Response-Time Analysis for Task Chains in Communicating Threads with pyCPA

Johannes Schlatow, Jonas Peeck and Rolf Ernst
Background

Motivation

- growing variety and complexity (e.g. automotive domain)
- component-based design (e.g. AUTOSAR)
- in-field updateability

→ Automated in-field integration of component-based systems under timing constraints.
Background

Motivation

- growing variety and complexity (e.g. automotive domain)
- component-based design (e.g. AUTOSAR)
- in-field updateability

→ Automated in-field integration of component-based systems under timing constraints.

Desired functionality is implemented by a composition of interacting software components:

- represented by communicating threads *(operating-system view)*
- represented by task chains *(timing view)*
Demo scenario – Automotive use case

Two ADAS functions implemented by multiple software components with RPC interfaces (client-server).

- Park assist (P), Trajectory calculation (T), Object recognition (O1)
- Lane detection (L), Object recognition & Object masking (O2), Steering (S)

System configuration

- Composition of software components
- Thread-priority assignment

Response-time analysis (RTA) [1] is performed to find a priority assignment that satisfies the given end-to-end timing constraints.

[1] Analysis details see RTAS’16 paper (presentation on Thursday).
Demo scenario – Design-space exploration for in-field reconfiguration

- contract specification
- lab-centric or in-field tracing

in-field update

- priority assignment
- model extraction

no

RTA w/ pyCPA

valid?

yes

apply
design-space exploration

no
Demo scenario – Design-space exploration for in-field reconfiguration

- contract specification
- lab-centric or in-field tracing

→ in-field update

<table>
<thead>
<tr>
<th>priority assignment</th>
</tr>
</thead>
</table>

→ model extraction

- no

→ RTA w/ pyCPA

- valid?

→ design-space exploration

- yes

→ apply

- sync
- MAST
- async