

Towards a Reactive Model for the Power-Aware Control of an Embedded Platform

(Work in progress. . .)

Nicolas BERTHIER



October 23, 2009

Outline

- Context
 - Target platforms & applications
 - Problems
- Current solutions
 - Ad-Hoc solutions
 - Dedicated Operating Systems
 - Conclusion
- Approach
 - Preliminary remarks
 - Principles
 - Pros & Cons
- Conclusion

Outline

- Context
 - Target platforms & applications
 - Problems
- Current solutions
 - Ad-Hoc solutions
 - Dedicated Operating Systems
 - Conclusion
- Approach
 - Preliminary remarks
 - Principles
 - Pros & Cons
- Conclusion

Target platforms

- ▶ μ -Controller
 - ▶ Sleep/low-power modes and fast wake-up time
 - ▶ Possibly DV(F)S capable
- ▶ Radio transceiver
- ▶ Real-time clock
- ▶ Sensors (Temperature, Humidity...)
- ▶ Battery-operated
 - ▶ Possibly solar-reloadable
- ▶ ...

Target platforms (contd.)

Energy vs. Power

- ▶ Energy-awareness: control *average* current consumption
 - ▶ related to discharge profile
- ▶ Power-awareness: control *maximum* current consumption
Highly related to:
 - ▶ capacity
 - ▶ quality (\rightsquigarrow cost)

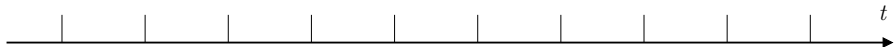
Target applications

- ▶ Data acquisition
 - ▶ Periodically
 - ▶ On demand
- ▶ Triggering an alarm on certain events

Target applications

- ▶ Data acquisition
 - ▶ Periodically
 - ▶ On demand
- ▶ Triggering an alarm on certain events

⇒ In most cases:

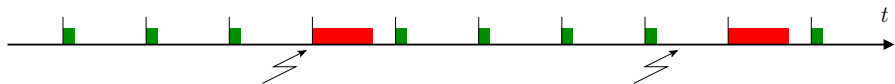


- ▶ Periodic sensing of environment and/or listening radio channel
- ▶ Should sleep most of the time

Target applications

- ▶ Data acquisition
 - ▶ Periodically
 - ▶ On demand
- ▶ Triggering an alarm on certain events

⇒ In most cases:



- ▶ Periodic sensing of environment and/or listening radio channel
- ▶ Should sleep most of the time

Problems: distribution

- ▶ “Chaotic” global behavior, dynamism
- ▶ Debugging
- ▶ Shelf life
- ▶ Unreliable communications
- ▶ Security
- ▶ ...

worse and worse. . .

Problems: “embeddedness”

- ▶ Cyber-physical device: closed loop (reliable simulation...?)
- ▶ Various constraints
 - ▶ low speed
 - ▶ very small memory
 - ▶ ...
- ▶ Various peripherals
- ▶ ...

Outline

- Context
 - Target platforms & applications
 - Problems
- Current solutions
 - Ad-Hoc solutions
 - Dedicated Operating Systems
 - Conclusion
- Approach
 - Preliminary remarks
 - Principles
 - Pros & Cons
- Conclusion

Current solutions: ad-hoc solutions

Manual implementation

- ▶ Method?
- ▶ Toolchain?
- ▶ Power management?
- ▶ ...
- ▶ Reliability?

Current solutions: dedicated Operating Systems

TINYOS

- ▶ Event-driven paradigm
 - ▶ All implemented in NESC
 - ▶ restricted / extended C
 - ▶ fully component-based approach
 - ▶ Compiled together with the applications
 - ▶ Locks \rightsquigarrow deadlocks
 - ▶ Power management: manual & decentralized
 - ▶ Available on many platforms
- \rightsquigarrow tens of components needed for simple applications. . .

Current solutions: dedicated Operating Systems (contd.)

MANTIS

- ▶ Classical programming model
- ▶ Supports long-running computations
- ▶ Power management by hand

- ~> More efficient than TINYOS when:
- ▶ long idle periods
 - ▶ timely processing

Current solutions: dedicated Operating Systems (contd.)

CONTIKI

- ▶ *Exokernel*
 - ▶ tries to reduce provided abstractions
 - ▶ instead: use libraries
- ▶ *Proto-threads*
 - ▶ event-based, stack-less
 - ▶ optional preemptive multi-threading implemented as a library
- ▶ Inter-process communication: *event posting*
- ▶ Power-management by hand (only helped by exposing the size of the event queue!)

Context: conclusion

⇒ *No mechanism allowing global management of the platform!*

Outline

- Context
 - Target platforms & applications
 - Problems
- Current solutions
 - Ad-Hoc solutions
 - Dedicated Operating Systems
 - Conclusion
- Approach
 - Preliminary remarks
 - Principles
 - Pros & Cons
- Conclusion

Approach: preliminary remarks

- ▶ Driver \approx Automaton — In fact, a *Transducer*:
 - ▶ Inputs = software requests & hardware interrupts / data
 - ▶ Outputs = software responses & hardware requests
 - ▶ Reflects the state of the associated device

Approach: preliminary remarks

- ▶ Driver \approx Automaton — In fact, a *Transducer*:
 - ▶ Inputs = software requests & hardware interrupts / data
 - ▶ Outputs = software responses & hardware requests
 - ▶ Reflects the state of the associated device

- ▶ Ensuring properties of the *whole platform*
 - \Rightarrow controlling the drivers behavior

Approach: preliminary remarks

- ▶ Driver \approx Automaton — In fact, a *Transducer*:
 - ▶ Inputs = software requests & hardware interrupts / data
 - ▶ Outputs = software responses & hardware requests
 - ▶ Reflects the state of the associated device
- ▶ Ensuring properties of the *whole platform*
 \Rightarrow controlling the drivers behavior
- ▶ *Controller Synthesis Problem*:

$$D_1 \quad D_2 \quad \cdots \quad D_n$$

Approach: preliminary remarks

- ▶ Driver \approx Automaton — In fact, a *Transducer*:
 - ▶ Inputs = software requests & hardware interrupts / data
 - ▶ Outputs = software responses & hardware requests
 - ▶ Reflects the state of the associated device
- ▶ Ensuring properties of the *whole platform*
 \Rightarrow controlling the drivers behavior
- ▶ *Controller Synthesis Problem*:

$$D_1 \parallel D_2 \parallel \cdots \parallel D_n$$

Approach: preliminary remarks

- ▶ Driver \approx Automaton — In fact, a *Transducer*:
 - ▶ Inputs = software requests & hardware interrupts / data
 - ▶ Outputs = software responses & hardware requests
 - ▶ Reflects the state of the associated device
- ▶ Ensuring properties of the *whole platform*
 \Rightarrow controlling the drivers behavior
- ▶ *Controller Synthesis Problem*:

$$D_1 \parallel D_2 \parallel \cdots \parallel D_n \parallel C$$

Approach: preliminary remarks

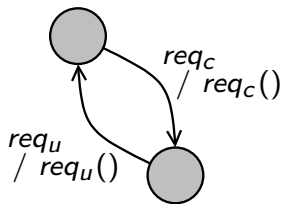
- ▶ Driver \approx Automaton — In fact, a *Transducer*:
 - ▶ Inputs = software requests & hardware interrupts / data
 - ▶ Outputs = software responses & hardware requests
 - ▶ Reflects the state of the associated device
- ▶ Ensuring properties of the *whole platform*
 \Rightarrow controlling the drivers behavior
- ▶ *Controller Synthesis Problem*:

$$D_1 \parallel D_2 \parallel \cdots \parallel D_n \parallel C \models \phi$$

Approach: preliminary remarks (contd.)

Controllable Driver

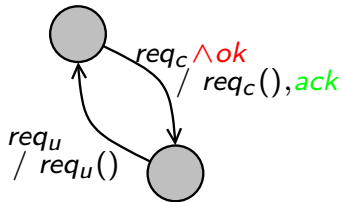
- ▶ Inputs: $\{req_c, req_u\}$
- ▶ Outputs: $\{req_c(), req_u()\}$



Approach: preliminary remarks (contd.)

Controllable Driver

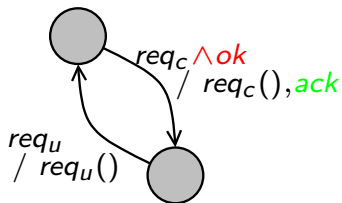
- ▶ Inputs: $\{req_c, req_u, ok\}$
- ▶ Outputs: $\{req_c(), req_u(), ack\}$



Approach: preliminary remarks (contd.)

Controllable Driver

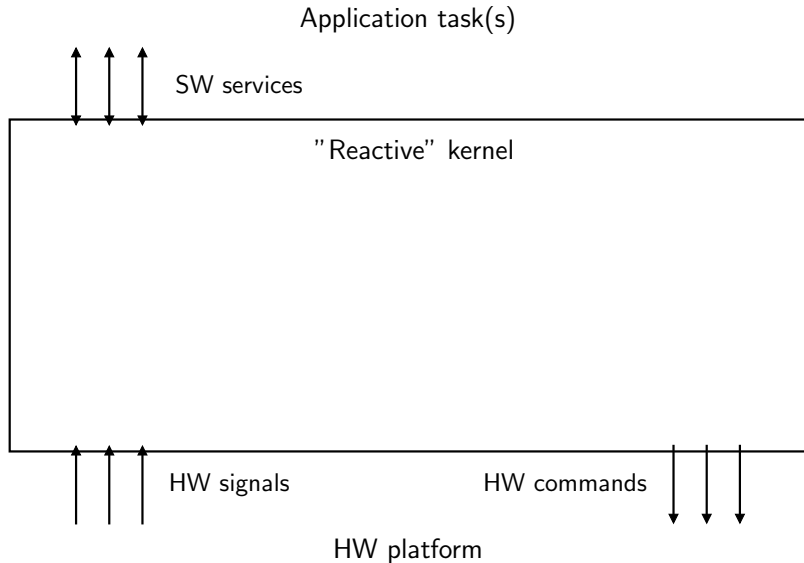
- ▶ Inputs: $\{req_c, req_u, ok\}$
- ▶ Outputs: $\{req_c(), req_u(), ack\}$



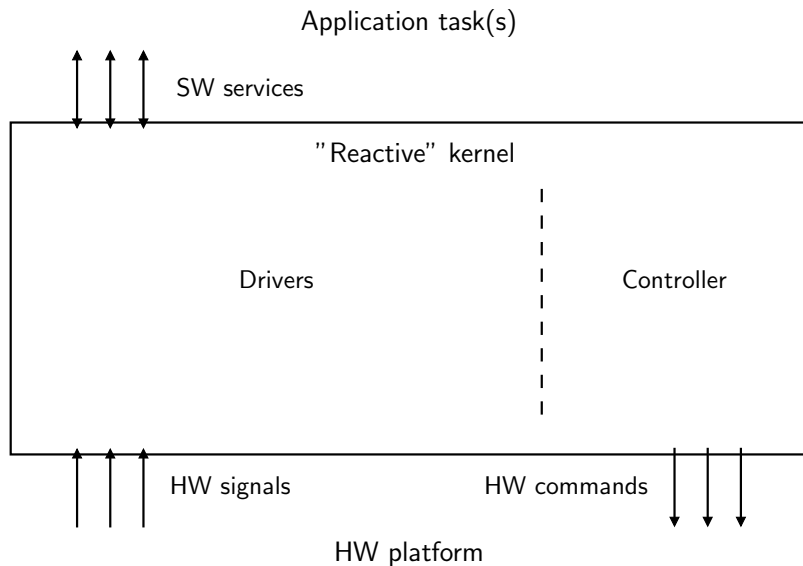
ok: Validation from controller

ack: Refused request handling

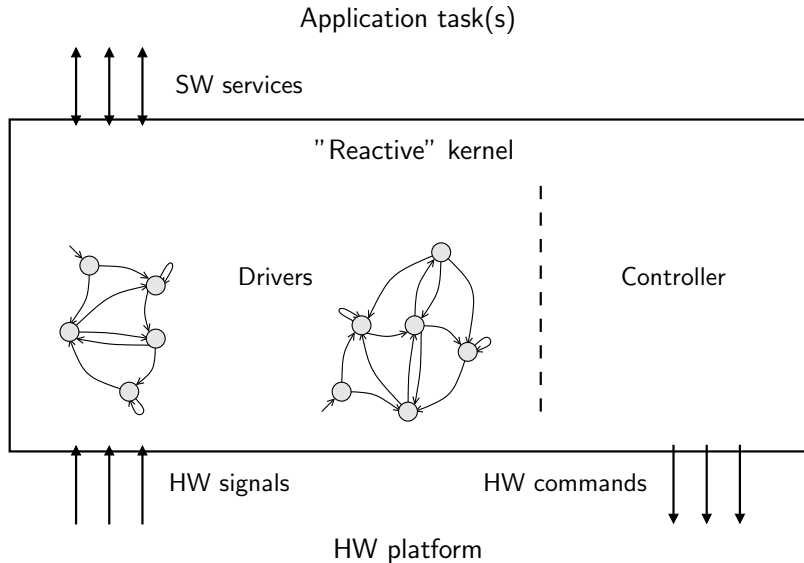
Approach: overview



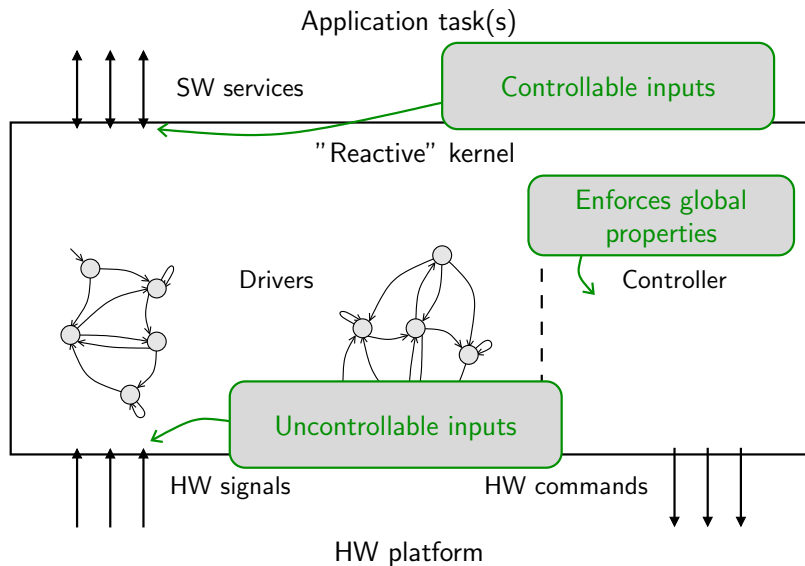
Approach: overview



Approach: overview



Approach: overview



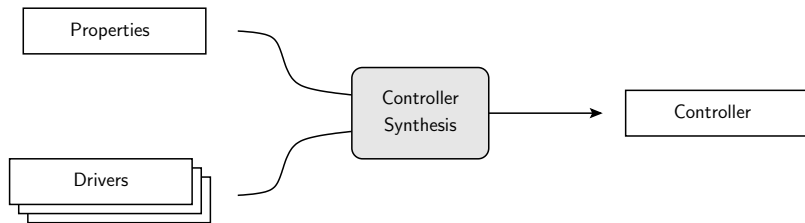
Approach: framework



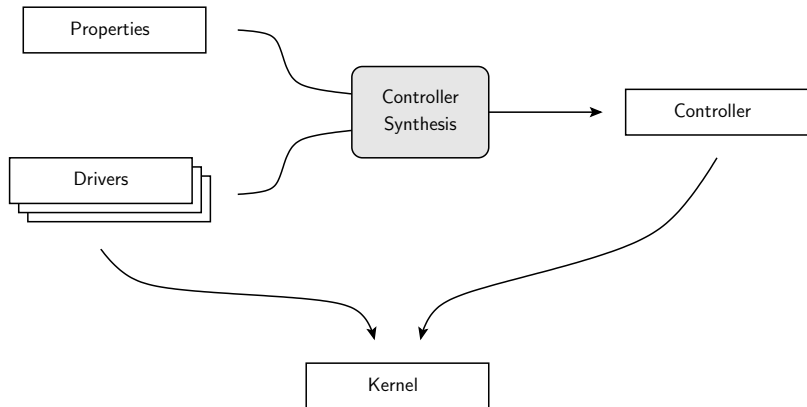
Properties

Drivers

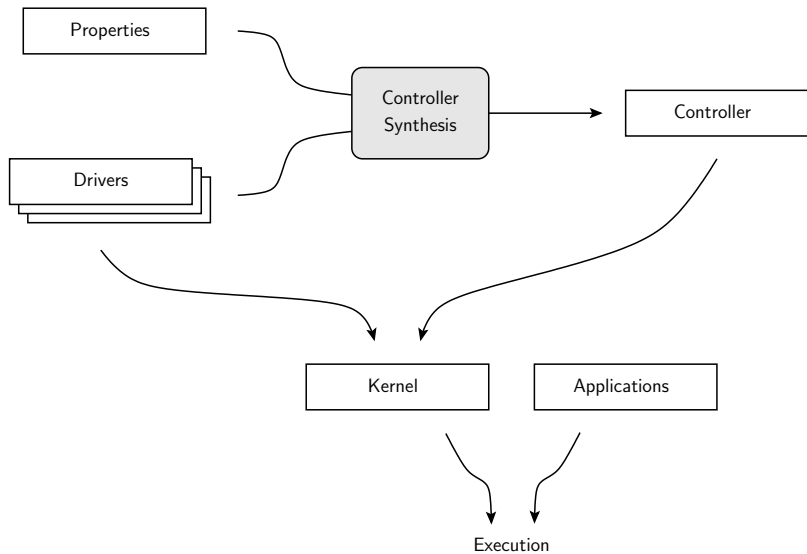
Approach: framework



Approach: framework



Approach: framework



Approach: cons

Cons

- ▶ Uncontrollable combinations of platforms & properties
- ▶ Very platform-specific properties
- ▶ It is still possible to write energy-hungry applications
... not surprisingly!

Approach: pros

Pros

- ▶ Power & Energy -aware “by construction”
depending on the properties. . .
- ▶ Unchanged application layer
- ▶ Concurrency handling by the controller:
 \leadsto No locks \Rightarrow No deadlocks
- ▶ Supports long-running application tasks
 - ▶ Classical programming model

Outline

- Context
 - Target platforms & applications
 - Problems
- Current solutions
 - Ad-Hoc solutions
 - Dedicated Operating Systems
 - Conclusion
- Approach
 - Preliminary remarks
 - Principles
 - Pros & Cons
- Conclusion

Conclusion

- ▶ Lack of global control capabilities in current tool-sets
 - ▶ Most often too hard to implement
- ▶ Global control of the platform
 - ▶ Concurrency handling
 - ▶ Power & Energy management
- ▶ Towards a “Real-Energy Programming” paradigm
- ▶ Proof-of-concept implementation (in progress)
 - ▶ Realistic application

Thank you

Questions ?