SDL Implementations for Wireless Sensor Networks

Incorporation of PragmaDev's RTDS into the Deterministic Protocol Stack BiPS

Tobias Braun, Dennis Christmann, Reinhard Gotzhein, Alexander Mater
{tbraun, christma, gotzhein, a_mater09}@cs.uni-kl.de
http://vs.cs.uni-kl.de

Networked Systems Group

Technische Universität
Kaiserslautern

2014-09-30 – SAM 2014, Valencia, Spain

Outline

1. Introduction
2. BiPS
3. Incorporation of SDL into BiPS
   - Scheduling the SDL System
   - Interfacing the SDL Environment
4. Evaluation
5. Conclusions

Motivation

- challenges of modern (wireless) sensor systems
  - efficiency
    - energy
    - storage
  - predictability
    - communication: Transfer rates, delays, . . .
    - software implementations: Run-time, waiting times
  - complexity
  - reuse
  - determinism
  - . . .

Motivation

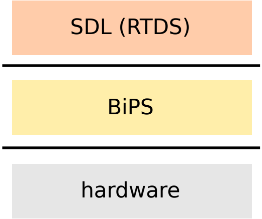
- challenges of modern (wireless) sensor systems
  - efficiency → **manual implementation**
    - energy
    - storage
  - predictability → **manual implementation**
    - communication: Transfer rates, delays, . . .
    - software implementations: Run-time, waiting times
  - complexity → **model-driven implementation**
  - reuse → **model-driven implementation**
  - determinism → **manual implementation**
  - . . .

Motivation

- challenges of modern (wireless) sensor systems
  - efficiency → **manual implementation**
    - energy
    - storage
  - predictability → **manual implementation**
    - communication: Transfer rates, delays, ...
    - software implementations: Run-time, waiting times
  - complexity → **model-driven implementation**
  - reuse → **model-driven implementation**
  - determinism → **manual implementation**
  - ...

objective: Find a trade-off combining the benefits of manual and model-driven implementations

Hybrid Design: Model-driven vs. Hand-written



SDL (RTDS)

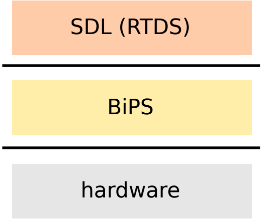
BiPS

hardware

- Specification and Description Language (SDL)
  - language for the specification of distributed systems
  - tool support for model-driven implementations

⇒ *use for applications and higher-layer protocols*

Hybrid Design: Model-driven vs. Hand-written

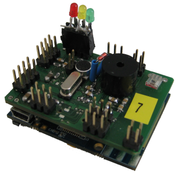

SDL (RTDS)

BiPS

hardware

- Specification and Description Language (SDL)
  - language for the specification of distributed systems
  - tool support for model-driven implementations

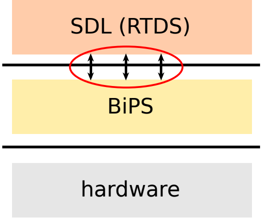
⇒ *use for applications and higher-layer protocols*



- Black burst-integrated Protocol Stack (BiPS)
  - protocol framework for wireless sensor nodes
  - operating system functionalities
  - manual bare implementation for Imote2

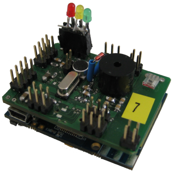
⇒ *use for hardware-related functionality and time-critical (MAC) protocols*

Hybrid Design: Model-driven vs. Hand-written



- Specification and Description Language (SDL)
  - language for the specification of distributed systems
  - tool support for model-driven implementations

⇒ *use for applications and higher-layer protocols*



- Black burst-integrated Protocol Stack (BiPS)
  - protocol framework for wireless sensor nodes
  - operating system functionalities
  - manual bare implementation for Imote2

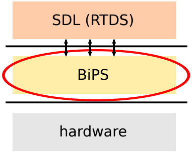
⇒ *use for hardware-related functionality and time-critical (MAC) protocols*

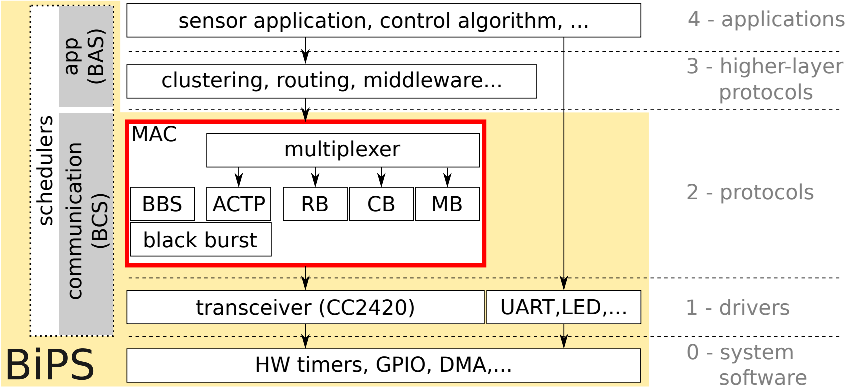
Black burst-integrated Protocol Stack

1. Introduction
2. BiPS
3. Incorporation of SDL into BiPS
   - Scheduling the SDL System
   - Interfacing the SDL Environment
4. Evaluation
5. Conclusions

BiPS – (Deterministic) Protocols

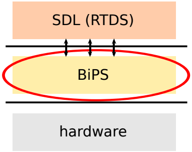
- BBS – Synchronization protocol with bounded offset
- RB – reservation-based MAC (TDMA)
- CB – contention-based MAC (CSMA/CA)
- ...

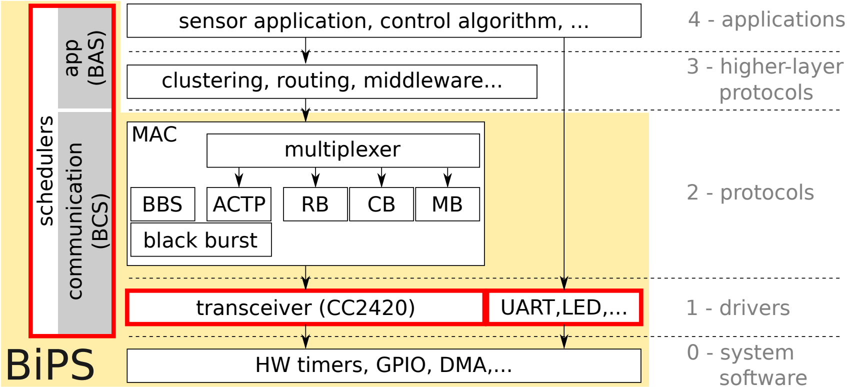




BiPS – Operating System Functionalities

- hardware drivers
- schedulers
  - BiPS Communication Scheduler (BCS)
  - BiPS Application Scheduler (BAS)

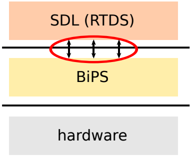




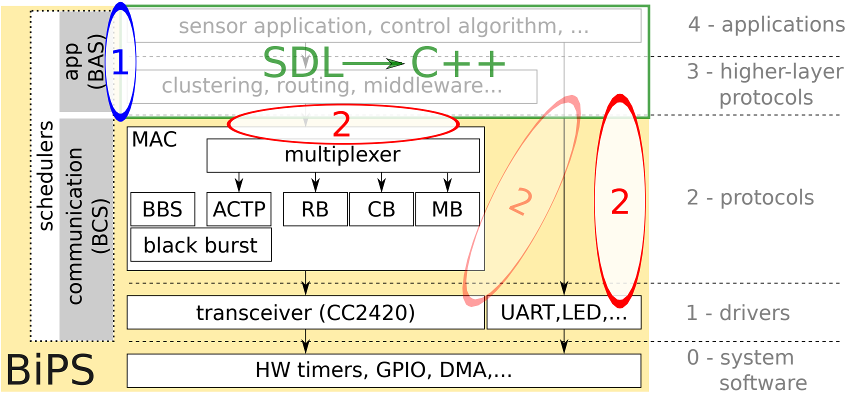
Incorporation of SDL into BiPS

1. Introduction
2. BiPS
3. Incorporation of SDL into BiPS
   - Scheduling the SDL System
   - Interfacing the SDL Environment
4. Evaluation
5. Conclusions

Incorporation of SDL into BiPS – Integration Steps



- integration steps
  1. schedule the SDL system with BAS
  2. interface the SDL environment with BiPS



Incorporation of SDL into BiPS

1. Introduction
2. BiPS
3. Incorporation of SDL into BiPS
   - Scheduling the SDL System
   - Interfacing the SDL Environment
4. Evaluation
5. Conclusions

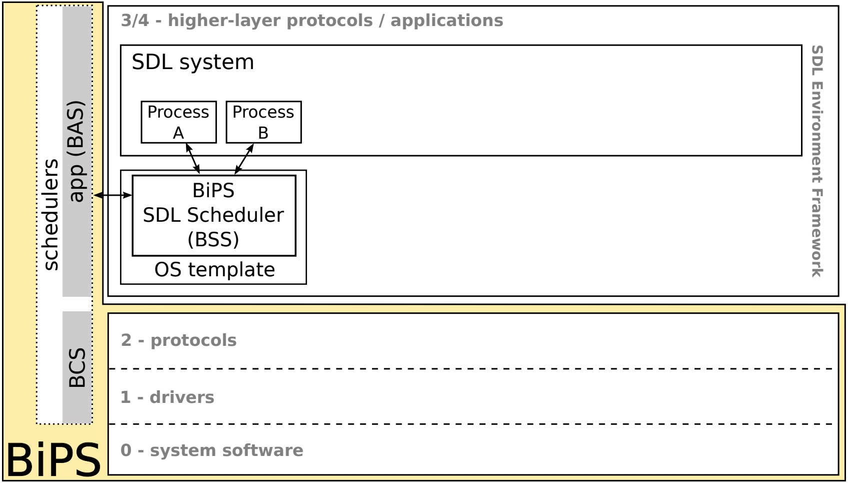
Scheduling the SDL System – Overview

- tasks of an SDL scheduler
  - serialize SDL transition executions
  - deliver SDL signals inter and intra SDL systems
  - manage SDL timers

Scheduling the SDL System – Overview

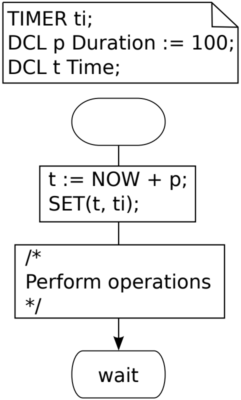
- tasks of an SDL scheduler
  - serialize SDL transition executions
  - deliver SDL signals inter and intra SDL systems
  - manage SDL timers
- integration approach
  - single task scheduling
    → PragmaDev's *rtosless* template
  - adoption of PragmaDev's CPPScheduler for intra-task scheduling
    → **BiPS SDL Scheduler (BSS)**
    - signal-based (FIFO)
    - non-preemptive execution of transitions
  - scheduling of BSS as application of BAS
    - SDL system runs with lower priority than BCS
    - interruptible execution of the SDL system in favor of BiPS protocols

Scheduling the SDL System – BSS in BiPS



Scheduling the SDL System – Comments on BSS

- realization of SDL time (NOW)
    - derivation from hardware clock
    - fine-grained (1 μs)

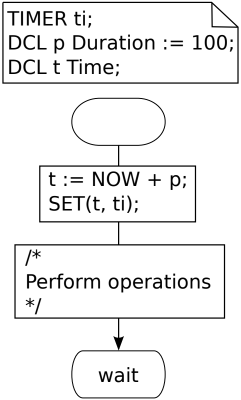

TIMER ti;
DCL p Duration := 100;
DCL t Time;

t := NOW + p;
SET(t, ti);

/*
Perform operations
*/

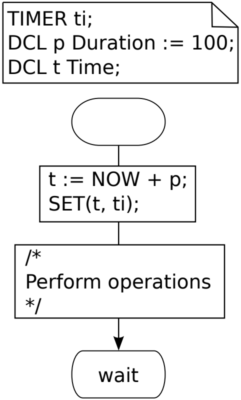
Scheduling the SDL System – Comments on BSS

- realization of SDL time (NOW)
  - derivation from hardware clock
  - fine-grained (1 μs)
- incorporation of SDL timers
  - SDL SET with absolute time values
  - delegation to timer system of BAS
    → setup of hardware timer
  - expiration of timer by hardware interrupt
    → execution of BSS after interrupt mode

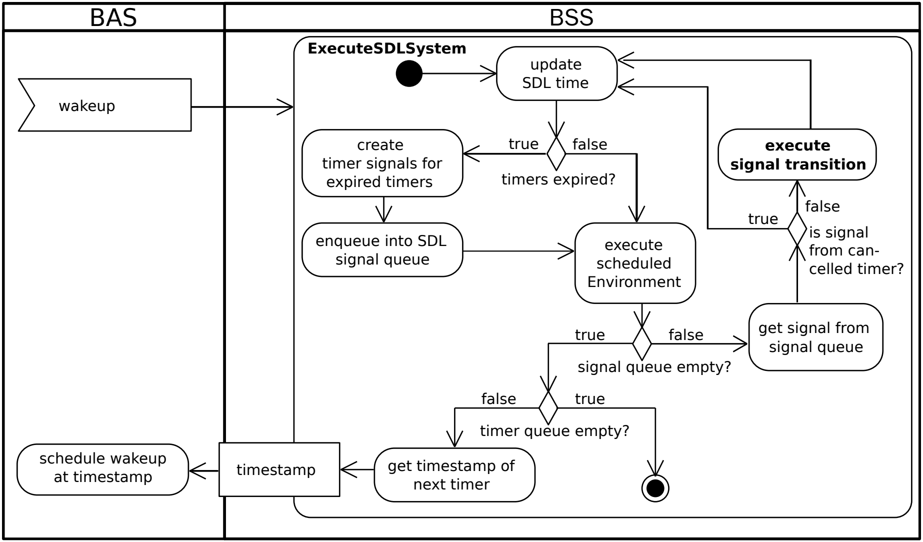


Scheduling the SDL System – Comments on BSS

- realization of SDL time (`NOW`)
    - derivation from hardware clock
    - fine-grained (1 μs)
- incorporation of SDL timers
    - SDL `SET` with absolute time values
    - delegation to timer system of BAS
      → setup of hardware timer
    - expiration of timer by hardware interrupt
      → execution of BSS after interrupt mode
- processing of external events
    - announced by hardware interrupts
    - execution of BSS via BAS after interrupt mode



Scheduling the SDL System – Mode of Operation


BAS
BSS
ExecuteSDLSystem
wakeup
update SDL time
true
false
timers expired?
create timer signals for expired timers
enqueue into SDL signal queue
execute scheduled Environment
execute signal transition
false
true
is signal from can-celled timer?
get signal from signal queue
true
false
signal queue empty?
false
true
timer queue empty?
get timestamp of next timer
timestamp
schedule wakeup at timestamp

Incorporation of SDL into BiPS

1. Introduction
2. BiPS
3. Incorporation of SDL into BiPS
   - Scheduling the SDL System
   - Interfacing the SDL Environment
4. Evaluation
5. Conclusions

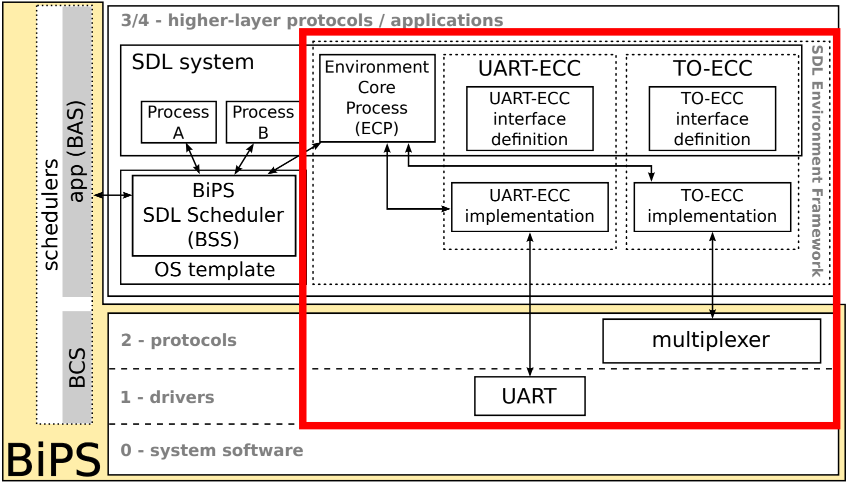
Interfacing the SDL Environment – Overview

- tasks of the SDL environment
  - providing access to hardware peripherals from within SDL systems
  - transfer data to/from peripherals
  - trigger the execution of the system in consequence of external events

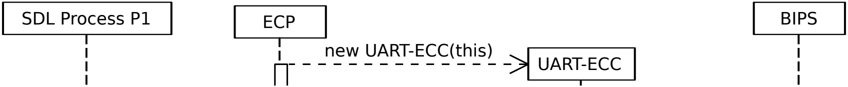
Interfacing the SDL Environment – Overview

- tasks of the SDL environment
  - providing access to hardware peripherals from within SDL systems
  - transfer data to/from peripherals
  - trigger the execution of the system in consequence of external events
- realization as SDL process → *Environment Core Process* (ECP)
  - runs under control of BSS
  - sub-divided into Environment Core Components (ECCs)
    - access to BiPS functionality (drivers, protocols)
    - interaction with SDL system via SDL signals
    - consists of interface definition (SDL package) and implementation (C++)

Interfacing the SDL Environment – Architecture

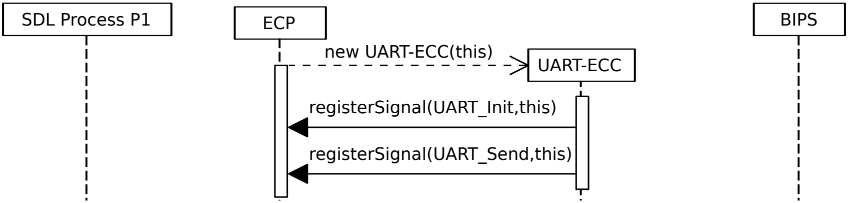


Interfacing the SDL Environment – Initialization of ECCs



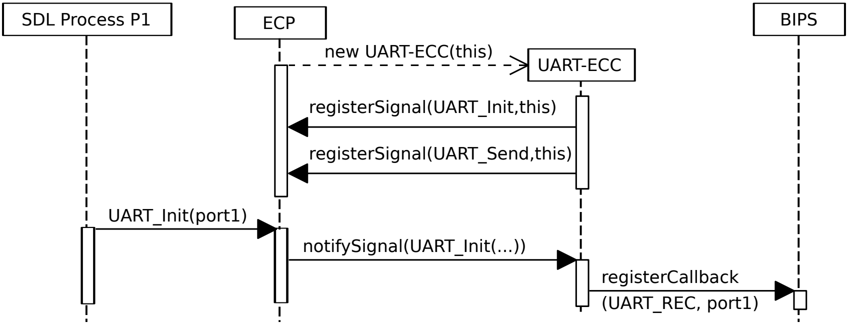
1. ECP creates required ECCs depending on declared SDL signals

Interfacing the SDL Environment – Initialization of ECCs


SDL Process P1
ECP
BIPS
new UART-ECC(this)
UART-ECC
registerSignal(UART_Init,this)
registerSignal(UART_Send,this)

1. ECP creates required ECCs depending on declared SDL signals
2. ECC registers responsible signals at ECP

Interfacing the SDL Environment – Initialization of ECCs



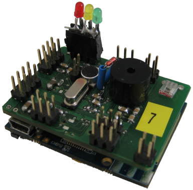
1. ECP creates required ECCs depending on declared SDL signals
2. ECC registers responsible signals at ECP
3. ECP forwards signal to registered ECC

Evaluation

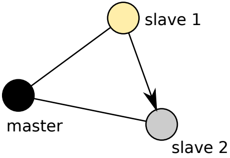
1. Introduction
2. BiPS
3. Incorporation of SDL into BiPS
   - Scheduling the SDL System
   - Interfacing the SDL Environment
4. **Evaluation**
5. Conclusions

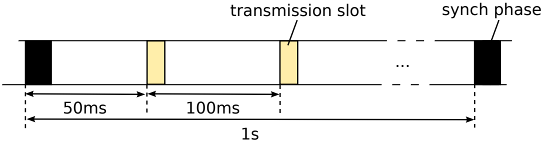
Evaluation – Scenario

- objectives
  1. functional evaluation
  2. quantification of integration's advantage over pure SDL solution



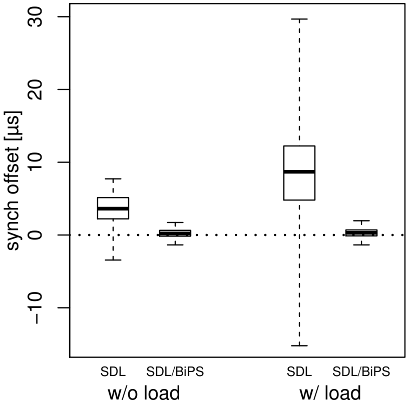
- Scenario





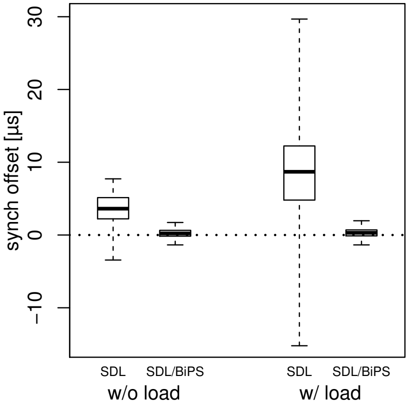
- realizations
  1. **SDL** only (w/o BBS and MAC protocols of BiPS)
  2. full **SDL/BiPS** integration

Evaluation – Results

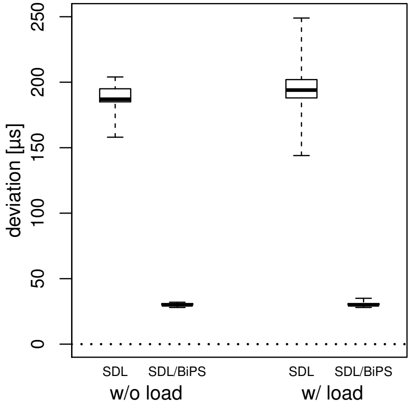


synchronization offset (slaves only)

Evaluation – Results



synchronization offset (slaves only)



temporal deviation of data frames

Conclusions

1 Introduction

2 BiPS

3 Incorporation of SDL into BiPS
- Scheduling the SDL System
- Interfacing the SDL Environment

4 Evaluation

5 Conclusions

Conclusions

- results of incorporating SDL into BiPS
  - BSS: extended SDL scheduler under control of BiPS
  - extendible and modular environment framework

Conclusions

- results of incorporating SDL into BiPS
  - BSS: extended SDL scheduler under control of BiPS
  - extendible and modular environment framework

- lessons learned
  1. hybrid approaches have advantages w.r.t. efficiency and predictability
  2. BiPS is an adequate framework and basis for SDL
  3. RTDS provides a flexible interface for new software platforms

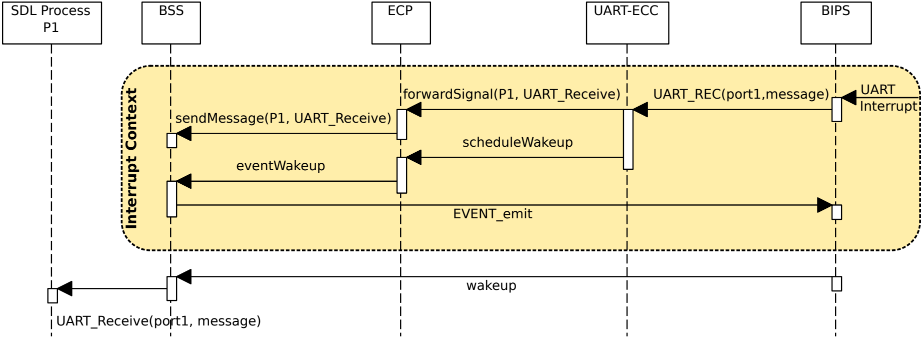
Conclusions

- results of incorporating SDL into BiPS
  - BSS: extended SDL scheduler under control of BiPS
  - extendible and modular environment framework

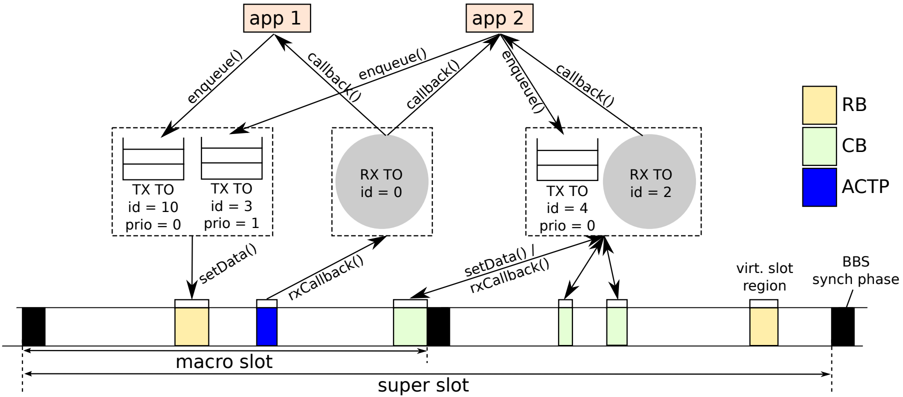
- lessons learned
  1. hybrid approaches have advantages w.r.t. efficiency and predictability
  2. BiPS is an adequate framework and basis for SDL
  3. RTDS provides a flexible interface for new software platforms

- future work
  - more sophisticated scheduling strategies

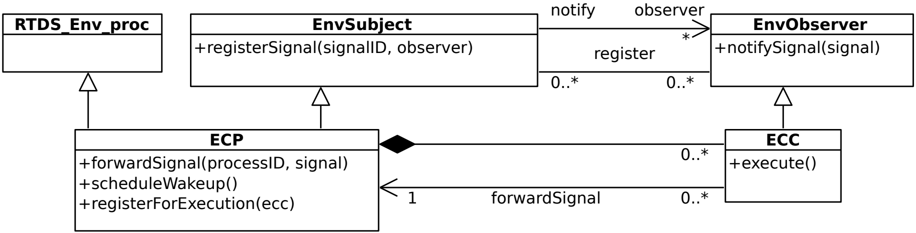
SDL Environment – Signals from External Events



BiPS – Multiplexer with Transmission Opportunities (TOs)



SDL Environment: Class Diagram with Observer Pattern


RTDS_Env_proc
EnvSubject
+registerSignal(signalID, observer)
notify
observer
*
register
0..*
0..*
EnvObserver
+notifySignal(signal)
ECP
+forwardSignal(processID, signal)
+scheduleWakeup()
+registerForExecution(ecc)
0..*
1
forwardSignal
0..*
ECC
+execute()