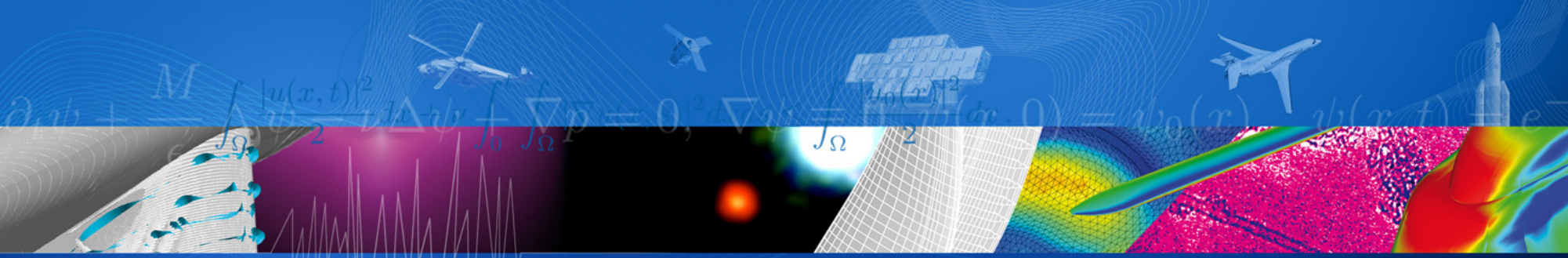


A modeling framework for software architecture specification and validation

Nicolas Gobillot, Charles Lesire, David Doose
firstame.lastname@onera.fr

October 21 2014

ONERA
THE FRENCH AEROSPACE LAB

return on innovation

Introduction

Robotic system

A robotic system has to enforce some properties:
For example the safety about the environment

- Software architecture: functional chain analysis
  Do not collide into the environment.
- Deployment: contract respect analysis
  Take into account the intrinsic parameters of the robot.
- Software functional analysis
  Obstacle avoidance or emergency stop guarantee.
- Real-time: schedulability analysis
  The actions are done in a finite and known time.

Introduction

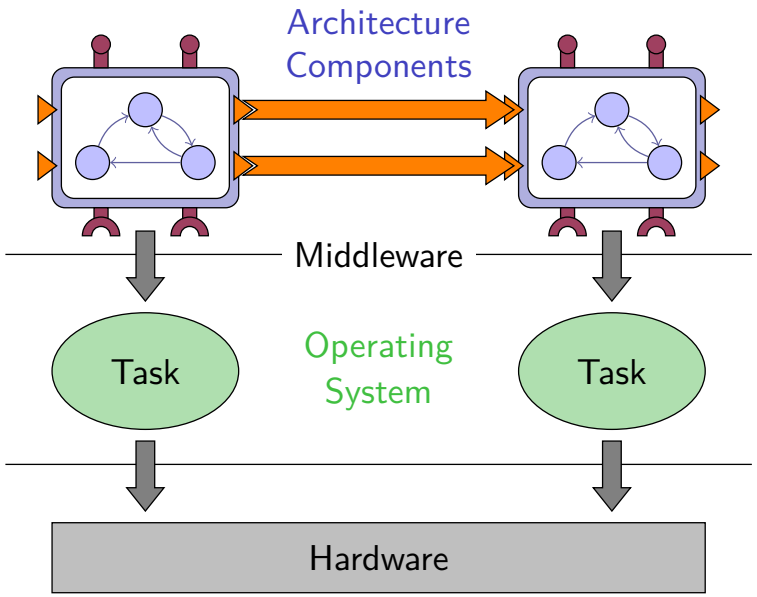
Usual practices:

When we design complex architectures, we often use component or task-based software but sometimes it is not enough to obtain a safe robot behaviour.

Introduction

Component-based modular architecture

- Components
- Activity
- Architecture: links the components
- Allocated onto the Operating System tasks through the middleware
- Executed on a hardware platform



Introduction

Solution:

Refine the component model (MAUVE DSL) to smaller parts in order to refine the analysis (MAUVE tools).

Introduction

The MAUVE DSL

MAUVE is a modelling framework for specifying, analysing and generating component-based robotic architectures.

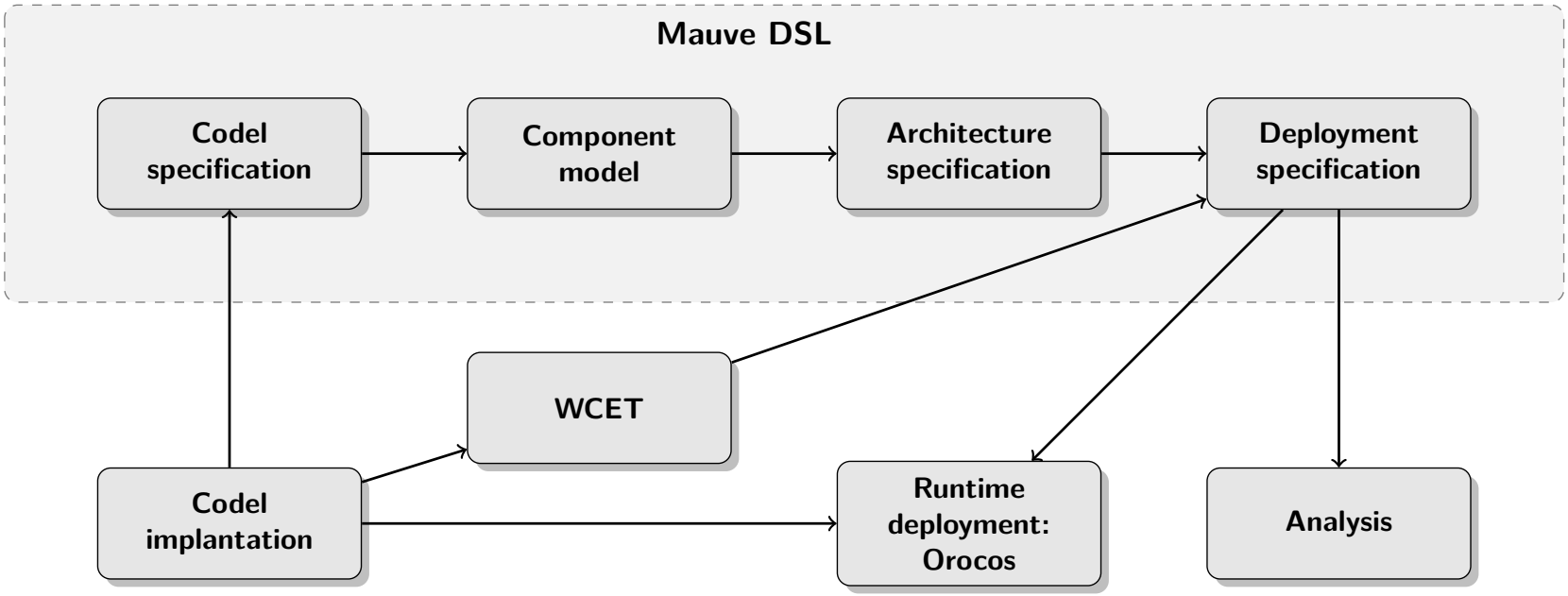
Why a new framework ?

Safety concerns are primordial to autonomous robots. Many existing DSLs can be found but none provides analysis tools:

- precise timing analysis
- contract checking at architecture level

MAUVE DSL elements

- codels: program functions
- components: *shell* and *core*
- architecture: component organisation
- deployment: architecture instantiation



MAUVE DSL elements

Components

Codels

- Elementary Code: the elementary functions that defines a specific functionality
- Separation of concerns: codels are independent of the model and the middleware

```
codel Detect(img: Img): Pose
codel Track(img: Img): Pose
```

MAUVE DSL elements

Components

Shell

- Properties: to parametrize the component
- Ports/operations: to communicate with other components

```
shell TrackingShell {
 input port image_port: image_type
 output port position_port:
      position_type
 property cameraType: string
}
```

MAUVE DSL elements

Components

Shell

- Properties: to parametrize the component
- Ports/operations: to communicate with other components

```
shell TrackingShell {
 input port image_port: image_type
 output port position_port:
      position_type
 property cameraType: string
}
```

Core

- State-Machine
- Codel call

```
core TrackingCore (TrackingShell) {
 update StateMachine TrackingSM
}

StateMachine TrackingSM {
 initial state Initialize {
  transition then { start(); }
       select Detect
 }
 state Detect {
  run {
   detect();
  }
  transition if (condition())
       select Track
  transition if (condition()) then
       { stop(); } select Cleanup
 }
 state ...
}
```

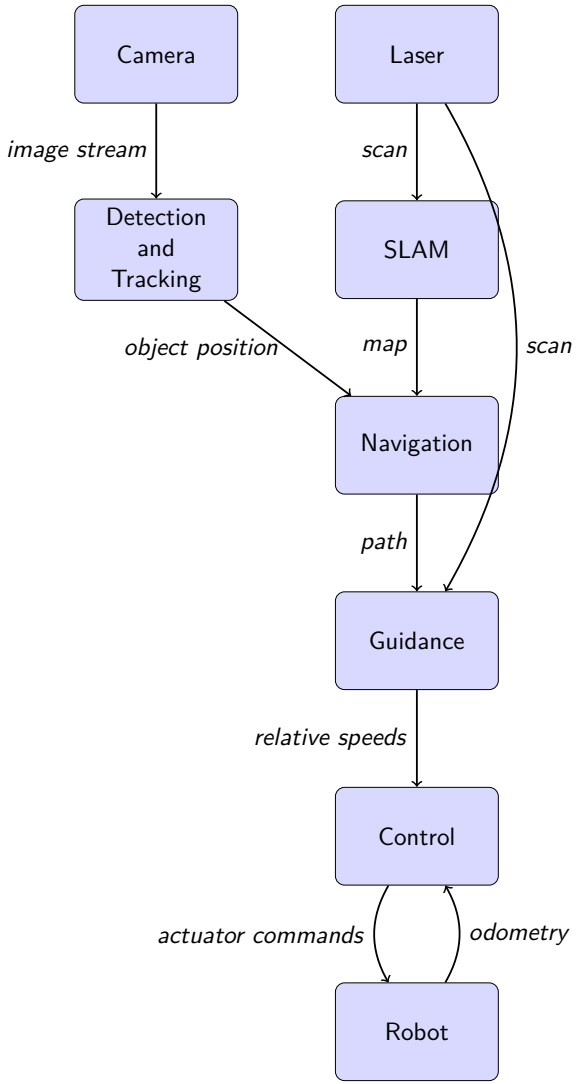
MAUVE DSL elements

Architecture

- Component instantiation
- Connection between components

```
architecture Architecture {
 instance camera: Camera
 instance tracking: Tracking

 connection camera.image_port -> tracking.
      image_port
}
```



Software execution

Deployment

The deployment is target-specific

It depends on

- the hardware: CPU architecture
- the middleware: Orocos (ROS, bare C++, bare Scala)

to generate the executable code from the models.

Software execution

Deployment

The deployment is target-specific

It depends on

- the hardware: CPU architecture
- the middleware: Orocos (ROS, bare C++, bare Scala)

to generate the executable code from the models.
And needs activities to be defined for every component

- Period
- Priority
- Deadline
- Affinity

```
deployment {
 codel command = 12..16
 activity robot {
  priority = 0
  period   = 100
  deadline = 100
 }
}
```

Summary

On-line tools: `https://forge.onera.fr/projects/mauve`

On this website you can find a wiki with useful tips and a tutorial, documents related to the MAUVE project and the lastest releases of the MAUVE DSL.
MAUVE is available under GPL licence.

Work in progress

Contracts for formal analysis are in the DSL but not yet parsed.
Generator:

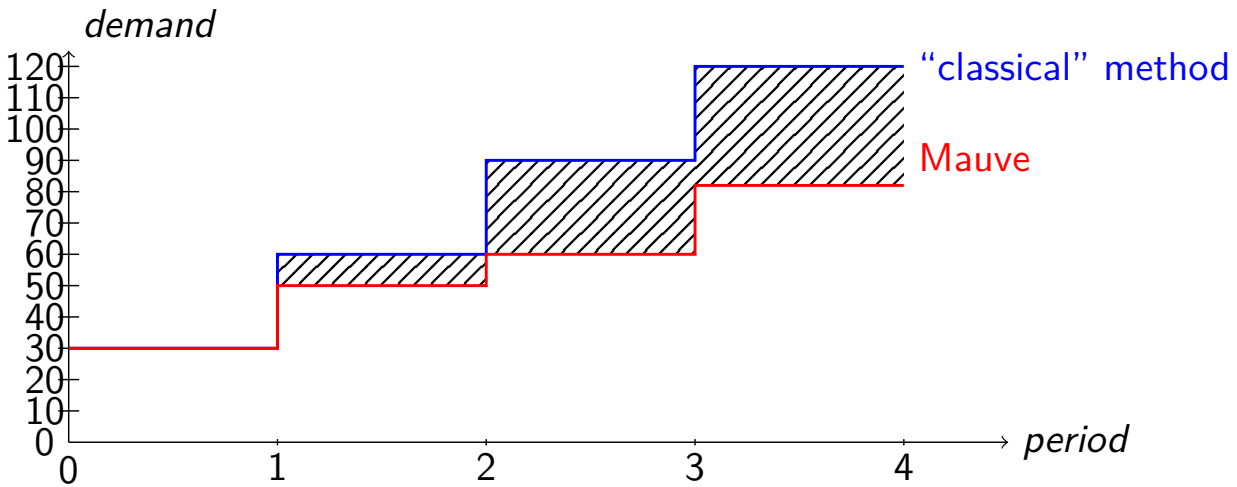
- other middlewares: ROS, bare C++, bare Scala.
- traces for execution analysis

Software analysis

Real-time analysis

Task model

- One task per component
- The "classical" method using the task's execution time is not precise enough
- MAUVE uses the task model to compute an accurate component execution time



Software analysis

Real-time analysis

- Use the ports and the operations of the components
- Find a *Periodic State-Machine* (PSM) from the state-machine and the component's interface
- Generate execution traces from the PSM
- Adapt the usual fixed point algorithms to use the traces:

$$\mathcal{R}_i^0 = \mathcal{T}_i^+(0)$$

$$\mathcal{R}_i^{n+1} = \sum_{j \leq hp(i)} \mathcal{T}_j^+(\mathcal{R}_i^n) + \mathcal{T}_i^+(0)$$

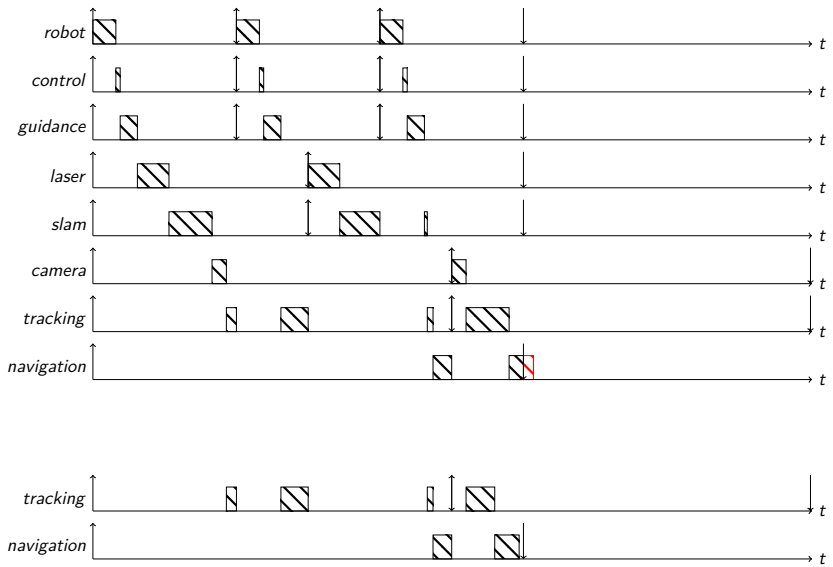
- Compute the Worst Case Response Time (WCRT) for all the components
- A component is schedulable if its response time is lesser than its deadline.
- The architecture is schedulable if all its components are schedulable.

Software analysis

Results

Table and execution timelines from the same architecture, assuming the *tracking* component is monolithic and taking into account its state-machine.

component	prio.	WCET	WCRT	WCRT+	period/deadline
robot	0	16	16	16	100
control	1	3	19	19	100
guidance	2	12	31	31	100
laser	3	22	53	53	150
slam	4	30	83	83	150
camera	5	10	93	93	250
tracking	6	30	237	237	250
navigation	7	30	**307**	**297**	300



Summary

On-line tools: `https://forge.onera.fr/projects/mauve`

On this website you can find a wiki with useful tips and a tutorial, documents related to the MAUVE project and the lastest releases of the MAUVE DSL.
MAUVE is available under GPL licence.

Work in progress

Extract probabilistic timings from traces of the components to compute probabilistic execution times.
Contract checking (with Tina tool by generating Petri nets)

Conclusion

The MAUVE toolchain allows:

- High level architectures to low level component's state-machine specification
- Target specific parameters definition
- Real-time analysis
- Code generation

Work in progress:

- Contracts for formal analysis are in the DSL but not yet parsed.
- Generator:
  - other middlewares: ROS, bare C++, bare Scala.
  - traces for execution analysis
- Extract probabilistic timings from traces of the components to compute probabilistic execution times.
- Contract checking (with Tina tool by generating Petri nets)

Questions

Try it by yourself: `https://forge.onera.fr/projects/mauve`