Periodic state-machine aware real-time analysis

Nicolas Gobillot, David Doose, Charles Lesire, Luca Santinelli
firstname.lastname@onera.fr

ONERA
THE FRENCH AEROSPACE LAB

retour sur innovation

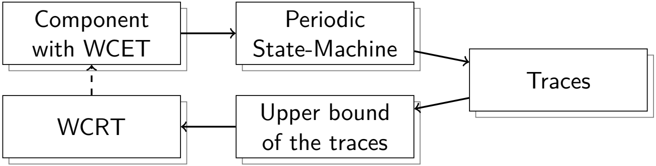
Introduction

- More and more computer-based systems are used (aircraft control, medical assistants, power-plant management . . .)
- It is therefore necessary to prove the safety of these systems. To ensure the safety of a system it is necessary to prove its temporal behavior.
- Software structure complexity have increased to cope with software requirements[1], particularly in the robotic domain often using a component model executed through a middleware.
- We propose a method to analyze component-based software architectures containing state-machines by precisely computing the WCRT of the components.

[1] Martin Stigge et al. "The Digraph Real-Time Task Model". In: *RTAS*. 2011.

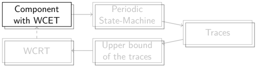
Introduction

Overall analysis process:



- Component definition with their worst case execution times
- Periodic State-Machine extraction from the components
- Traces computation
- Traces upper bound computation
- Worst Case Response Time analysis

Component

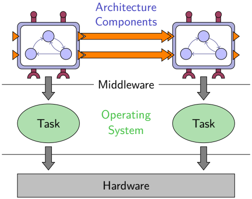


Definition: A component is a software device carrying an elementary function.

Structure: It has an interface to communicate with other components and a behavior modeled as a state-machine.

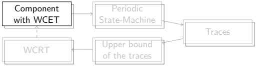
Task model:

- Components are mapped onto operating system tasks through a *middleware*.
- The resulting tasks are defined by the tuple *period* (T_i), *priority* (P_i), *deadline* (D_i), *state-machine*



Component

State-machine model:



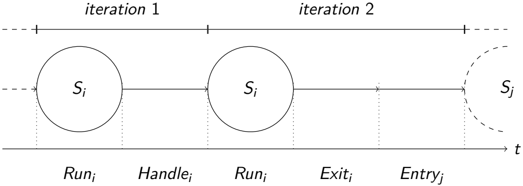
A state-machine is a set S of n states and a set E of m transitions:

$$S = \{s_1, \ldots, s_n\} \quad \text{and} \quad |S| = n \tag{1}$$

$$E = \{e_1, \ldots, e_m\} \quad \text{and} \quad |E| = m \tag{2}$$

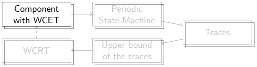
State-machine's structure:

Each state s_i contains up to four time consuming functions: $entry_i$, run_i, $handle_i$ and $exit_i$. There are two possible execution cycle per iteration for a task:



Component

Hook example:




```
run = {
  read(velocity, cmd);
  pos = compute_position(pos, cmd, inertia, (period/1000.0));
  write(position, pos);
}
```

Time consumption:

The WCET estimation is done on codels, which are executed in the state-machine's hooks along with communication functions.

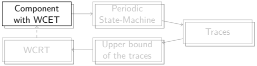
WCET computation:

We have used two methods: static analysis with the Otawa[2] tool and measurement based probabilistic analysis[3]

[2] Christine Rochange and Pascal Sainrat. "OTAWA: An Open Toolbox for Adaptive WCET Analysis". In: *IFIP Workshop (SEUS)*. 2010.

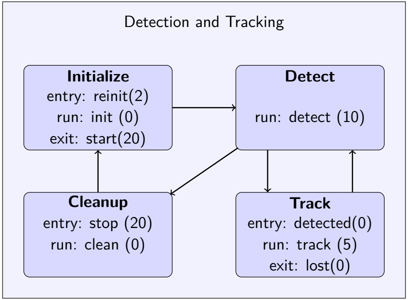
[3] Liliana Cucu-Grosjean et al. "Measurement-Based Probabilistic Timing Analysis for Multi-path Programs". In: *ECRTS 2012*.

Example

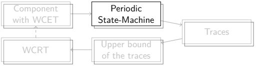


Visual detection and tracking:

This is an example of a detection and tracking algorithm embedded into a state-machine with four states.



Periodic State-Machine



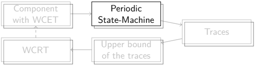
Definition: A Periodic State-Machine (PSM) is a periodically executed state-machine: it fires a transition at every period. A PSM is defined as a set of states S, which are the same states than the original state-machine, and a set of transitions Σ:

$$\Sigma = E \cup \{s \rightarrow s \,|\, s \in S\} \tag{3}$$

Construction: The PSM abstracts the state-machine implementation such as the *entry*, *run*, *handle* and *exit* functions. It also abstracts the computational times into its transitions with a timing function δ:

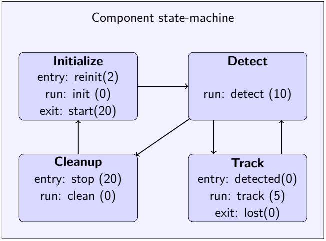
$$\forall \sigma \in \Sigma, s_i \xrightarrow{\sigma} s_j,\ \delta(\sigma) = \begin{cases} s_i \neq s_j : & wcet(run_i) + wcet(exit_i) + wcet(entry_j) \\ s_i = s_j : & wcet(run_i) + wcet(handle_i) \end{cases} \tag{4}$$

Example

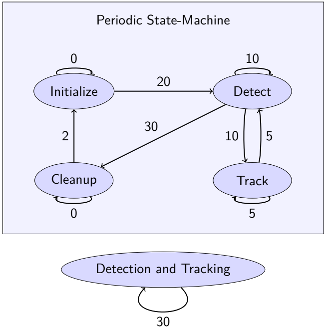


Detection and tracking PSM:

The detection and tracking component set as a Periodic State-Machine.

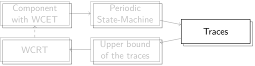


$\Longrightarrow$



Traces

Definition: A trace $\mathcal{T}$ is defined as any sequence of transitions from the PSM:



$$\mathcal{T} = \langle \sigma_1, \ldots, \sigma_N \rangle \tag{5}$$

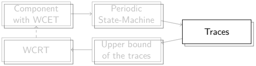
Feasible trace: A feasible trace is a defined as a trace in which the arrival state of a transition is the starting state of the next one:

$$\begin{aligned} \phi(\langle \sigma_1, \sigma_2 \rangle) &\equiv to(\sigma_1) = from(\sigma_2) \\ \phi(\langle \mathcal{T}, \sigma \rangle) &\equiv \phi(\mathcal{T}) \wedge to(\mathcal{T}[|\mathcal{T}|]) = from(\sigma) \end{aligned} \tag{6}$$

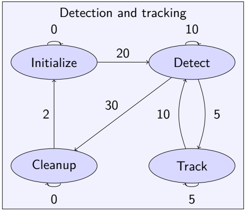
Request function: The traces are used to compute the *(cumulative) request function rbf* of the transition sequence:

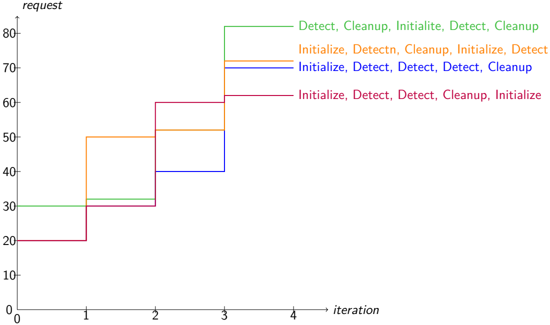
$$rbf\,(\mathcal{T}, t) = \sum_{i=1}^{|\mathcal{T}|} \left\lfloor \frac{t}{T} \right\rfloor \delta\,(\mathcal{T}\,[i]) \tag{7}$$

Example



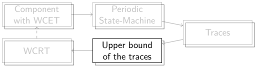
Some traces from the detection and tracking component:





Upper bound of the traces

A PSM have many different possible executions. In order to compute the request function of the PSM, we have to define a trace set.

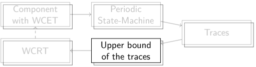


Definition of a trace set:

A trace set is defined as a set of all the feasible traces of a PSM. It is built recursively from all the possible transitions of the PSM:

$$\begin{aligned} \mathcal{U}^1 &= \{\langle\sigma\rangle \mid \sigma \in \Sigma\} \\ \mathcal{U}^{n+1} &= \bigcup_{\mathcal{T}\in\mathcal{U}^n} \mathit{next}\,(\mathcal{T}) \\ &= \{\mathcal{T} \mid \mathcal{T} = \langle\mathcal{T}',\sigma\rangle \wedge \sigma \in \Sigma \wedge \mathcal{T}' \in \mathcal{U}^n \\ &\qquad \wedge \langle\mathcal{T}',\sigma\rangle \in \mathit{next}\,(\mathcal{T}')\} \end{aligned} \tag{8}$$

Upper bound of the traces

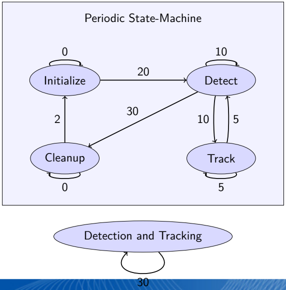


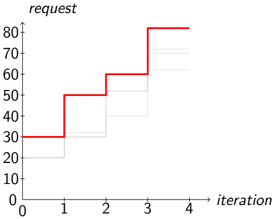
Definition of the upper bound of the traces:

The upper bound of the traces $\mathcal{T}^+$ is defined as a trace and bounds all the possible feasible traces of the PSM:

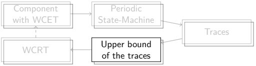
$$\mathcal{T}^+ : \forall n, \forall \mathcal{T} \in \mathcal{U}^n \mid \mathcal{T}^+ \geq \mathcal{T} \qquad (9)$$

It is used to provide the Worst Case Execution Time of the component by maximizing its PSM request function.





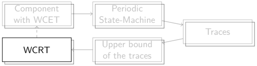
Upper bound of the traces



Upper bound computation algorithm:

- Optimized iterative algorithm taking into account traces included in others to reduce the traces set size.
- Iterative computations stopped when the request function of the upper bound traces hits the biggest deadline.
- The algorithm starts with every transition: the components are not synchronized and can start their execution in any state.

Worst Case Response Time



Definition: The Worst Case Response Time represents the worst time between the beginning and the end of the execution of a task, including the possible preemptions.

WCRT computation from the upper bound traces: In order to exploit the precision brought by the PSM analysis, we use an adapted version of the usual recursive procedure:

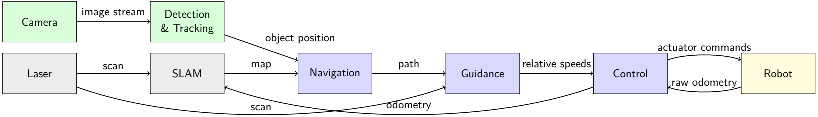
$$\mathcal{R}_i^0 = \mathcal{T}_i^+(0)$$

$$\mathcal{R}_i^{n+1} = \sum_{j \leq hp(i)} \mathcal{T}_j^+(\mathcal{R}_i^n) + \mathcal{T}_i^+(0) \qquad (10)$$

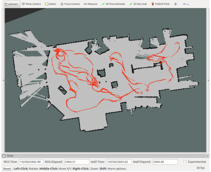
with $hp(i)$ the higher priority task's instance.

Schedulability: The system is schedulable iff the WCRT $\mathcal{R}$ of the components are lesser or equal to their deadlines.

Experiment



- Architecture core: robot's guidance functions
- The robot's driver is added
- So as a mapping component and a laser scanner
- And a detection and tracking algorithm from a video stream



Experiment

Analysis result (*Detection and Tracking* component)

iteration	0	1	2	3	4
*track**	5	10	15	20	25
*detect**	10	20	30	40	50
*start, detect, detected, track**	20	30	40	45	50
*(start, stop, reinit)**	20	50	52	72	102
*(stop, reinit, start)**	30	32	52	82	84
*(reinit, start, stop)**	2	22	52	54	74
⋮					
PSM based analysis	30	50	52	82	102
analysis without PSM	30	60	90	120	150
precision gain (%)	0	17	42	32	32

Experiment

Analysis results (whole architecture)

component	prio.	WCET	WCRT*	WCRT+	period
Robot	8	16	16	16	100
Control	7	3	19	19	100
Guidance	6	12	31	31	100
Laser	5	22	53	53	150
SLAM	4	30	83	83	150
Camera	3	10	93	93	250
Det.&Track.	2	30	237	237	250
Navigation	1	30	**307** (338)	**297**	300

WCRT*: analysis without PSM
WCRT+: analysis with PSM

Conclusion

This analysis provides precise WCRT estimation using the state-machines contained in some components.

Work in progress:

- Analyze the middleware's protocols time consumption. (done)
- Adapt the analysis to multicore or multiprocessor hardware architectures. (partially done)
- Extract probabilistic timings from traces of the components to compute probabilistic execution times. (ongoing)
- Use different scheduling analyses, such as EDF.