**Rocoma**

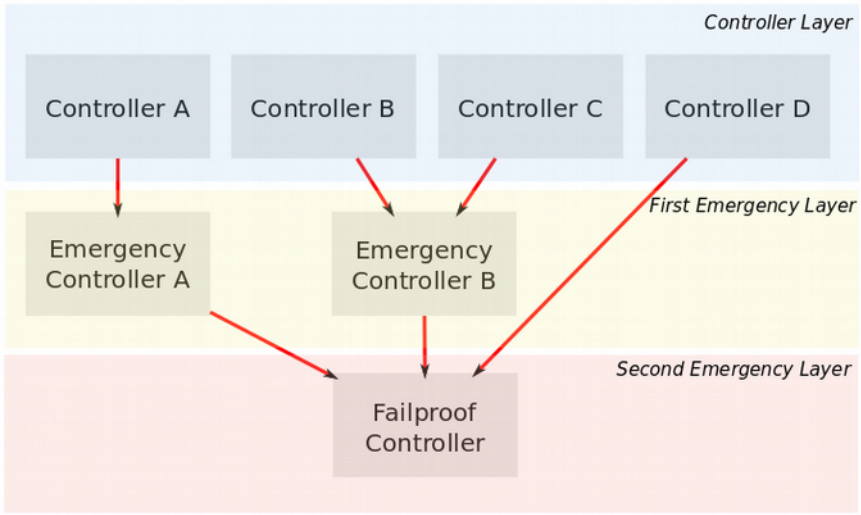
# **Ro**bot **Co**ntroller **Ma**nager

- Safe, two-layered, emergency stop procedure
- Switching between different controllers
- Multi-threaded features (higher controller frequencies)
- Run-time controller loading using the ROS pluginlib
- Emergency stop, controller switching via service call
- Notify other nodes of an action via rostopic

**Rocoma**

# Safety Mechanism

- Optional first emergency layer allows smart emergency stops
- Second layer ensures proper behavior in states impossible to recover from
- Separate interfaces prevents using controllers in an unintended layer



**Rocoma**

# Types of Controllers

## Controller

- Can be activated by switching controllers
- Performs a complex task, that can fail, or even throw exceptions
- Initialization can be very time consuming and take multiple time steps
- Has access to extensions (e.g workers)
- ControllerRos
  Access to the ROS nodehandle (!constructor)

## Emergency Controller

- Activated if the controller update fails, an exception is thrown or an emergency stop is triggered
- *Can fail, throw exceptions*
- Has an additional, fast initialization method
  ( $t_{init}$ << dt )
- Has access to extensions (e.g workers)
- EmergencyControllerRos
  Access to the ROS nodehandle (!constructor)

## Failproof Controller

- Activated if the controller update fails, an exception is thrown, an emergency stop is triggered **AND** no emergency controller is registered
- Activated if the emergency controller update fails, throws an exception or a second emergency stop is triggered
- Has no initialization function
- Can't fail nor throw exceptions (leads to a crash)
- No access to extensions, ROS

Roco

# Robot Controllers

- The roco package is a collection of interfaces:
  - Controller adaptee interfaces
  - Controller adapter interfaces
  - State and command interfaces
  - Worker manager interface
  - ( Time interface )
- Provides controller tuple implementation

# Controller Interface

- roco::ControllerAdapteeInterface
  - create(dt) → Creates the controller (use instead of constructor)
  - initialize(dt) → Controllers are initialized before they are activated
  - advance(dt) → Updated the controller and sets the commands
  - reset(dt) → Called instead of initialize, when already initialized
  - preStop() → Prepares a controller for stop
  - stop() → Stops the controller
  - cleanup() → Frees memory etc. (use instead of destructor)
  - swap(dt, state) → Called instead of initialize/reset, if defined
  - getSwapState(state) → Provides controller state information

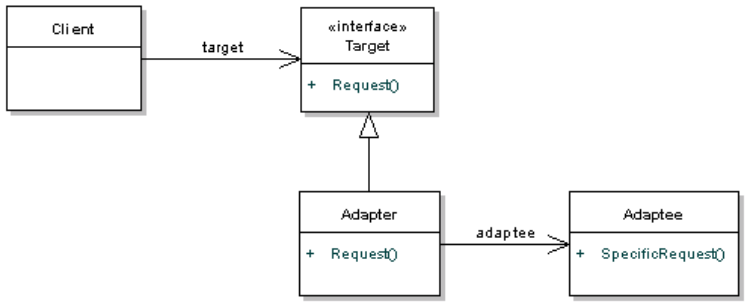
# Emergency / Failproof Controller Interfaces

- roco::EmergencyControllerAdapteeInterface
  - initializeFast(dt) → fast initialization method ( dt >> $t_{init}$ )
- roco::FailproofControllerAdapteeInterface
  - create(dt) →create controller (use instead of constructor)
  - advance(dt) → sets commands / can not fail !
  - cleanup() → destroys controller (use insted of destructor)

**Rocoma**

# Adapter Pattern

- Adaptee: Concrete controller implementation
  Adapter: Decorates adaptee (e.g. logging), exposes it to the client with the Target interface
  Target: An interface for controller adapters
  Client: The controller manager




- Adapter allows to use controllers with a different interface, rocoma only “knows” the adapter interface

**Rocoma**

# Emergency Stop

**Table 1a: Emergency Stop on OK state**

| Timestep | Thread 1 | Thread 2 |
|---|---|---|
| 1 | controller.advanceController(dt)<br>⚡ returns false ⇒ emergency stop | |
| 2 | emcy_controller.initializeControllerFast(dt) | controller.preStopController()<br>controller.stopController() |
| 3 | emcy_controller.advanceController(dt) | |
| 4 | emcy_controller.advanceController(dt) | |
| 5 | emcy_controller.advanceController(dt) | |

**Table 1b: Emergency Stop on OK state, emcy controller stopping**

| Timestep | Thread 1 | Thread 2 | Thread 3 |
|---|---|---|---|
| 1 | controller.advanceController(dt)<br>⚡ returns false ⇒ emergency stop | | emcy_controller.preStopController()<br>emcy_controller.stopController() |
| 2 | failproof_controller.advanceController(dt) | controller.preStopController()<br>controller.stopController() | |
| 3 | failproof_controller.advanceController(dt) | | |
| 4 | failproof_controller.advanceController(dt) | | |
| 5 | failproof_controller.advanceController(dt) | | |

**Table 2: Emergency Stop on EMERGENCY state**

| Timestep | Thread 1 | Thread 2 |
|---|---|---|
| 1 | emcy_controller.advanceController(dt)<br>⚡ returns false ⇒ emergency stop | |
| 2 | failproof_controller.advanceController(dt) | emcy_controller.preStopController()<br>emcy_controller.stopController() |
| 3 | failproof_controller.advanceController(dt) | |
| 4 | failproof_controller.advanceController(dt) | |
| 5 | failproof_controller.advanceController(dt) | |

**Rocoma**

# Switching Controllers

- Controllers are switched in a different thread
- The old controller advances until new controller is initialized
- Controllers can be switched from any state (OK, Emcy, Failproof)
- The old controller can pass data/state to the new controller

**Table 3: Controller switching to controller2**

| Timestep | Thread 1 | Thread 2 |
|---|---|---|
| 1 | controller.advanceController(dt)<br>⚡ switchController(controller2) | |
| 2 | controller.advanceController(dt) | controller.preStopController()<br>while(controller2.isBeingStopped()){}<br>controller.getControllerSwapState(state)<br>controller2.swapController(dt,state)<br>⇒ switch active controller to controller2 |
| 3 | controller.advanceController(dt) | |
| 4 | controller.advanceController(dt) | |
| 5 | controller.advanceController(dt) | |
| 6 | controller.advanceController(dt) | |
| 7 | controller.advanceController(dt) | |
| 8 | controller.advanceController(dt) | |
| 9 | controller2.advanceController(dt) | controller.stopController() |
| 10 | controller2.advanceController(dt) | |
| 11 | controller2.advanceController(dt) | |

**Controller Implementation**

# Where to start ?

- There is an example package → rocoma_example
- Well documented with doxygen rocoma_doc
- catkin_create_roco_pkg script in roco that creates a controller template for you
- Several implementations that can be used as a template (e.g. m545_hip_balancing controller, anymal_ctrl_trot)

**Rocoma:**

# Pass States on Switching

- Switch from MyController2 to MyController
  - Get state from MyController2

```
bool MyController2::getSwapState(roco::ControllerSwapStateInterfacePtr& swapState) {
  swapState.reset( new my_package::MyState(someState);
  return true;
}
```

  - Call swap on MyController with the obtained state

```
bool MyController::swap(double dt, const roco::ControllerSwapStateInterfacePtr& swapState) {

  bool success = MyController::initialize(dt);

  if(success) {
    my_package::MyState * myState = dynamic_cast<my_package::MyState *>(swapState);
    if(myState != nullptr) {
      this->someState_ = myState->getSomeState();
    }
  }

  return success;
}
```

  - Defaults to initialize if not provided by the user

**Controller Implementation**

# Start Workers in Controllers

- Use the worker manager if you want to start new threads in your controllers (No rule w/o exception e.g. std::async)
- Allows setting priorities (pthread)
- Example: Start worker in initialize, stop worker in preStop

```
// Start a worker that publishes
roco::WorkerOptions publishWorkerOptions;
publishWorkerOptions.autostart_ = false;
publishWorkerOptions.frequency_ = 30.0;
publishWorkerOptions.name_ = "myPublishWorker";
publishWorkerOptions.priority_ = 0;
publishWorkerOptions.callback_ = boost::bind(&MyController::publishWorker, this, _1);
publishWorkerHandle_ = addWorker(publishWorkerOptions);
startWorker(publishWorkerHandle_);
```

```
bool MyController::preStop() {
  cancelWorker(publishWorkerHandle_, true);
  return true;
}
```

**Controller Implementation**

# Export Controller as a Plugin

- Export controller via macro

```
#include "rocoma_plugin/rocoma_plugin.hpp"
ROCOMA_EXPORT_CONTROLLER(MyControllerPlugin, my_model::State, my_model::Command, my_controller::MyController)
```

- Add a plugin description file

```
<library path="lib/libmy_controller">
  <class type="MyControllerPlugin"
base_class_type="rocoma_plugin::ControllerPluginInterface<my_model::State, my_model::Command>">
  </class>
</library>
```

- Export the plugin description file

```
<export>
  <rocoma_plugin plugin="${prefix}/my_controller_plugin.xml" />
</export>
```

Controller Implementation

# Controller Tuples

- Combines n controllers to a tuple
  - Order of execution (C1, C2, …, Cn)
  - If one controller fails, the others aren‘t executed
- Things to remember
  - this pointer always accesses the tuple controller,
    thus this→initialize(dt) calls the init function of all controllers
  - Swap function might get a bit complicated since you have to check against a TupleState first

# What Rocoma can not do

- State and command classes are shared, thus controllers using different classes for state and/or command can not be handled by the same controller manager instance
- Timing on your functions is not supervised
  - make sure you have no deadlocks
  - InitializeFast() and advance() must fulfill the timing constraints
- …