Design and analysis of real-time robotic architectures

J. Brunel, D. Doose, C. Grand, Ch. Lesire, L. Santinelli

May 23, 2019
Civil and defense applications

- Area or building inspection
- Monitoring of industrial sites
- Long range linear inspection
- Search and rescue operations
- Environmental monitoring
- ...
Software architectures in robotics are:

- complex, modular
- critical (safety guarantees needed)
- dynamic (reconfiguration mechanisms)

Thus, need for:

- models and languages for design
- safety/robustness mechanisms
- safety assessment methods
Robotics software at ONERA

- Before 2007: pure C/C++ code without any middleware; manual thread management
- Since 2008: extensive use of the Orocos middleware
- Since 2014: development of MAUVE DSL for modeling Component-Based Architecture
- Since 2017: development of MAUVE runtime
Approach

MAUVE rationale

- Ease development of component-based architectures
- Formalize best practices backed by our experience
- Analyze real-time constraints
- Prove safe behavior based on formal models

MAUVE framework

- a Domain Specific Language to design components and architectures
- a Runtime to efficiently execute architectures
- real-time analysis tools to process timed execution traces

http://mauve.gitlab.io
MAUVE DSL

- Architecture modeling using a DSL \(^1\)
- Generation of code
- Real-time schedulability analysis \(^2\)

1. gobillot2014simpar.
2. gobillot2016iros.
MAUVE Runtime\textsuperscript{3} rationale:

1. Provide a C++ API for programmers (still 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 an architecture out of:

- 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 Components

A **component** is an entity that supports an activity, i.e. a real-time behavior, composed of:

- a **Shell** (in/out ports and properties)
- a **Core** (processing algorithms)
- a **Finite State-Machine** (behavior)

```cpp
def HelloShell: public Shell {
    Property<string>& who = mk_property("who", "World");
    WritePort<string>& talk = mk_port<WritePort<string>>("talk");
};
```
MAUVE Finite State-Machines

MAUVE runtime provide enhanced clock-based FSM to model complex behaviors:

- **Execution** states (call a Core function)
- **Synchronization** states (pause and wait for a clock trigger)
- **Transitions** between states: ordered with guards

![Diagram of MAUVE FSM states and transitions](image-url)
- A real-time task is created to execute each component.
- Components' clocks are synchronized.
- Deployer can interact with components.
Reconfiguration mechanisms:
- change property values
- replace a part of an element; most common uses:
  - replace an FSM: changes the behavior of the component while keeping the same functions
  - replace a Core: changes the computational part (switch between algorithms)

Keep Component/Resource dependencies (Shell, Core, FSM/Interface)

Reconfiguration implementation:
- preserves the real-time components priorities
- preserves the real-time synchronization of the components
- for reconfiguration timing analysis
Why logging?
- get real-time execution data
- component behaviors
- compute WCET from traces
- real-time analysis
- trace analysis

What logging?
- component FSM changes
- evolution of real-time parameters
- ...

How logging?
- using LTTNG
- with customized events

Path:
\[ E_1, S_1, E_1, E_2, S_2, E_1, S_1, E_1, E_3, S_3 \]

Timeline:
\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \]
Mauve elements logged
- each component: start & end
- each FSM state: begin & end
- each Resource service: begin & end
- each monitor

Data logged
- mauve element name
- component internal clock
- cpu
- timing
- ...

How?
- LTTng
- CTF format with specific messages
- ordered sequence of events

---

4. www.lttng.org
FSM abstraction creation
- for each component
- sequence of its FSM execution

For each FSM (execution) State
- \textit{begin} & \textit{end}
- \textit{active} = \textit{end} \ - \ \textit{begin}
- \textit{response} = \textit{end} \ - \ \textit{release}
- \textit{duration} = \textit{active} \ - \ \sum \text{preemptions}

Component FSM information
- transitions statistics
- cumulative duration
- precise analysis (for each state)
  - ...
Schedulability analysis

- Determine if the software components are executed on time.

The Worst Case Execution Time (WCET)

- The WCET is the **longest CPU time** passed to compute a piece of code without any interaction (alone on the CPU).
- The WCET depends on both the source code and the hardware.
- The WCET is an input for the schedulability analysis and the computation of the WCRT.
- WCET can be obtained using two different methods:
  1. by static analysis;
  2. by probabilistic analysis.

The Worst Case Response Time (WCRT)

- The WCRT is the **longest time** spent by a “task” from its beginning to its end.
- The WCRT takes into account preemptions and delays from other “tasks”.
- A “task” is schedulable if its WCRT is lower or equals to its deadline.

---

5. and also the compiler
Worst Case Execution Time

Requires

- real code and hardware
- executions traces

Method

- from the real executions traces of the robot
- execution time of the core methods is computed
- with Extreme Value Theory (EVT) rare events are inferred

Provides

- probabilistic WCET
- metrics on the applicability of the theory

6. www.lttng.org
**Worst Case Response Time**

- requires the WCET of each execution state
- each execution state can have a deadline
- a deadline is related to a sequence of execution states

**Examples**

- with $d_1$ deadline of $E_1$: state $E_1$ must be completed before $d_1$
- with $d_3$ deadline of $E_3$: executions of $E_1$ and $E_3$ must be completed before $d_3$

**Schedulability analysis**

- new method
- compute WCRT of each execution state of each FSM
- can be compared to its deadline (schedulability)
LTL formula

- LTTng/CTF trace converted into MauveTrace
- each event is a MauveEvent
- predicate: any function \((\text{MauveEvent} \rightarrow \text{Boolean})\)
- LTL syntax: \(\phi : = \text{pred} \mid \neg \phi \mid \phi_1 \land \phi_2 \mid \phi_1 \lor \phi_2 \mid F\phi \mid G\phi \mid \phi_1 U \phi_2\)
- temporal operators

\[
\begin{align*}
    t \models F\phi & \iff \exists t' \geq t \ t' \models \phi \\
    t \models G\phi & \iff \forall t' \geq t \ t' \models \phi \\
    t \models \phi_1 U \phi_2 & \iff \exists t' \geq t \ t' \models \phi_2 \land \forall t'' \geq t \land t'' < t' \ t'' \models \phi_1
\end{align*}
\]

Advantages

- predicate expressiveness
- easy to use
- efficient on a single trace

Example

- alarm \(\rightarrow\) react
- safety stop: \(G(\text{obstacle} \Rightarrow F(\text{stopped}))\)
TPTL formula

- LTL + clock constraints
- LTL syntax: $\phi := LTL \mid x.\phi \mid x \leq \ldots$
- add a context $C$ to save clocks
- clock value is a difference between two event dates
- can compare clock

$$C, t \models x.\phi \quad \text{iff} \quad C :: \{x \rightarrow t\}, t \models \phi$$

$$C :: \{x \rightarrow t'\}, t \models x \ldots : (t - t')\ldots$$

Advantages

- as efficient as LTL (finite number of finite traces)
- expressiveness

Example

- alarm $\rightarrow$ react before a deadline
- safety stop: $G(\text{obstacle} \Rightarrow x.F (\text{stopped} \land (x \leq 200\text{ms})))$
Event Context

- add a context $\mathcal{E}$ to save events
- add $\circ$ operator to save the current event if the formula is satisfied
- formula evaluation gives two information: valuation and $\mathcal{E}$
- add "all" operators collect multiple event contexts

Advantages

- as efficient as TPTL
- TPTL semantic

Example

- safety stop: $\forall (\text{obstacle} \circ o \land x.F (\text{stopped} \circ s))$
- result a set of $\mathcal{E}$ containing the obstacle detection event and the stopped event
Robot
  - P3DX robot
  - sensors: xtion, laser scanner (sonar)
  - ROS communication & RViz

Mission
  - defined as skills
  - create a map (Mauve gmapping imp. with resources)
  - detect and "analyse" objects
  - guidance, control, . . .
  - safety with emergency stop (with different modes)
  - . . .
Conclusion

Current works

- add temporal logic patterns
- continuous time logic
- skills (preti net)
- architecture (re)configuration with monitors ans skills
- efficient and optimal pwcet
- …

https://mauve.gitlab.io