Formal models for verification and control of software architectures

Charles Lesire$^{1,2}$

$^1$ONERA/DTIS, University of Toulouse, France

$^2$Artificial and Natural Intelligence Toulouse Institute (ANITI)

IEEE TC-VAS – 1 October 2020
Introduction

MAUVE Toolchain

Robot Skills

ASPiC

Conclusion

Context

Formal models for verification and control  Charles Lesire  IEEE TC-VAS
mission programming

Photo by Matthew Henry

© Charles Lesire
Introduction

MAUVE Toolchain

Robot Skills

ASPiC

Conclusion

Context

mission programming

software-level guarantees?

mission-level guarantees?

Photo by Matthew Henry

© Charles Lesire

Photo by Matthew Henry

© Charles Lesire

Formal models for verification and control

Charles Lesire

IEEE TC-VAS
1. Component-based Middleware and Schedulability

2. DSL for Skills Modeling

3. Petri-net based Mission Programming
Outline

1. Component-based Middleware and Schedulability
2. DSL for Skills Modeling
3. Petri-net based Mission Programming
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
- Architecture modeling using a DSL\(^1\)
- Code Generation (ROS/OROCOS)
- Model Analysis compliant with Execution Analysis

---

MAUVE Runtime\textsuperscript{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)

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

![Diagram]

\[
\begin{align*}
S_1 &: 1 \\
S_2 &: 5 \\
S_3 &: 2 \\
E_1 &\rightarrow S_1 \\
E_1 &\rightarrow S_2 \\
E_1 &\rightarrow S_3 \\
S_2 &\rightarrow E_2 \\
S_3 &\rightarrow E_3
\end{align*}
\]
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

Outline

1. Component-based Middleware and Schedulability
2. DSL for Skills Modeling
3. Petri-net based Mission Programming
Robot Skills Models

- Organize the software architecture into functions, or skills

---

Robot Skills Models

- Organize the software architecture into functions, or **skills**
  - DSL for Skill Modeling... and tools\(^4\)

---

Robot Skills Models

- Organize the software architecture into functions, or **skills**
  
  ↝ **DSL for Skill Modeling**... and tools$^4$

---

Robot Skills DSL: Turtlesim

```haskell
1 type {
  Pose
  Pen
  Bool
}
6 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

Introduction
MAUVE Toolchain
Robot Skills
ASPiC
Conclusion

Formal models for verification and control
Charles Lesire
IEEE TC-VAS
and also:

- interface library for integration into StateMachines and BehaviorTree models
- translation towards PDDL models
- translation towards NuSMV State Machines
Outline

1. Component-based Middleware and Schedulability

2. DSL for Skills Modeling

3. Petri-net based Mission Programming
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

---

■ 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\textsuperscript{5}:
  ■ Petri nets with Control-Flow semantics
  ■ Colored PN with control-flow tokens $\bullet$ and $\circ$
  ■ Control-flow places $P^\circ$: entry, internal, exit
  ■ Modelling of skills using specific CFPN
  ■ composition operators that (tend to) preserve properties by construction

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

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

- Cleanness and control-safety are not ensured by all operators (typically \(N_{\otimes}\))
Being able to guarantee a **correct** behavior, from **mission**-level to **functional components**, with models **compliant** to the actual execution.