Towards a declarative modeling and execution framework for real-time systems

Sebastian Altmeyer, Nicolas Navet



1.DPRTCPS, San Antonio 2015

More than 35 years of Research in Real-Time Systems

with some very impressive results and applications ...



More than 35 years of Research in Real-Time Systems

with some very impressive results and applications ...



... but timing is still a mere side-product.

Design of an real-time/cyber-physical system:

- timing behaviour happens (side product of the functional behaviour)
- timing verification after system integration
- system designer must be aware of all scheduling details few abstractions provided
- even dedicated design tools avoid timing specification (Matlab, SCADE/Esteral, Ascet)

Design of an real-time/cyber-physical system:

- timing behaviour happens (side product of the functional behaviour)
- timing verification after system integration
- system designer must be aware of all scheduling details few abstractions provided
- even dedicated design tools avoid timing specification (Matlab, SCADE/Esteral, Ascet)

 \Rightarrow

timing is not treated as a first-class citizen

Principles of our declarative framework:

- Designer only declares the desired timing behaviour
- Show only what is needed to the designer, hide the rest.
- Simplicity is key.

Premise:

Better abstraction of a system's timing behaviour needed!

Example: Specifying timing behaviour

State-of-the art: Plenty of design choices and details.

- Do we allow pre-emption?
- Static or dynamic scheduling?
- Which scheduling policy?
- Dynamic or static priorities?
- How to assign priorities?



Concentrates on how to realize the timing behaviour

Example: Specifying timing behaviour

Our vision: Only declare timing correctness.

4 simple types of constraints*:

Execution frequency: process τ_a executes every [x : y] seconds. Conditional execution: process τ_a executes (i) if its period has elapsed and (ii) if condition C evaluates to true. Relative deadlines: process τ_a must complete within y seconds. Temporal dependencies: process τ_a must execute after process τ_b has finished. *(Complete? Probably not, but sufficient to start with.)

Concentrates on what instead of how, environment does the rest.

Functional Model Timing Model

+ User View 4

Designer writes the functional and timing model



Designer writes the functional and timing model ... in the way it shall behave on the system. + User View +



- Designer writes the functional and timing model ... in the way it shall behave on the system.
- Hide as many details as possible



- Designer writes the functional and timing model ... in the way it shall behave on the system.
- Hide as many details as possible ... but show how it will behave.







Runtime Environment

- uniprocessor system
- a system-wide clock
- time-triggered task release + FIFO queues
- prototype environment for Raspberry Pi



FIFO Scheduling: Why?

- easy to implement
- non-pre-mptive policy
- unique event-order
- ensures equivalence between
 - (i) runtime behaviour
 - (ii) simulation
- (work-conserving)
- resilient to overload conditions
- but not as performant as EDF/RM

FIFO Scheduling: How?

- *n* processes (tasks) $\{\tau_1, \ldots, \tau_n\}$
- for each process τ_i : (O_i, C_i, T_i, D_i) ,
 - Oi offset
 - C_i execution time bound
 - T_i period (strictly periodic)
 - D_i relative deadline

Scheduler Synthesis

(i) Period Selection: Try:

- 1. Best Performance
- 2. Minimal Hyperperiod
- 3. Lowest Utilization

(ii) Offset Optimization:

distribute the workload and avoid load peaks







The complete picture, partly integrated

Design environment (Cyber-Physical Action Language CPAL)¹



The complete picture, partly integrated

Design environment (Cyber-Physical Action Language CPAL)¹



The complete picture, partly integrated

Design environment (Cyber-Physical Action Language CPAL)¹



Runtime environment:



Conclusions

Is it possible to just declare what what correct timing behaviour means, instead of defining how it is realized?



Declarative modeling and execution framework

- hide as much as possible from the designer
- automatize what's possible
- simplicity and usability in mind

Questions?