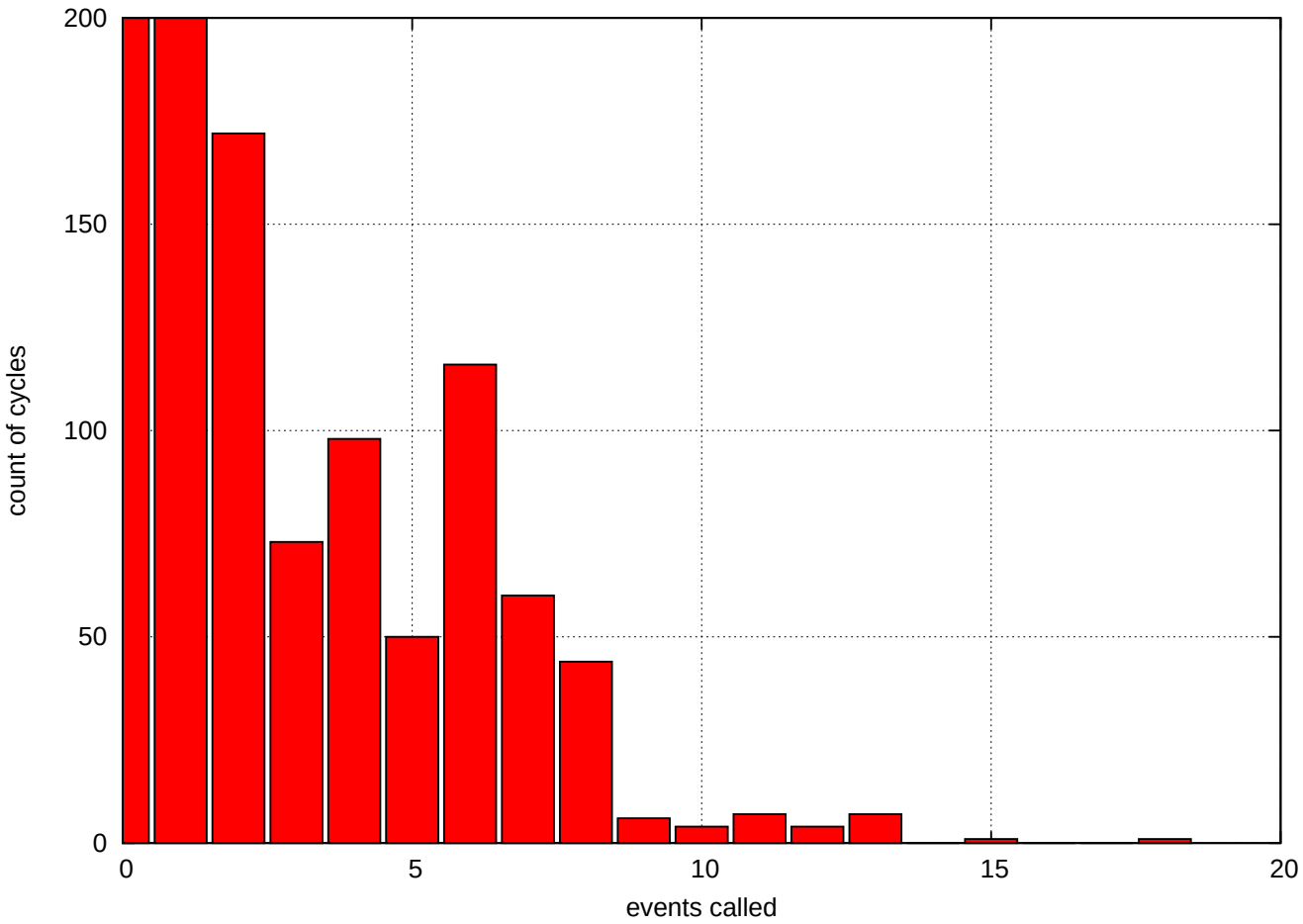
Count of cycles vs. simultaneous event emissions



$\Rightarrow$ numerous cases of simultaneous event emission

Mono-robot experiment

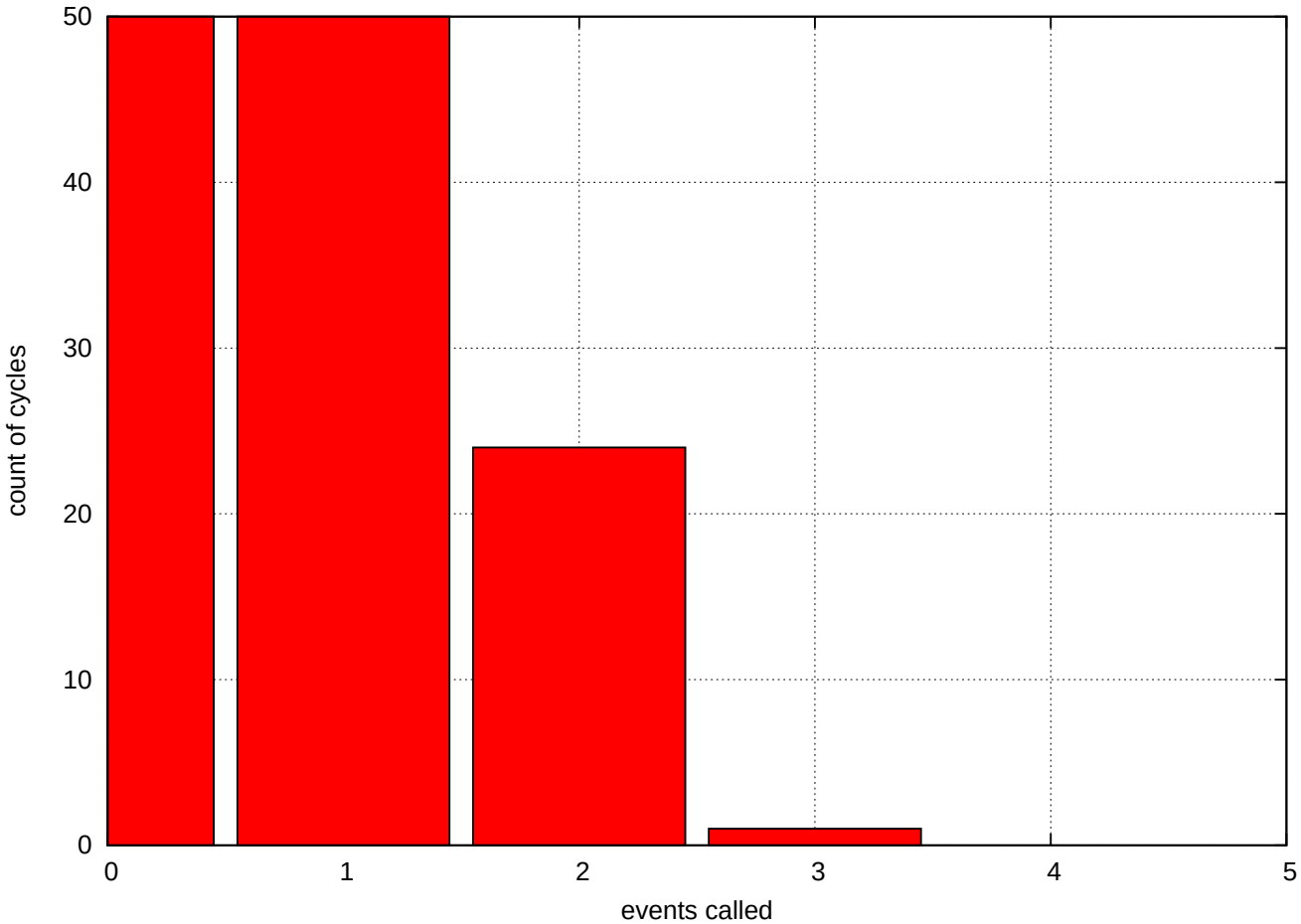
Count of cycles vs. simultaneous event calls



$\Rightarrow$ numerous cases of simultaneous event calls

Mono-robot experiment

Count of cycles vs. transaction commits

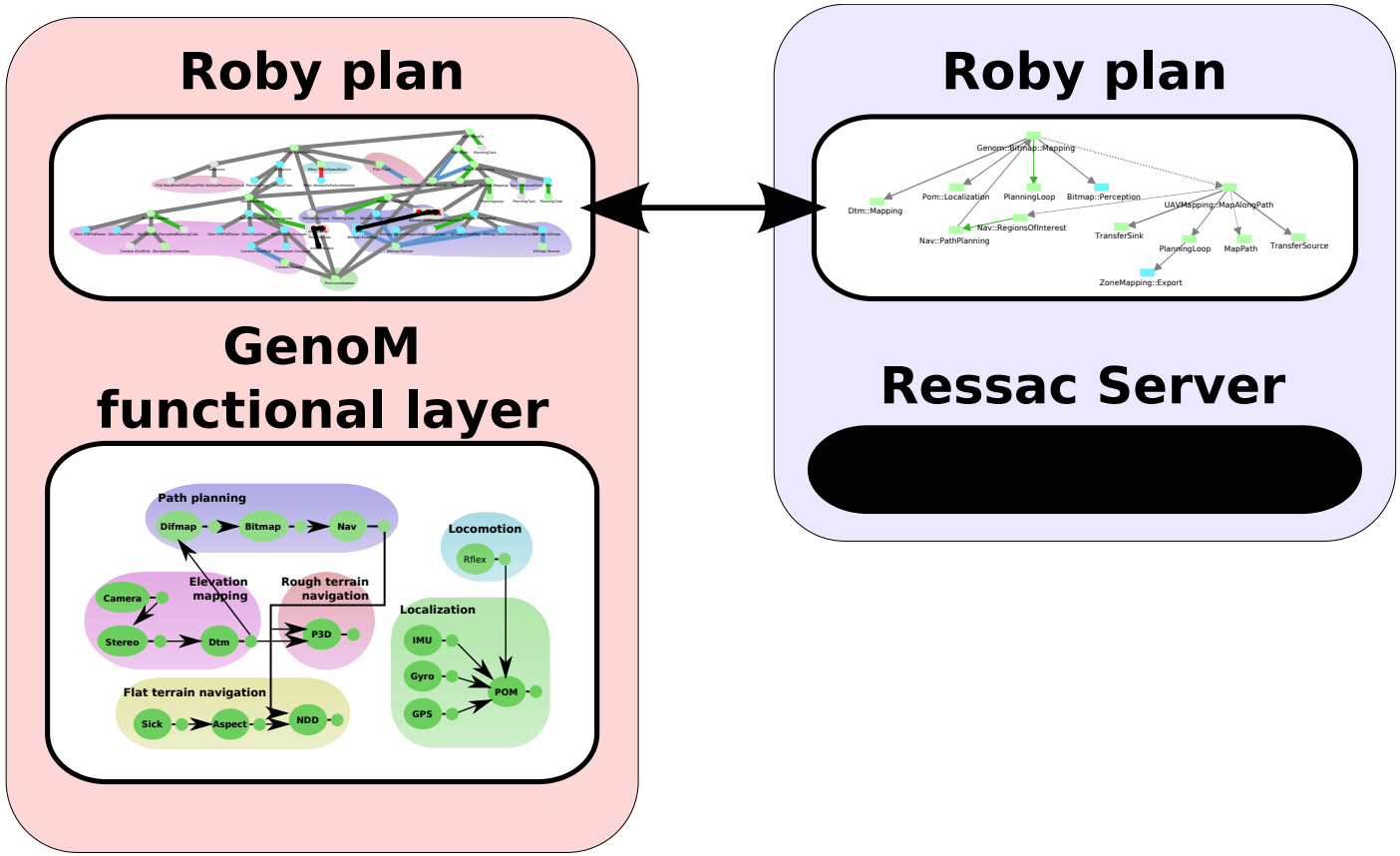


⇒ numerous cases of simultaneous commits of non-conflicting transactions

Bi-robot setup

⇒ both robots have their own Roby controller

⇒ failed because of functional layer issues on Dala

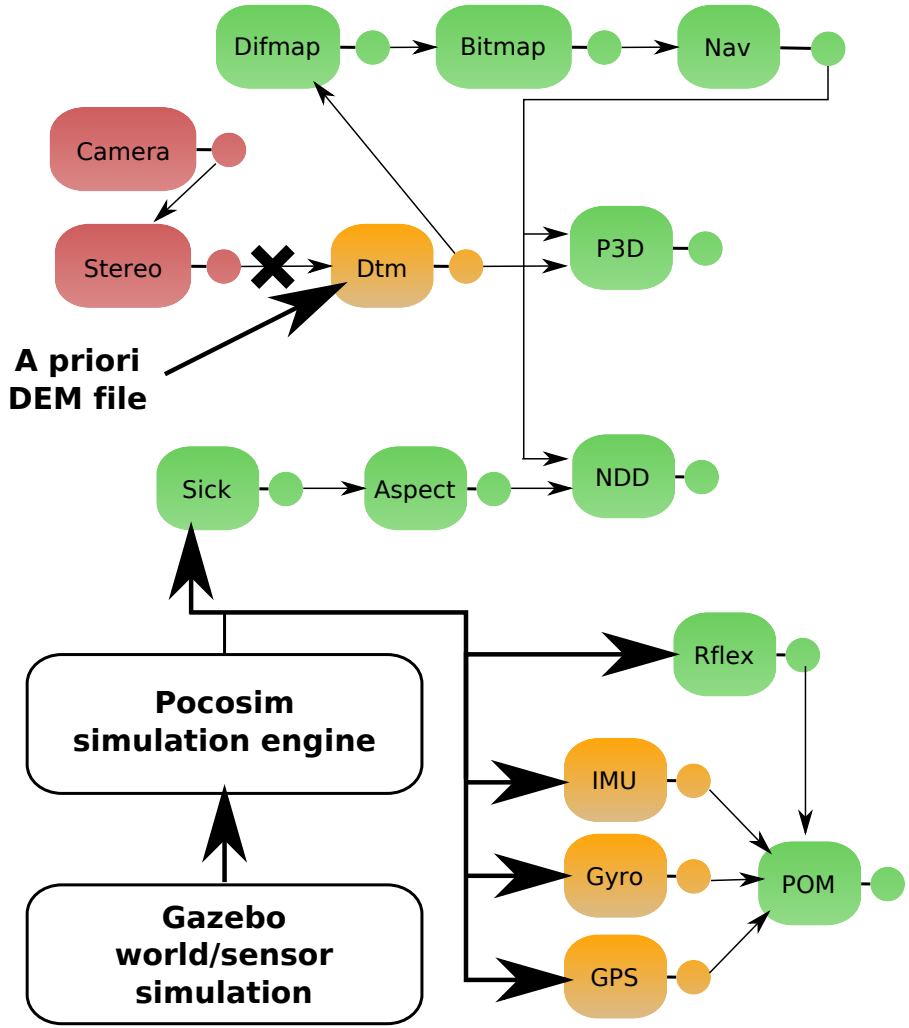


Bi-robot in simulation

Simulation engine

⇒ functional layer almost used as-is

⇒ time control: time synchronization between simulated robots



Bi-robot simulation results

duration 30 minutes per run for 400 meters

nominal cycle size 100 milliseconds

global execution statistics

- mean execution cycle size: 110
- 23490 emissions. **max**: 24/cycle
- 11518 commands called. **max:** 15/cycle
- 1180 transaction commits. **max:** 3/cycle

▸ Video

Summary

What still requires validation ?

- extensive use of plan adaptation operators
- transaction conflict resolution
- multi-robot validation – only bi-robot

Conclusion

- flexible enough to adapt to existing systems
- fast enough to manage a real-world, complex system
- built around the aim of being an application integration framework for robotics

6. Conclusion & Future Work

Contributions

1. **A rich plan model**
   - with separated activity and temporal structures
   - task graphs
   - adapted to multi-robot systems

Contributions

1. **A rich plan model**
2. **A complete execution scheme**
   - able to handle the specificities of the plan model
   - rich error handling procedures
   - multi-robot plan execution

Contributions

1. **A rich plan model**
2. **A complete execution scheme**
3. **A tool for simultaneous plan execution and adaptation**
   - adaptation of the notion of transaction from DBs
   - usable as whiteboard in multi-robot context

Future work

1. **State and time prediction**
   - explicit representation of time
   - integration of temporal constraint networks
   - integration of a causal model: state projection

Future work

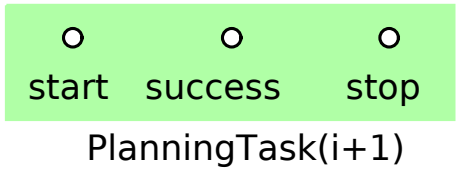
1. **State and time prediction**
2. **Planner/planner interaction through plans**
   - integration of real planners from diverse plan models using state projection
     - plan ahead-of-time
     - forestall faults related to changes in the projection
     - integration of repair capabilities
   - definition of loose planner-planner interaction protocols
     - $\Rightarrow$ plan merging
   - integration of multi-robot cooperation protocols

Future work

1. **State and time prediction**
2. **Planner/planner interaction through plans**
3. **Long term goals**
   - formalization of the approach
     - $\Rightarrow$ prove equivalence between our model and others
     - $\Rightarrow$ represent execution properties in a formal model
   - getting it used by more than one person

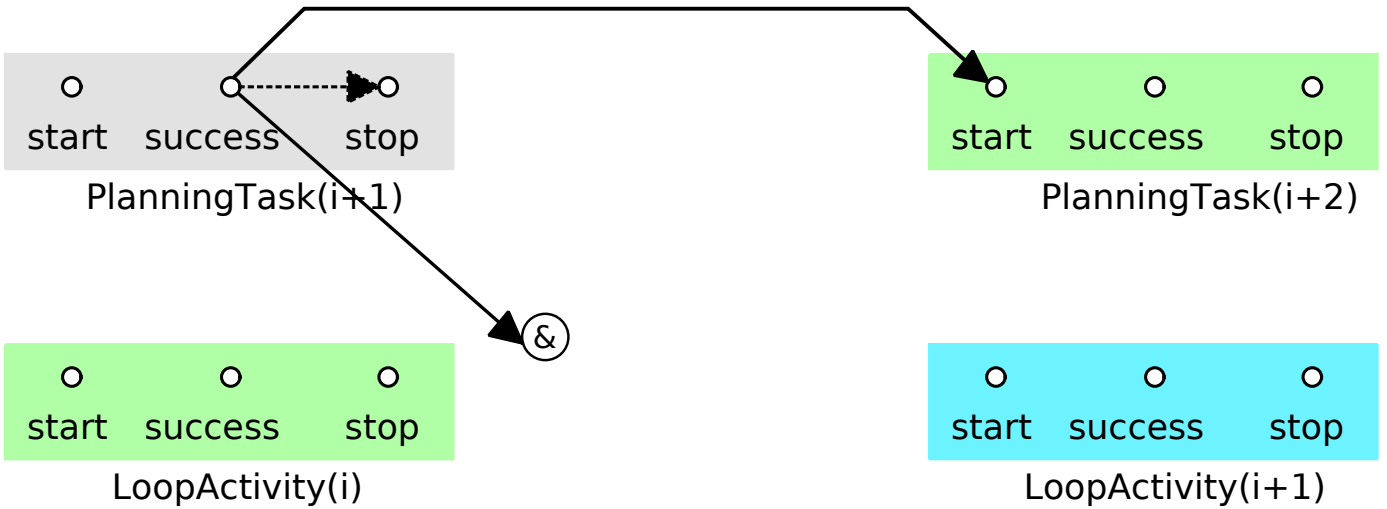
Thank you for your attention !

Looping: global structure



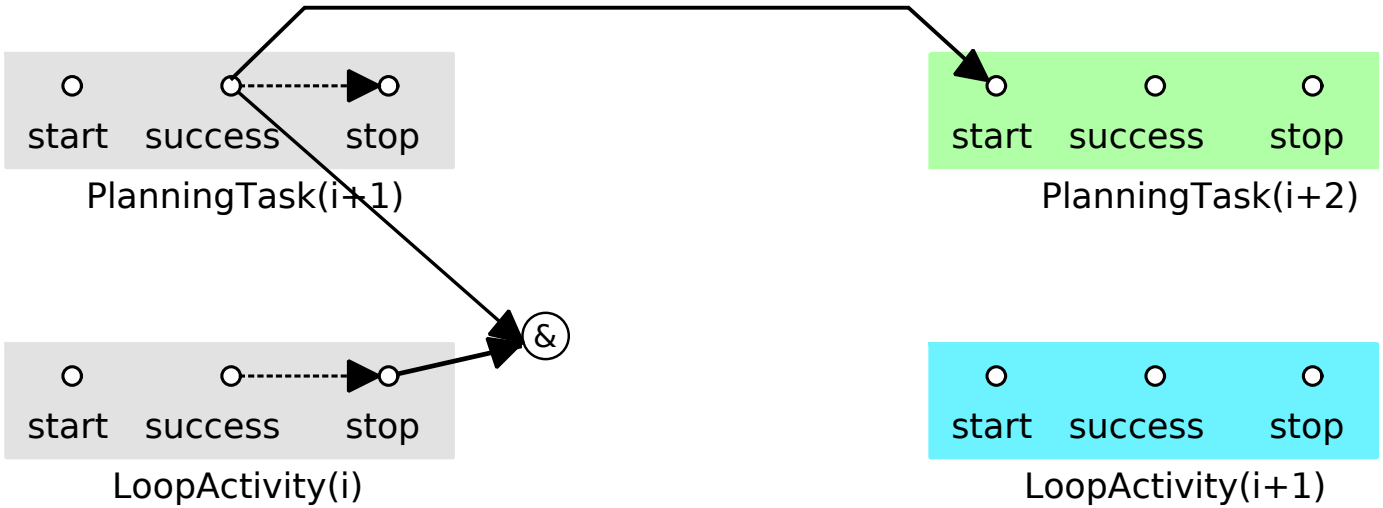
Return

Looping: global structure



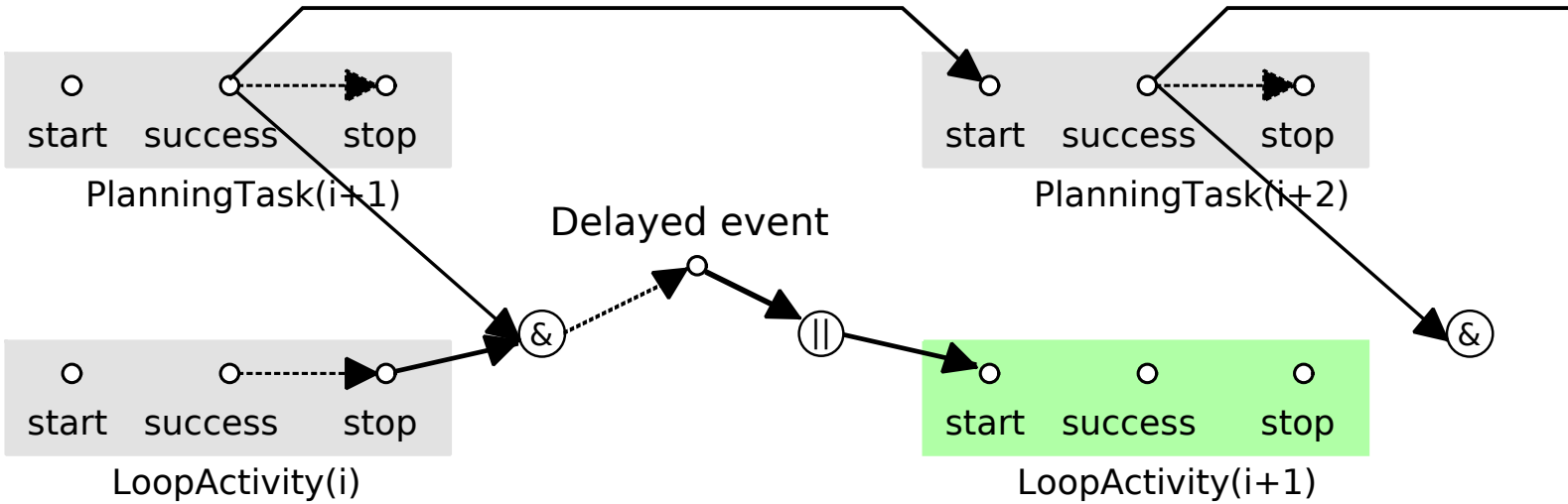
Return

Looping: global structure



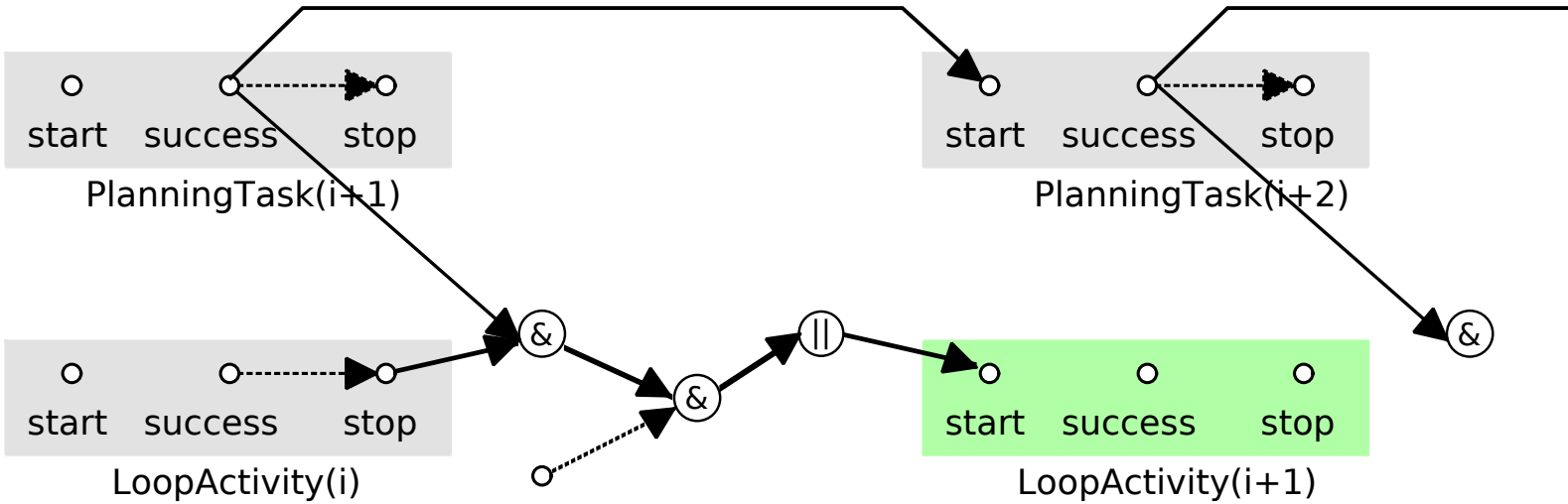
Return

Looping: global structure



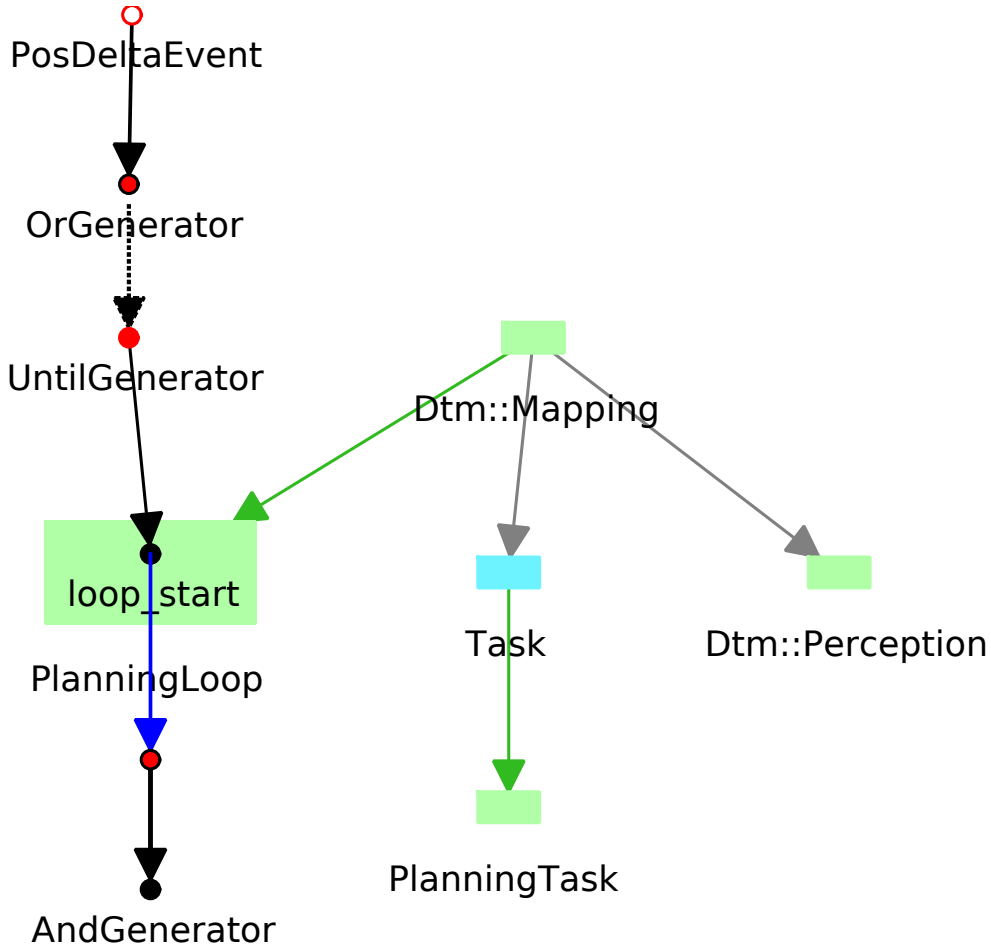
Return

Looping: global structure



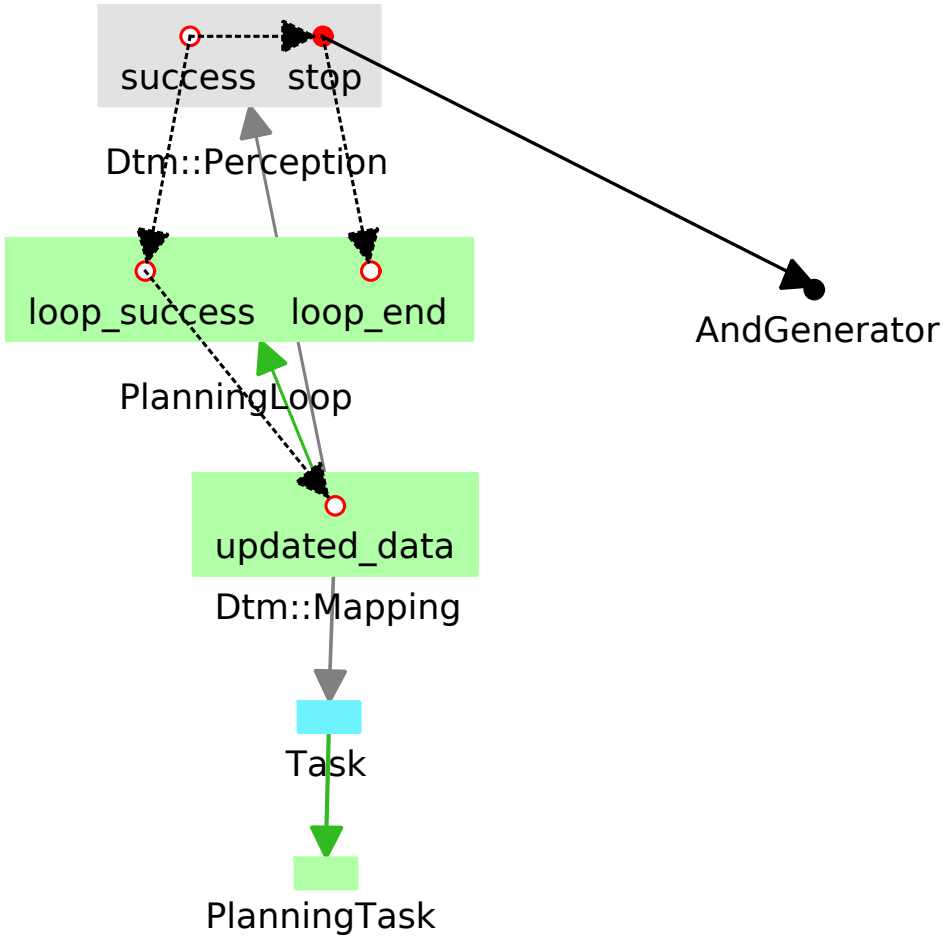
Return

So, what if the algorithms/planning are not fast enough ?



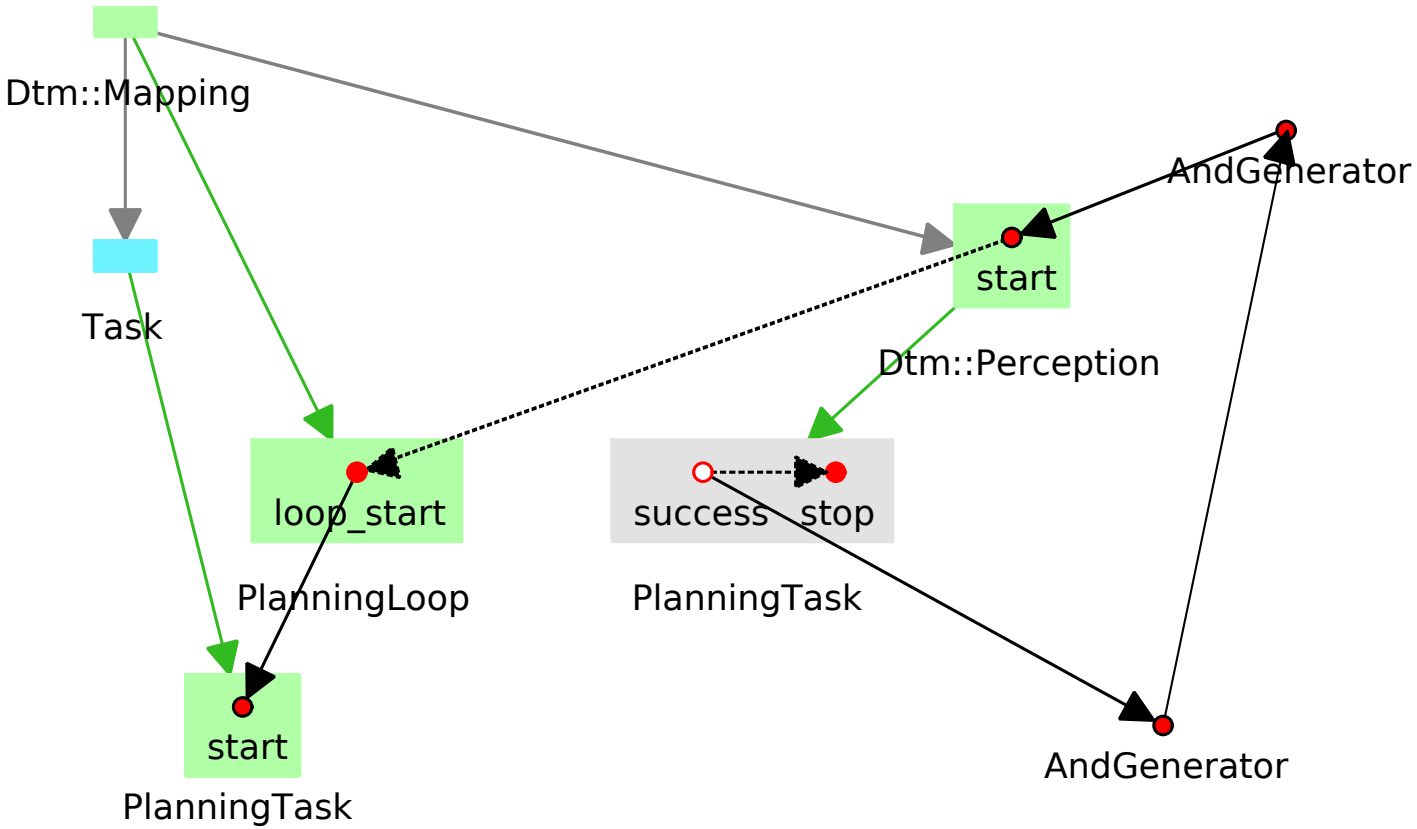
Return

So, what if the algorithms/planning are not fast enough ?



Return

So, what if the algorithms/planning are not fast enough ?



Return

Fault injection pattern

- replace the real task with a fake one
- when the real task is properly interrupted, forward its stop event to the wanted fault event