Formal Methods for Verification and Control of Robotic Software Architectures

Charles Lesire

ONERA/DTIS, Université de Toulouse
Outline

1. Introduction
2. Component-based Middleware and Schedulability
3. DSL for Skills Modeling
4. Petri-net based Mission Programming
5. Conclusion
Context

Photo by Matthew Henry
mission programming

Photo by Matthew Henry
Context

mission programming

software-level guarantees?

mission-level guarantees?
Contributions
Contributions
Contributions
Contributions
Contributions
Contributions
Contributions
Contributions

VideoStream → VO-based Loc. → Laser-based Avoidance → Hover → GoTo

GPS-based Loc. → DepthMap-based Avoidance → TakeOff → Land

/imu
/gps
/visual_odometry
/laser
/mapping
/obs_avoid
/depth_map

VideoStream

GPS

_lidar

/setpoint

/obstacle

/dji_guidance

/dji_sdk

status

images

speed

pose

images

imu

depth

pose

image

pose

visual_odometry

laser

mapping

/obs_avoid

/depth_map

Camera

Localization

Laser

Mapping

Obstacle Avoidance

GPS-based Loc.

DepthMap-based Avoidance

Laser-based Avoidance

Hover

Video Stream

GPS

_lidar
Contributions
Contributions
Contributions
Contributions

GPS\_loc \parallel \{\text{takeOff} \mid \{\text{laserAvoid} \parallel \{\text{goTo(wp)} \mid \{\text{goTo(wp)} \parallel \text{stream}\}\}\}\}\}

ASPiC

Robot Skills

MAUVE Toolchain
Outline

1. Introduction

2. Component-based Middleware and Schedulability

3. DSL for Skills Modeling

4. Petri-net based Mission Programming

5. Conclusion
## The MAUVE Toolchain

- **MAUVE DSL** to model component-based architectures
- **MAUVE Runtime** to execute real-time architectures
- **MAUVE RT Analyses** to evaluate WCET/WCRT of components
MAUVE DSL and Toolchain

- Architecture modeling using a DSL\(^1\)
- Code Generation (ROS/OROCOS)
- Model Analysis compliant with Execution Analysis

MAUVE Runtime

MAUVE Runtime\(^2\) rationales:

1. Provide a C++ API for programmers compliant with models
2. Provide reconfiguration mechanisms
3. Masterize the synchronization of real-time tasks
4. Provide an execution model both formal (to ease analyses) and expressive (to allow implementing complex behaviors)

MAUVE Runtime

A component-based architecture middleware to design architectures with:

- **components**, i.e. tasks that execute code
- **resources**, that own data
- connections between components and resources
- real-time activities assigned to components
- mechanisms to **reconfigure** parts of the architecture in real-time
MAUVE RT analysis

- **Schedulability analysis:**
  - Determine if the software components are executed on time.

- **Worst Case Execution Time (WCET):**
  - *longest CPU time* passed to compute a piece of code without any interaction (alone on the CPU)
  - depends on both the source code and the hardware.
  - an input for the schedulability analysis and the computation of the WCRT

- **Worst Case Response Time (WCRT):**
  - *longest time* spent by a “task” from its beginning to its end
  - takes into account preemptions and delays from other “tasks”
  - A “task” is schedulable is its WCRT est lower or equals to its deadline
MAUVE RT analysis

- WCET using probabilistic analysis\(^3\)
  - gather real executions traces of the robot
  - Extreme Value Theory (EVT) to infer rare events
  - computes a probabilistic WCET
  - metrics on the applicability of the theory

---

Corail / ROS2

- Integration of the RT core as a ROS2 extension\textsuperscript{4}

Outline

1. Introduction

2. Component-based Middleware and Schedulability

3. DSL for Skills Modeling

4. Petri-net based Mission Programming

5. Conclusion
Robot Skills Models

- Organize the software architecture into functions, or *skills*

→ DSL for Skill Modeling... and tools

Robot Skills Models

- Organize the software architecture into functions, or skills

⇝ DSL for Skill Modeling... and tools⁵

---

Robot Skills DSL

Functional layer organized into SkillSets (e.g., guidance, image_processing or PR2); a SkillSet defines:

- **Data** provided by the skillset / robot;
- **Functions** that can be used on other parts of the model;
- **Resources**, represented as state-machines, that can be used by skills, or that reflect actual state of a system (e.g., a driver);
- **Skills**, i.e. functions provided by the skillset.
A Skill is defined by:

- some **inputs**, i.e. parameters of the skill execution (e.g., position to reach);
- a **precondition** describing some logical condition for accepting execution;
- the list of **used resources**, with constraints (pre, invariants);
- the **terminal modes** of the skills, each described by post-conditions and effects on resources;
- a **progress** period at which the skill will report some progress.
Robot Skills DSL

Full language grammar described at:
http://oara-architecture.gitlab.io/robot-skills/robot_lang/
Robot Skills DSL: Turtlesim

```plaintext
type {
    Pose
    Pen
    Bool
}

skillset turtlesim {
    data position: Pose

    function isSafe(p: Pose): Bool

    resource simulator {
        initial UNAVAILABLE
        extern UNAVAILABLE -> AVAILABLE
        extern AVAILABLE -> UNAVAILABLE
    }

    skill teleport {
        progress=0
        input target: Pose
        precondition sim_avail: resource=(simulator==AVAILABLE)
        invariant sim_inv: resource=(simulator==AVAILABLE)
        mode {
            ARRIVED {}
            COLLISION {}
        }
    }
```
Robot Skills Execution Tools
Robot Skills Managers

**Managers**

- **Data** manager subscribes to internal data and provide getters
- **Resource** manager keeps track of the resources states, from skills request or from external requests
- **Skill** managers implement the management of skills in relation to functional modules

**Generators**

- **ROS2 Generator**
- One package for the ROS `msg`, one for skillset management
- Inherit from the skillset manager to implement the link with the functional modules
- Type mapping between model types and ROS types
- Skills can be triggered using ROS topics
Robot Skills Client Interface

Client Library

- ROS/Python library to get access to data and resources, and control skills
- Simple API that hide ROS constructs
- Generated from a skillset model
Robot Skills Interface: Turtlesim

```python
from turtlesim_interface import TurtlesimSkillsetInterface
from turtlesim.msg import Pose

turtlesim = TurtlesimSkillsetInterface()
teleport = turtlesim.skills.teleport
paint = turtlesim.skills.paint

# draw I
teleport.start(Pose(x=1, y=5))
teleport.wait()
paint.start()
teleport.start(Pose(x=1, y=9))
teleport.wait()
paint.interrupt()
paint.wait()

# draw R
teleport.start(Pose(x=3, y=5))
teleport.wait()
paint.start()
teleport.start(Pose(x=3, y=9))
teleport.wait()
paint.start()
teleport.start(Pose(x=5, y=9))
teleport.wait()
teleport.start(Pose(x=5, y=7))
teleport.wait()
teleport.start(Pose(x=5, y=7))
teleport.wait()
teleport.start(Pose(x=3, y=7))
teleport.wait()
teleport.start(Pose(x=5, y=5))
teleport.wait()
paint.interrupt()
paint.wait()
```
Robot Skills Interface
Safe Inspection Mission by a UVA

https://youtu.be/mZxy16v-tDw

# Outline

1. Introduction
2. Component-based Middleware and Schedulability
3. DSL for Skills Modeling
4. Petri-net based Mission Programming
5. Conclusion
ASPiC

- Task-level (or Mission) Programming:
  - Expressive "language" with complex operations
  - Management of failures in task execution
  - Petri nets as an underlying formal model
  - Execution control based on playing the resulting Petri net model

---

**ASPiC**

- **Task-level (or Mission) Programming:**
  - Expressive "language" with complex operations
  - Management of failures in task execution
  - Petri nets as an underlying formal model
  - Execution control based on playing the resulting Petri net model

- **ASPiC$^6$:**
  - Petri nets with Control-Flow semantics
  - Colored PN with control-flow tokens • and ○
  - Control-flow places $P^\bullet$: entry, internal, exit
  - Modelling of *skills* using specific CFPN
  - Composition operators that (tend to) preserve properties by construction

---

ASPiC Operators

- Composition operators to build up plans from skills
- Integrate exception propagation
- Each operator modelled as a CFPN with placeholder transitions
- Operators: sequence, choice, concurrency, if/then/else, race concurrency, loop/retry
ASPiC Properties

- **Well-formedness**
  - preserved when connecting with a handler
  - preserved by operators

- **Cleanness** of control-flow

\[
\forall M : M_0 \xrightarrow{*} M \land M(p_x) \neq \emptyset \Rightarrow \\
M(p_x) \in \{\bullet, \circ\} \land (p \in P^\wedge \setminus \{p_x\} \Rightarrow M(p) = \emptyset)
\]

- **Control-safe**: \(P^\wedge\) is safe

- Cleanness and control-safety are not ensured by all operators (typically \(N^\wedge\))
Conclusion

Being able to guarantee a correct behavior, from mission-level to functional components, with models compliant to the actual execution.
Thank you for your attention.

Questions?


