Component-based real-time systems

Ivica Crnkovic
Mälardalen Högskola
ivca.crnkovic@mdh.se
http://www.idt.mdh.se/~icc

Outline

- Basic principles of real-time (RT) systems
- Component specification and component-based process
- Suitability of component-based approach for RT systems
- RT Components, specification and composition
- Component-based RT systems development process
- (Examples of component-models for embedded and RT systems)

What is a real-time system?

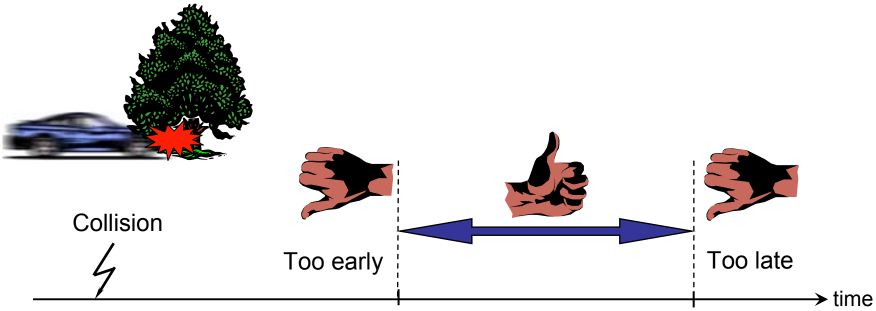
“A real-time system is a system that **reacts upon outside events** and performs a function based on these and **and gives a response within a certain time**. Correctness of the function does not only depend on correctness of the result, but also the **timeliness** of it”.

The controlled process dictates the time scale (some processes have demand on response at second-level, others at milliseconds or even microsecond level).

Correct result at the right time

Example:

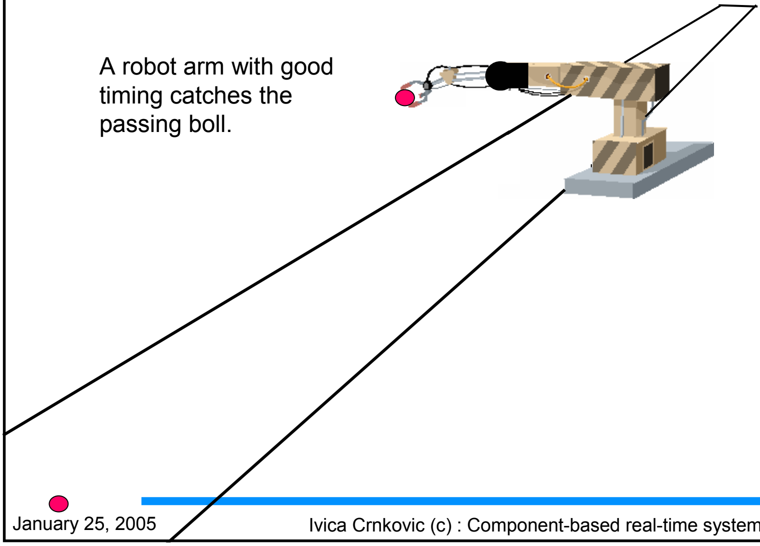
An air bag must not be inflated too late, nor too early!



In some cases the system must wait before it responds!

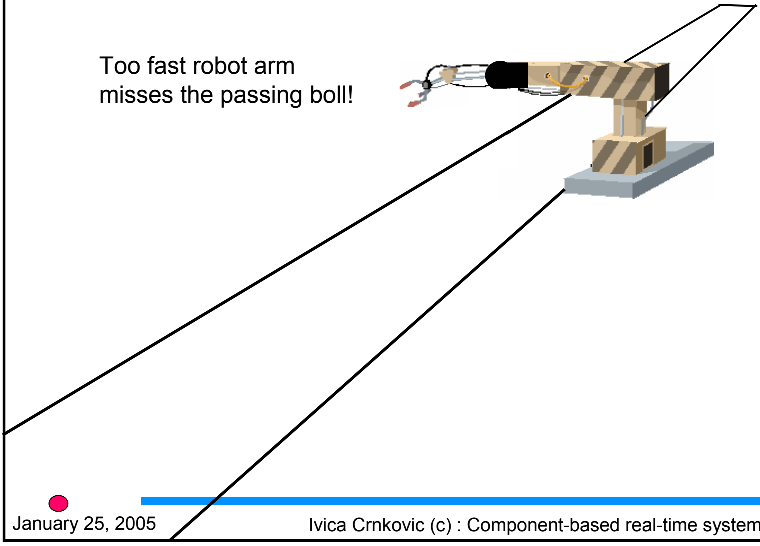
Predictable vs. fast

A robot arm with good timing catches the passing boll.



Predictable vs. fast

Too fast robot arm misses the passing boll!

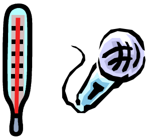


Interaction with the environment

A real-time system interacts with the environment via **sensors** and **actuators**

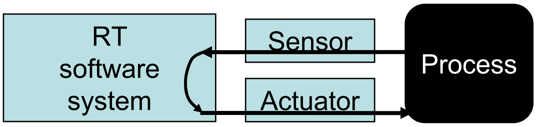
A **sensor** transforms physical data (temperature, pressure) to digital format

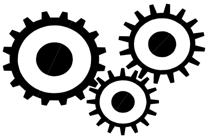
Examples: thermometer, microphone, video camera



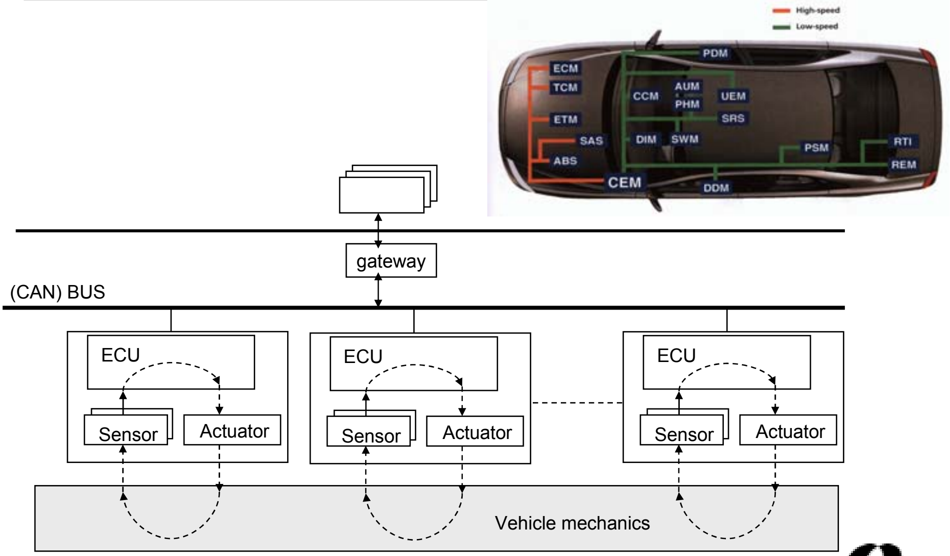
An **actuator** works the other way round - transforming digital data to physical format.

Example: motors, pumps, machines…





Embedded systems: **Volvo S80**



Classification of real-time systems

Resources

- Enough resources *(e.g., ABS break system)*
- System with limited resources *(e.g., telephone switches)*

Activation

- Event Triggered (ET) systems *(e.g., bank transaction systems)*
- Time Triggered (TT) systems *(e.g., aircraft control system)*

Service level

- Soft real-time systems *(e.g., multimedia systems)*
- Hard real-time systems *(e.g., airbag)*

Enough resources vs. Limited resources

Enough resources

- You can always guarantee that all functions in the system are able to execute when they so desire.
- Most often safety critical applications
- Expensive
- *Example: ABS-system, ”fly-by-wire”-system, power plant…*

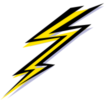
Limited resources

- There may be occasions when the system is unable to handle all functions that wants to execute.
- Designed to work well under normal conditions.
- *Example: telephone – everybody wants to make a phone call simultaneously will result in that some has to wait.*

Event driven vs. time driven systems

Event driven real-time systems

- External events determines when a program is to be executed
- Often through interrupts
- *<u>Example</u>: telephone switches, ”video-on-demand”, transaction systems…*



Time driven real-time systems

- The system handles external events at predefined points in time
- Most often cyclic systems → repeats a certain scenario
- *<u>Example</u>: ABS, control systems, manufacturing systems…*

Hard vs. Soft real-time systems

Hard real-time systems

- The cost for not fulfilling the functional and temporal constraints are severe
- Failing to meet hard real-time constraints results in computations, at best, being useless
- Often safety critical → the correctness must be verified before system operation
- *<u>Example</u>: ABS, airbag, defence system, power plant…*

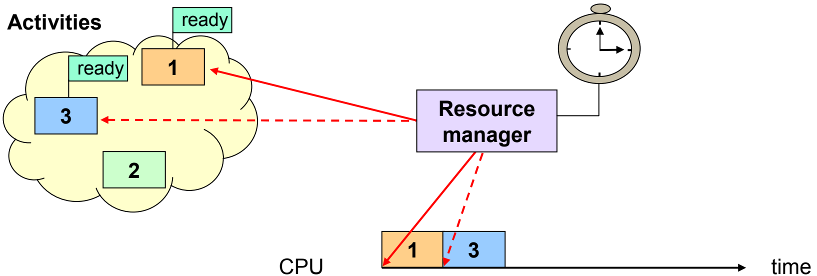
Soft real-time systems

- Occasional miss of fulfilling a timing constraint can be acceptable
- The usefulness of the computation is reduced (reduced service) *<u>Example</u>: reservation systems, ATM machines, multimedia, virtual reality…*

Challenges when constructing RT systems

Most of the real-time systems are based on following:

1. Several parallel activities are given some unique priorities
2. A resource manager makes sure the task with the highest priority will execute



Definition – task

Task

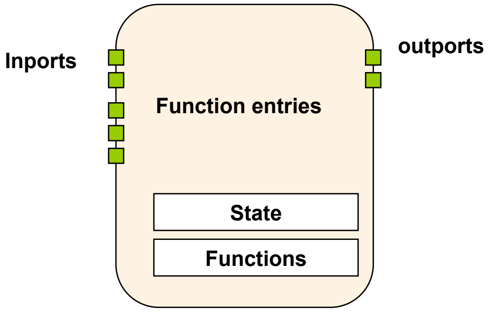
- A **sequential program** performing an activity and that possibly communicates with other tasks in the system. A task often has a **priority** relative to other tasks in the system.
- Sometimes **thread** is used instead of task.

Process

- A virtual processor that can handle **several tasks** with a common memory space.

Task

- A task is specified by a temporal behavior, function and state



Inports

outports

Function entries

State

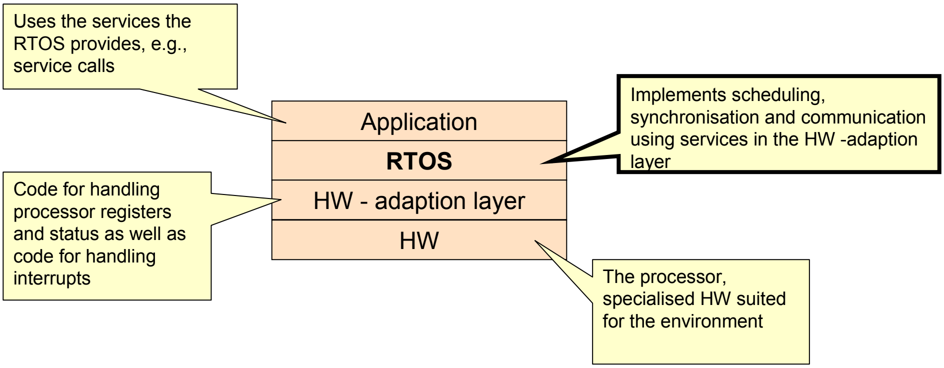
Functions

Taks Properties:
Execution time
Memory consumption
…..

System specification:
Period
Release Time
Deadline

Real-Time Operating System

- A platform for development of RTS Systems
- The development of RTS application software becomes easier and more effective



Uses the services the RTOS provides, e.g., service calls

Application

RTOS

HW - adaption layer

HW

Implements scheduling, synchronisation and communication using services in the HW -adaption layer

Code for handling processor registers and status as well as code for handling interrupts

The processor, specialised HW suited for the environment

Types of real time operating systems

Event triggered RTOS

- Each task has a **priority**
- Among the tasks willing to execute, the task with highest priority gets to
- Priority assignment before or during execution
- *Example: WxWorks from Windriver, Vertex from Mentor, Spring from Umass*

Time triggered RTOS

- Tasks are executed according to a **schedule** determined before the execution
- Time acts a means for synchronisation
- *Example: Rubus from Arcticus Systems, TTP from TTTech.*

Time triggered (TT) RTOS

A TT RTOS is often implemented by

- A **time table** (schedule)
- An RTOS executes the programs according to the time table
- A time table can either be *pre-emptive* or *non-pre-emptive*

Conflicts between tasks

- Resolved in the schedule, before system starts to run
- Requires a deeper understanding and knowledge about the system

Parallel from day-to-day life: time tables for trains

- Assuming no delays or break downs on trains can occur, we could eliminate the whole signalling system for trains (this due to the construction of the time table is such that no two trains can reside at the same railway section)

Task types

To implement real-time systems, we need:

- Models to describe the system (“constructive model”)
- Analysis to predicts the system’s behavior (“analytical model”)

A useful task model should be able to express:

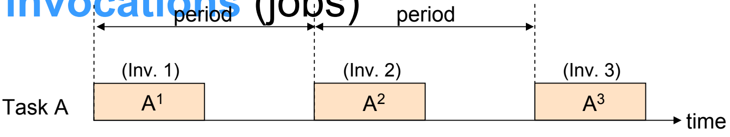
- Timing requirements
- Shared resources
- Communication
- Precedence demands

Task types

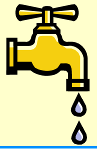
- Periodic tasks
- Sporadic tasks
- Aperiodic tasks

Periodic tasks

An infinite sequence of identical activities – **invocations** (jobs)



Analogy from real life:

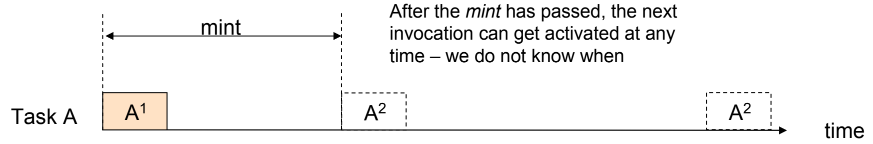


Example:

- Audio and video sampling
- Regulating
- Monitoring of temperature and pressure

Sporadic tasks

Known **minimum interarrival time (mint)** between two consecutive invocations



After the *mint* has passed, the next invocation can get activated at any time – we do not know when

Analogy from real life:

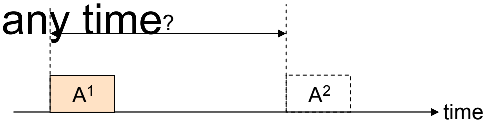
Laying an egg – A hen never lays more than one egg per day. After laying an egg, we do not know when exactly the next egg will come, but we do know that it has to pass at least 24 hours from the previous one

Example:

- Regulation of water tanks
- Detection of enemy missiles in a fighter jet

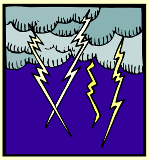
Aperiodic tasks

- Non-periodic execution
- Events that triggers an aperiodic task may occur at any time



Analogy from real life:

Lightning – can strike at any time (in right conditions)



Example:

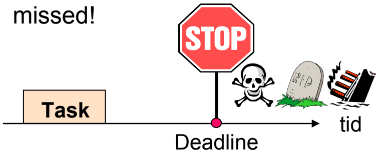
- A device generates interrupts
- An operator presses the emergency button
- Alarms

What is a deadline?

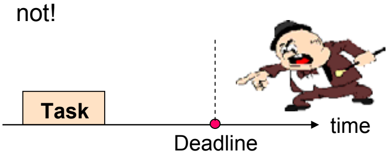
Task deadline is the latest point in time at which the task has to be completed

- Absolute deadline
- Relative deadline

Hard real-time systems: deadlines must not be missed!



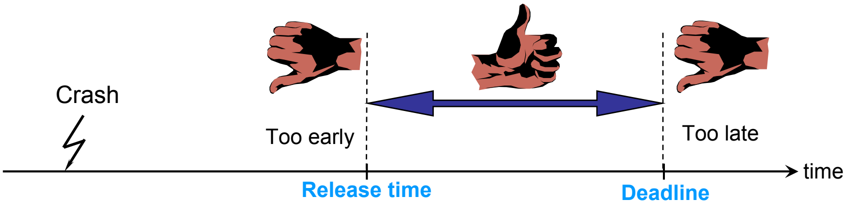
Soft real-time systems: deadlines can be missed, but preferably they are not!



What is a release time?

Task release time is the earliest point in time we can activate (release) the task

Airbag-example:



Definition of periodic task

Periodic task *i* can be described with:

T_i – period time, i.e., how often the task *i* gets ready

RT_i – release time, i.e., earliest start time for task *i*

C_i – execution time during a period time

D_i – relative deadline, i.e., the latest point in time within which task *i* has to complete, counted from the current period start

---

Event ordering

Events can be:

- Predictable → time triggered tasks
- Unpredictable → event triggered tasks

What happens if several events occur at the same time?

- We must have some **event ordering mechanism**!

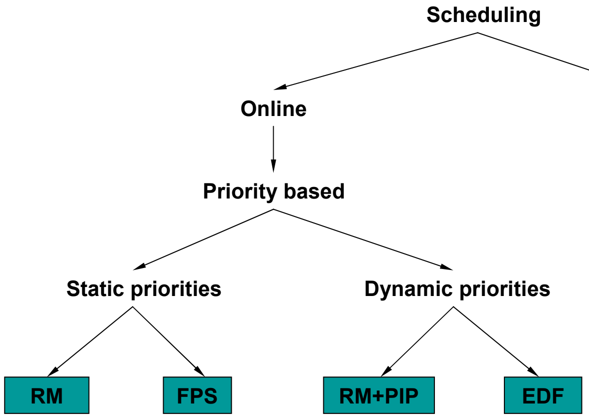
Example:

Assume two tasks handling two simultaneous events:

Task A: deadline = 5	**Task B: deadline = 7**
execution time = 3	**execution time = 4**

Does it matter in which order we execute the tasks?

Simple classification of scheduling algorithms



RM	Rate Monotonic
FPS	Fixed Priority Scheduling
EDF	Earliest Deadline First
PIP	Priority Inheritance Protocol

Online vs offline scheduling

Online scheduling

(+) flexible
(+) relatively simple analysis

(-) difficult to cope with complex constraints
(-) less deterministic

Offline scheduling

(+) deterministic
(+) simplier to test and verify
(+) handles complex constraints

(-) new schedule must be generated if we add a new function
(-) it could take a long time to produce a schedule

Online vs. offline scheduling

When to use each of the methods?

Offline scheduling

- High demands on timing and functional verification, testability and determinism
- Safety-critical applications, e.g., control system for Boeing 777

Online scheduling

- Demands on flexibility, meny non-periodic activities

What do wee need to schedule tasks?

Task properties:

- **Execution time:**
  - **Worst case execution time**
  - **Best case execution time**

- **Precedence requirements**
  - **Resources**
  - **Tasks**

- **Properties setup from the system requirements**
  - **Execution period**
  - **Release**
  - **Deadline**

  - **Priority**

Execution time analysis – WCET analysis

WCET = Worst Case Execution Time

- We want to find out how long time a task needs for its execution before we run the task

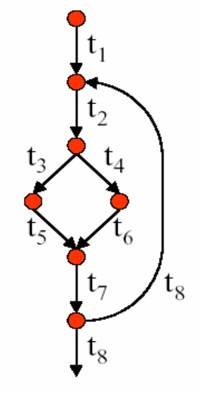
Why WCET analysis?

- Predictability!
- Pre-requisite for any schedulability analysis

How can we get WCET?

- measuring + safety margins
  - hazardous
  - difficult to measure in a right way
- Analysis
  - safe (can be proven)
  - difficult

Task code



What is the longest execution time (WCET)?

Execution time analysis

Earlier presented scheduling algorithms:

- **C_i = WCET**
- Can not guaranty correct timing behavior if WCET is wrong

Execution time depends on:

- Input data
- Program logic (defined by source code)
- Compiler
- Hardware (CPU, memory,…)

Other relevant values:

- **BCET** – *Best Case Execution Time*
- **ACET** – *Average Case Execution Time*

RT systems and component-based approach

- Why is CBD approach interesting for RT systems?
  - Reusability
  - Predictability
  - Managing complexity
- Why is CBD approach more difficult for development of RT systems?

Main challenges

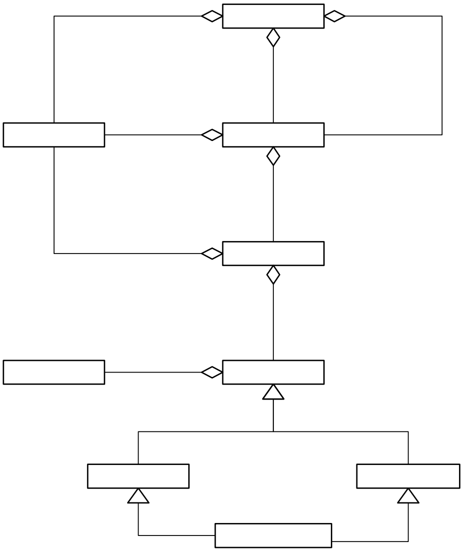
- How to specify components that they provide all information needed for RT design?
- How to flexible reuse specification of components when the environment is changed (problem with WCET, resources)?
- How to predict (RT) properties of the RT systems from the RT components?

Software Component Definition

Szyperski (Component Software beyond OO programming)

- A software component is
  - a unit of composition
  - with contractually specified interfaces
  - and explicit context dependencies only.
- A software component
  - can be deployed independently
  - it is subject to composition by third party.

Components and Interfaces - UML definition



Component – a set of interfaces
required (in-interfaces)
provided (out-interfaces)

Interface – set of operations
Operations – input and output parameters of certain type

Substitution

- Substituting a component Y for a component X is said to be safe if:
  - All systems that work with X will also work with Y
- From a syntactic viewpoint, a component can safely be replaced if:
  - The new component implements at least the same interfaces as the older components
- From semantic point of view?

Specifying the Semantics of Components

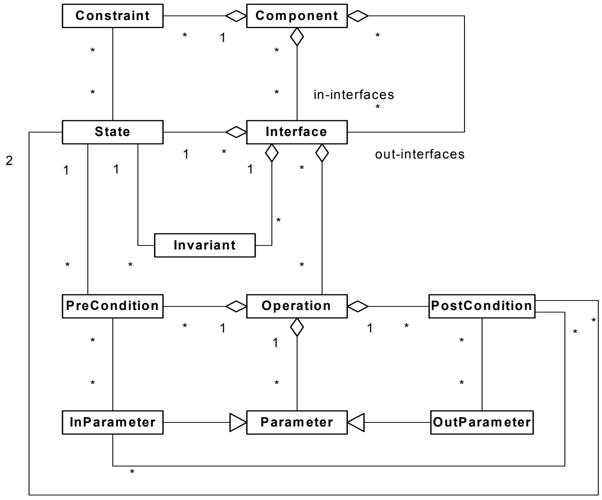
- Current component technologies assume that the user of a component is able to make use of such semantic information.
- Extension of Interface (adding semantics)
  - a set of interfaces that each consists of a set of operations.
  - a set of preconditions and postconditions is associated with each operation.
  - A set of invariants
- Also called: Contractually specified interfaces

Precondition, Postconditions, Invariants

- Precondition
  - an assertion that the component assumes to be fulfilled before an operation is invoked.
  - Will in general be a predicate over the operation's input parameters and this state
- Postcondition
  - An assertion that the component guarantees will hold just after an operation has been invoked, provided the operation's pre-conditions were true when it was invoked.
  - Is a predicate over both input and output parameters as well as the state just before the invocation and just after
- Invariant
  - Is a predicate over the interface's state model that will always hold

---

Semantic Specification in a UML metamodel



Extrafunctional properties

- Extrafunctional (non-functional) properties
  - runt-time properties
    - Performance, latency
    - Dependability (Reliability, robustness, safety)
  - Life cycle properties
    - Maintainability, usability, portability, testability,….
- There is no standards for specification of extrafunctional properties

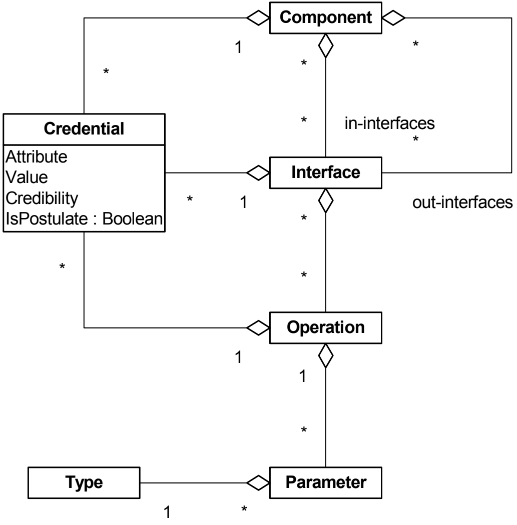
---

Extrafunctional properties specifications

Credentials (Mary Shaw)

- A Credential is a triple <Attribute, Value, Credibility>
  - Attribute: is a description of a property of a component
  - Value: is a measure of that property
  - Credibility: is a description of how the measure has been obtained
- Attributes in .NET
  - A component developer can associate attribute values with a component and define new attributes by sub-classing an existing attribute class.
- ADL UniCon
  - allows association of <Attribute, Value> to components
  - UML 2.0

Extra-functional Properties


Component

Credential

Attribute
Value
Credibility
IsPostulate : Boolean

Interface

Operation

Type

Parameter

in-interfaces

out-interfaces

Can we use de-facto standard component models?

- Commonly used CBSE technologies in not-RTS (EJB, CORBA and COM) are seldom used in RTS as they:
  - Require excessive processing requirements
  - Require excessive memory requirements
  - Provide unpredictable timing characteristics
  - Have no means for specifying RT properties

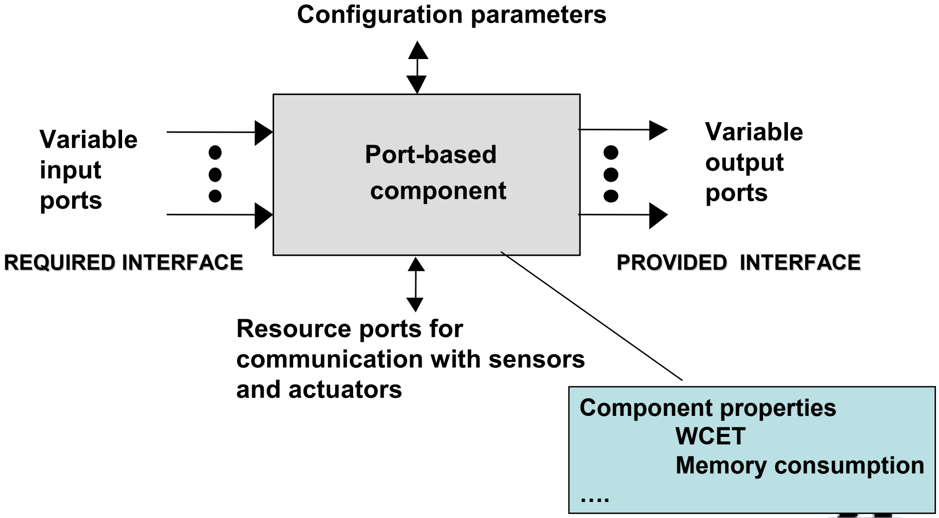
What is an RT component?

- It should include
  - Functional specification
    - Interface (provided and required)
  - Real-time specifications
    - Execution time
      - WCET
      - Average execution time
  - Requirements for the environment (resources)
  - Parameters that can be setup (period, priority,…)

What is an RT component?

- A simplest solution:
  - A component is a task
- A more complex (and more powerful) solution:
  - A component is a set of tasks
- OR
  - Many components build a task
- How do we specify a component?

A Port-based Component model



Configuration parameters

Variable input ports

REQUIRED INTERFACE

Port-based component

Variable output ports

PROVIDED INTERFACE

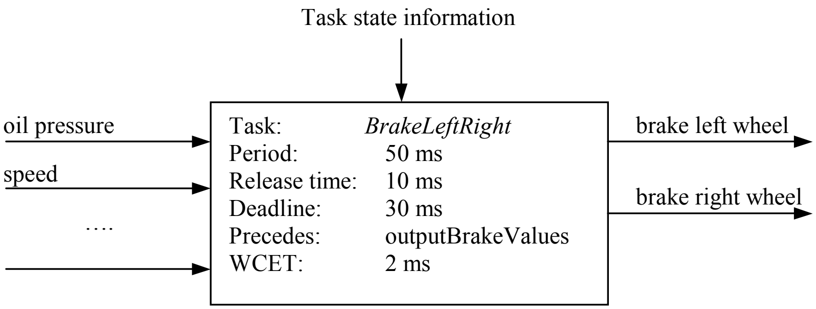
Resource ports for communication with sensors and actuators

Component properties
WCET
Memory consumption
….

Example: A task as a component

Component-model used in REBUS (Volvo construction equipment)

- The timing requirements are specified by release-time, deadline, WCET and period



Task state information

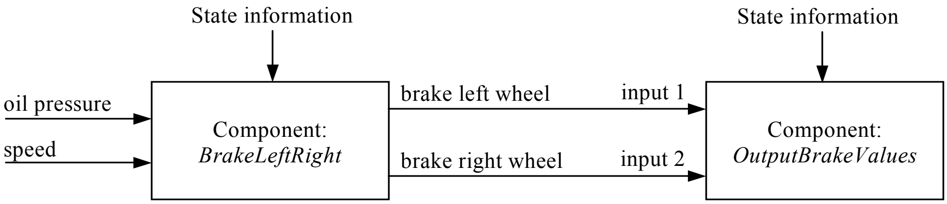
oil pressure
speed
….

Task:	*BrakeLeftRight*
Period:	50 ms
Release time:	10 ms
Deadline:	30 ms
Precedes:	outputBrakeValues
WCET:	2 ms

brake left wheel
brake right wheel

Component composition

- *BrakeLeftRight* precedes *outputBrakeValues*).



State information

State information

oil pressure

speed

Component:
BrakeLeftRight

brake left wheel

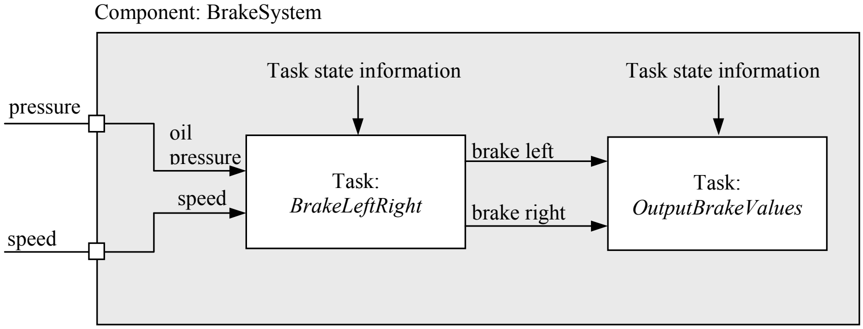
input 1

brake right wheel

input 2

Component:
OutputBrakeValues

Composition of Components



Component: BrakeSystem

Task state information

Task state information

pressure

speed

oil
pressure

speed

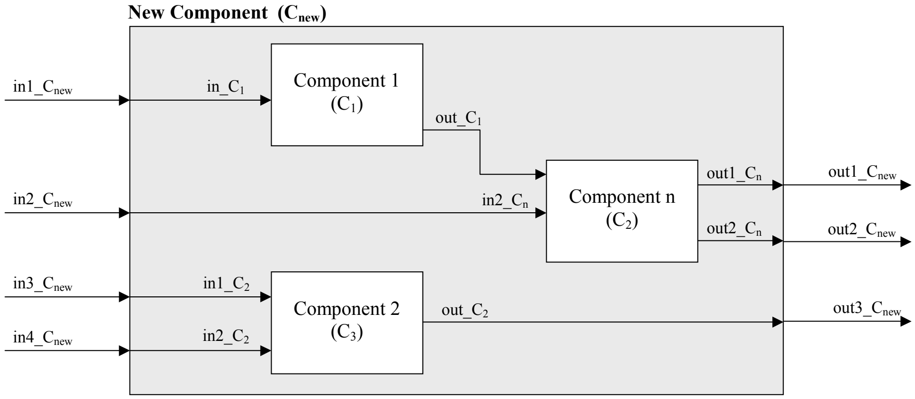
Task:
BrakeLeftRight

brake left

brake right

Task:
OutputBrakeValues

Composition of Components



New Component (C_{new})

in1_C_{new}

in2_C_{new}

in3_C_{new}

in4_C_{new}

in_C_1

Component 1 (C_1)

out_C_1

in2_C_n

Component n (C_2)

out1_C_n

out2_C_n

in1_C_2

in2_C_2

Component 2 (C_3)

out_C_2

out1_C_{new}

out2_C_{new}

out3_C_{new}

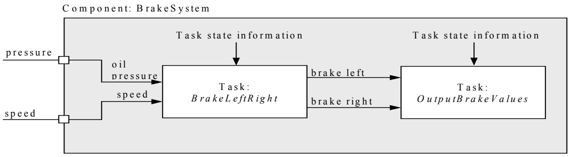
What is with time attributes?

- How do we specify properties of an assembly?
  - Execution time (WCET,…)
- How do we map assembly properties to the components being composed?
  - Period?
  - Priority?

Specification Of Time Attributes

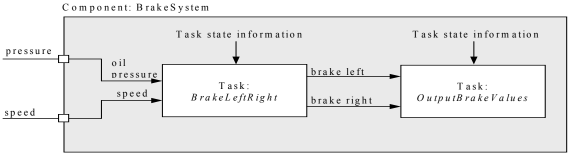
- We specify “virtual time attributes” of the composed component, which are used to compute the timing attributes of sub-components, ie:

IF virtual period is set to *P,*
THEN the period of a sub-component A should be $P_A = f_A * P$
AND the period of B is $P_B = f_B$ * P,
WHERE f_A and f_B are constants for the composed component



Properties of Composed components

- Can we use WCET?
  - No
  - WCET cannot be computed since its parts may be executing with different periods.
- End-to-end deadlines
  - Are set such that the system requirements are fulfilled
  - Should be specified for the input to and output from the component
- Latency – time for an assembly to respond to input signal
  - Average, Worst case (end-to-end deadline), best case



Substitution principles

- When we can replace a component?
- Goal: on-line upgrade task components in a ‘safe’ way

- Two issues:
  - new components must not be faulty
  - schedulability of all tasks must be guaranteed

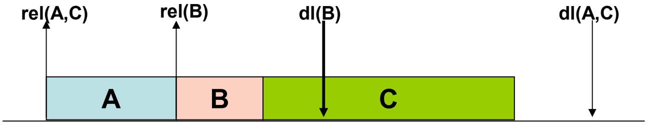
- Existing approach – Sha 1998
  - basic idea: monitor output to ensure values within valid range
  - run-time upgrade possible if
    WCET (new comp) ≤ WCET (old comp)

- Problem :
  - tasks execute for less then WCET
    - order of task execution may change
    - deadlines can be missed

Example 1: preemptive FPS

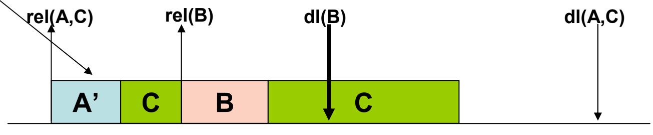
task	**priority**
A	**high**
B	**medium**
C	**low**

(a)



A is replaced by A’; wcet(A’)<wcet(A)

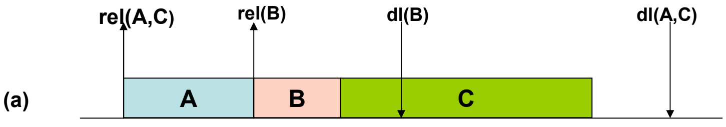
(b)



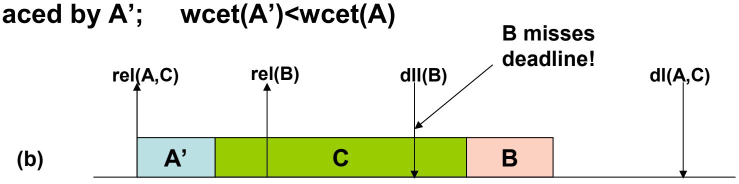
Order of execution changed – deadline met

Example 2: non-preemptive FPS

task	priority
A	high
B	medium
C	low



A is replaced by A'; wcet(A')<wcet(A)



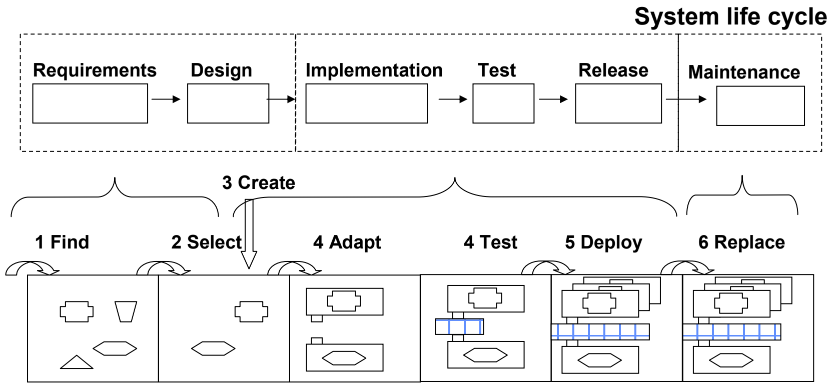
On-line upgrading of real-time components in priority-based RTS

- Solution?
  - preemptive FPS
    - eliminate/predict preemptions at design stage
  - non-preemptive FPS
    - work on the WCET
    - predict exact execution time?
- Another solution
  - Off-line scheduling
  - Missing possibility of on-line upgrading of a component

Component-based development process

Component-based development process

- Basic principles
  - Separation of development processes. Components may be developed separately from the systems



System life cycle

Requirements → **Design** → **Implementation** → **Test** → **Release** → **Maintenance**

3 Create

1 Find

2 Select

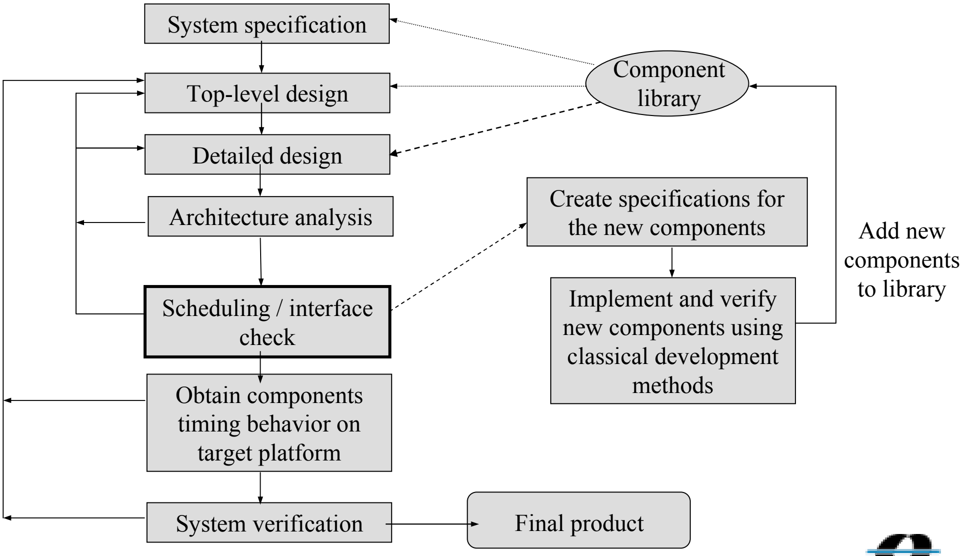
4 Adapt

4 Test

5 Deploy

6 Replace

Designing Component-based RT System



System specification

Top-level design

Detailed design

Architecture analysis

Scheduling / interface check

Obtain components timing behavior on target platform

System verification

Final product

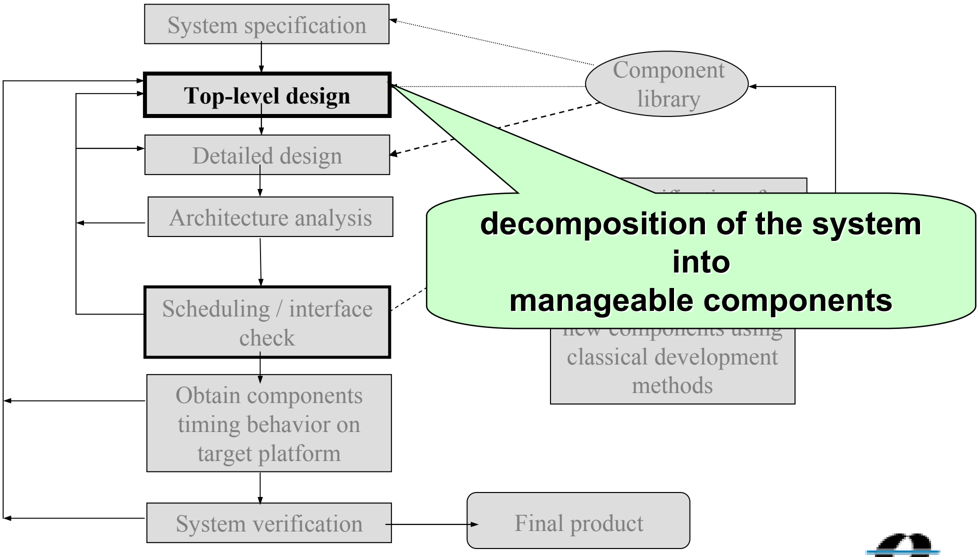
Component library

Create specifications for the new components

Implement and verify new components using classical development methods

Add new components to library

Designing Component-based RT System



System specification

Top-level design

Detailed design

Architecture analysis

Scheduling / interface check

Obtain components timing behavior on target platform

System verification

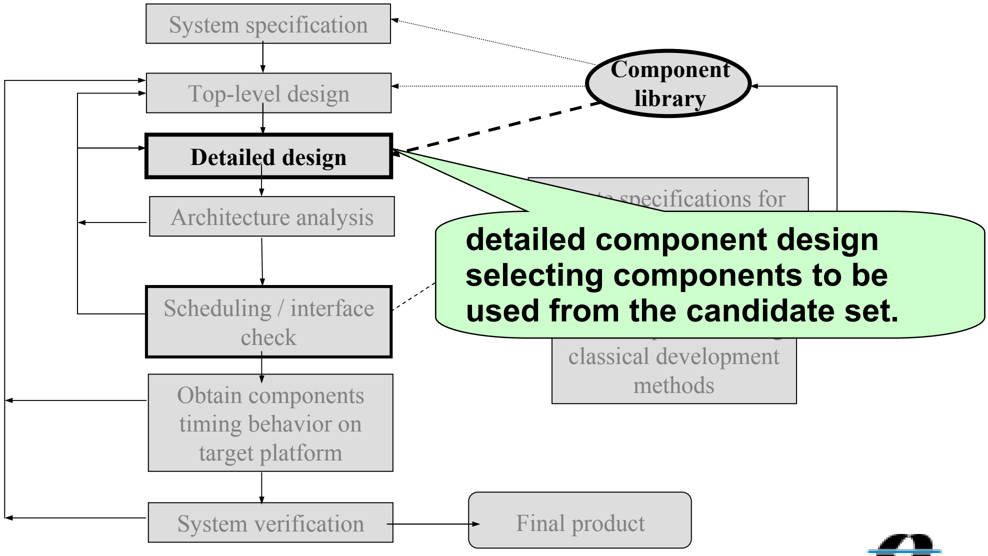
Final product

Component library

new components using classical development methods

decomposition of the system into manageable components

Designing Component-based RT System



System specification

Component library

Top-level design

Detailed design

te specifications for

Architecture analysis

Scheduling / interface check

classical development methods

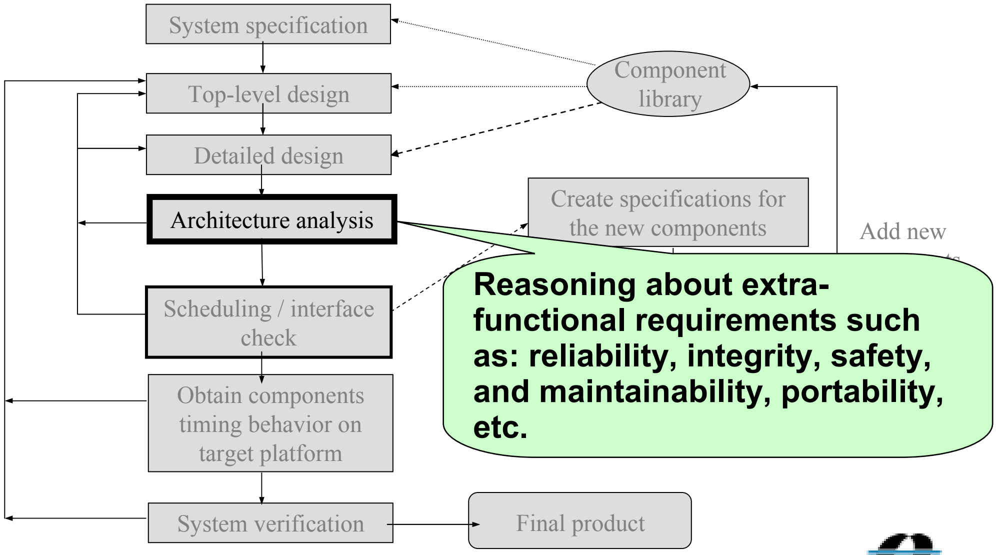
Obtain components timing behavior on target platform

System verification

Final product

detailed component design selecting components to be used from the candidate set.

Designing Component-based RT System



System specification

Component library

Top-level design

Detailed design

Architecture analysis

Create specifications for the new components

Add new

Scheduling / interface check

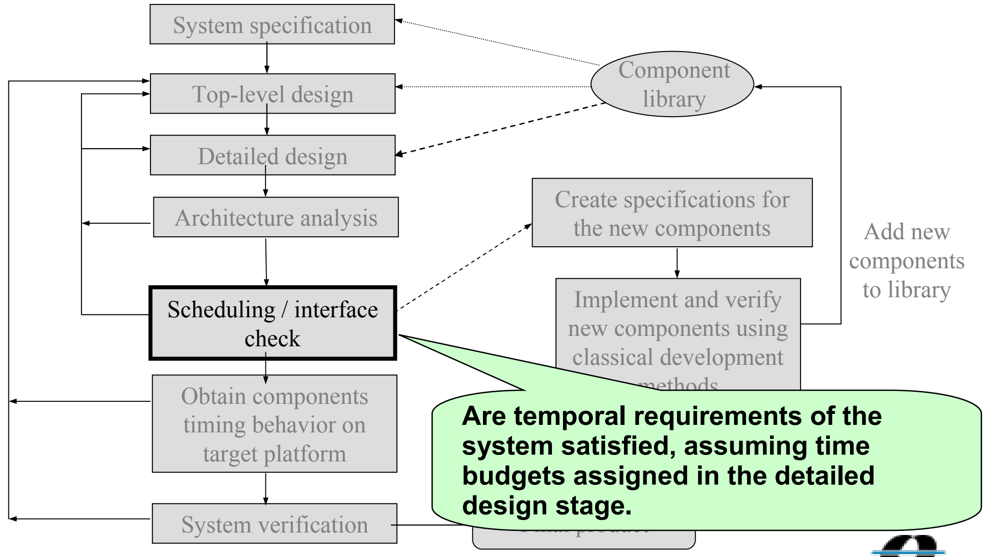
Obtain components timing behavior on target platform

System verification

Final product

Reasoning about extra-functional requirements such as: reliability, integrity, safety, and maintainability, portability, etc.

Designing Component-based RT System



System specification

Top-level design

Detailed design

Architecture analysis

Scheduling / interface check

Obtain components timing behavior on target platform

System verification

Component library

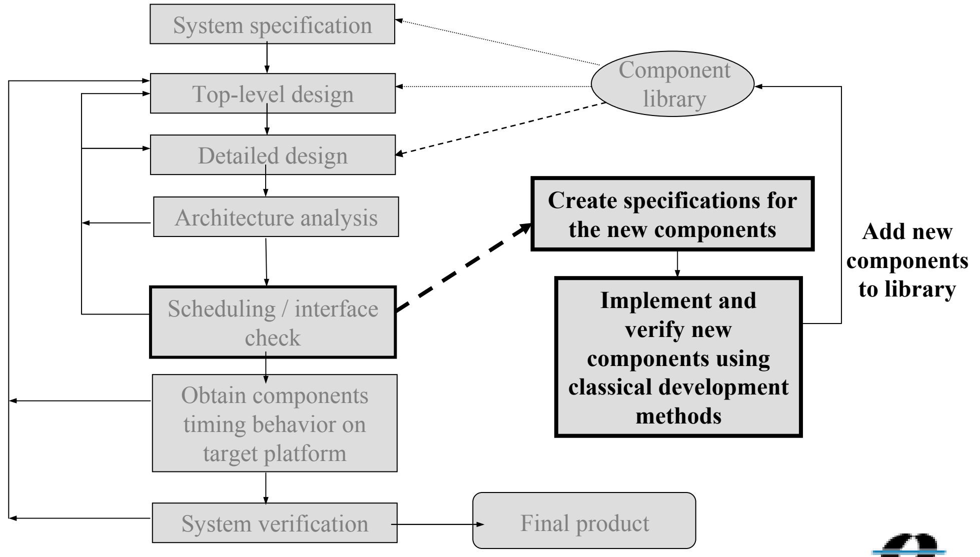
Create specifications for the new components

Implement and verify new components using classical development methods

Add new components to library

Are temporal requirements of the system satisfied, assuming time budgets assigned in the detailed design stage.

Designing Component-based RT System



System specification

Top-level design

Detailed design

Architecture analysis

Scheduling / interface check

Obtain components timing behavior on target platform

System verification

Final product

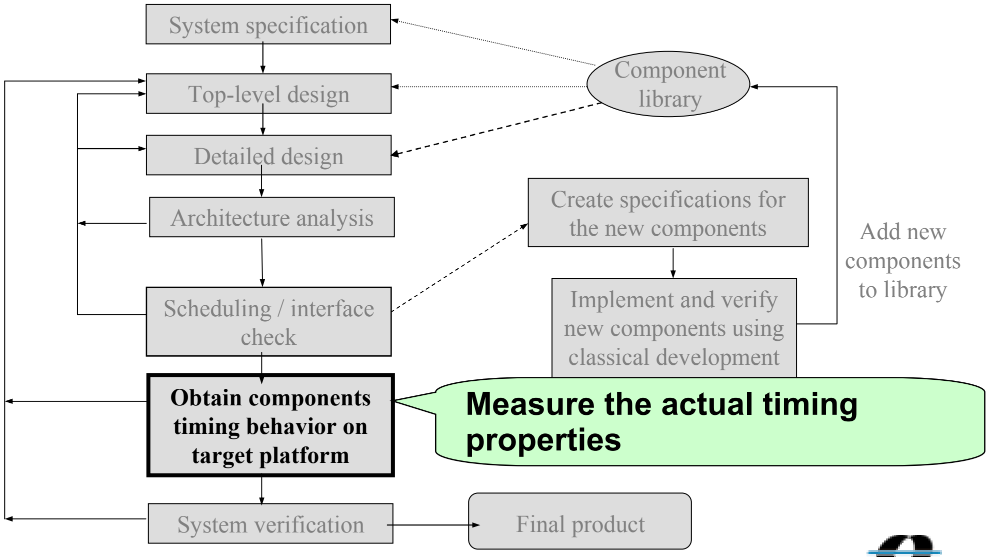
Component library

Create specifications for the new components

Implement and verify new components using classical development methods

Add new components to library

Designing Component-based RT System



System specification

Top-level design

Detailed design

Architecture analysis

Scheduling / interface check

Obtain components timing behavior on target platform

System verification

Final product

Component library

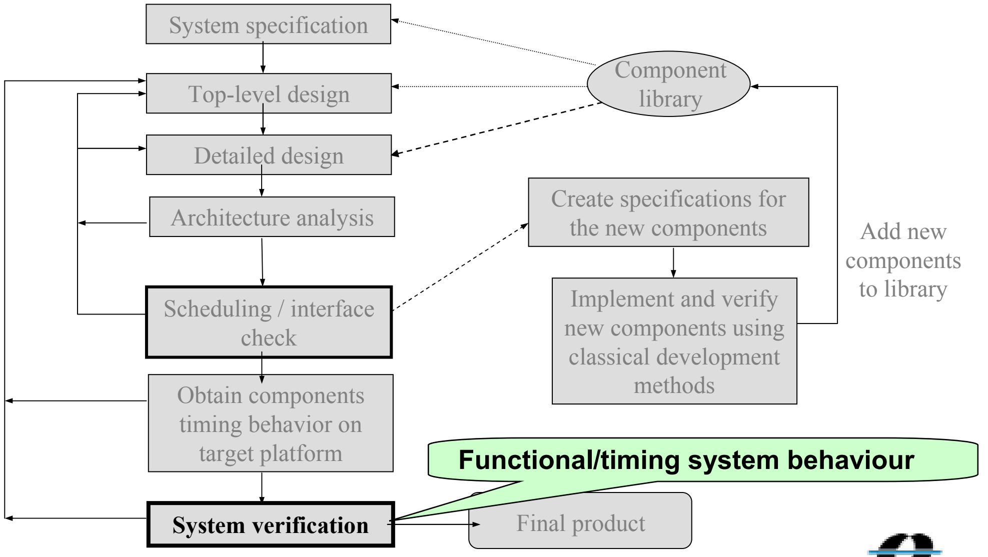
Create specifications for the new components

Implement and verify new components using classical development

Add new components to library

Measure the actual timing properties

Designing Component-based RT System



System specification

Top-level design

Detailed design

Architecture analysis

Scheduling / interface check

Obtain components timing behavior on target platform

System verification

Final product

Component library

Create specifications for the new components

Implement and verify new components using classical development methods

Add new components to library

Functional/timing system behaviour

Component technology in embedded world

We should consider:

- Contractually specified interfaces
- Managing extrafunctional properties
- Component as a unit of composition and independent deployment
- Explicit context dependencies
- Component granularity
- Reuse
- Location transparency
- Component wiring
- Portability, platform independence

Unit of composition and independent deployment

- Run-time composition
  - Component lifecycle,
  - Run-time environment,
  - Dynamic composition (binding)

Component technology

- Configuration composition
  - Capable of generating monolithic firmware from component-based design,
  - Optimization

More feasible for embedded systems

Explicit context dependencies

What is a context in embedded world?

- Other components and interfaces
- Run-time environment
  - CPU,
  - RTOS,
  - Component implementation language,
  - Resource constraints

Component technology

embedded systems specific

Component granularity

- Coarse-grained components
  - Often distributed components
  - Too heavy bag of unnecessary functionality,
  - Too much resources used

Component technology

- Fine-grained components,
  - Light, unneeded functionality reduced,
  - Scarcer uses of resources,

More feasible for embedded systems

Reuse

- Black-box reuse
  - From component's user point of view

Component technology

- White-box reuse
  - From composition environment point of view
- Gray-box reuse (composition environment)
  - If clear conventions for knowledge about implementation are introduced

More feasible for embedded systems

Portability, Platform independence

- Binary independence

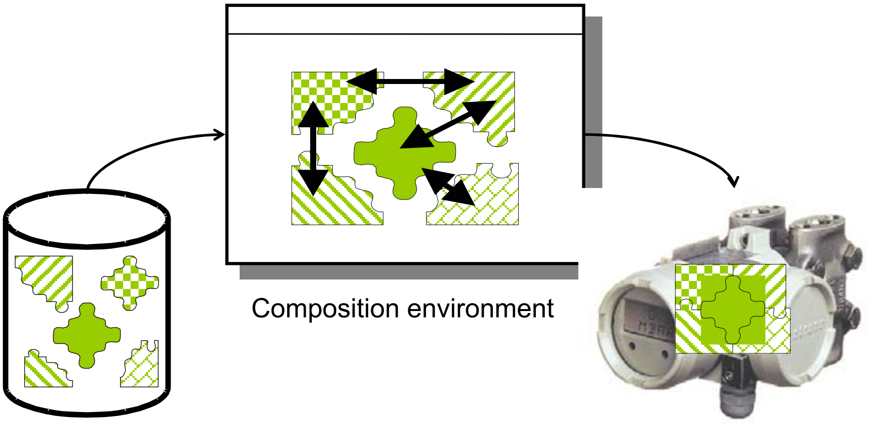
Component technology

- Source level portability suffices,
  - Design-time composition,
  - Run-time environment restrictions

More feasible for embedded systems

- Source level portability requires:
  - Agreement on implementation language,
  - Agreement on available libraries,
  - Providing proper abstractions (i.e. RTOS API)

Framework



Composition environment

Component Repository

Run-time environment

Summary

- It is more difficult to apply component-based approach to real-time systems
- The main challenges are in
  - prediction of timing properties in different environments
    - Execution time – depends of CPU
  - Compositing of real-time properties
  - Substitution principles are not clear
- Composition (deployment) time is different from run time