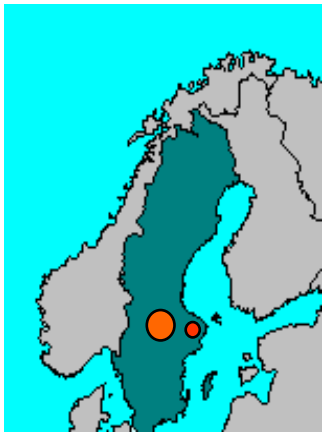
# Component-based approach for embedded systems

Ivica Crnkovic
Mälardalen University
Department of Computer Science and Engineering, Sweden
http://www.idt.mdh.se/~icc

# Mälardalen University (MdH)



**Mälardalen University, Vasteras (Västerås)**

**Prof. in Software Engineering**
**http://www.idt.mdh.se/~icc**
**ivica.crnkovic@mdh.se**

**Department of Computer Engineeri**
- **Real-Time Systems Design Lab**
- **Computer Architecture Lab**
- **Computer Science Lab**
- **Software Engineering Lab**

# Outline

- Basic characteristics of Component-based Software Engineering
- Component-based approach in different domains – benefits and challenges
- (Embedded systems – some examples)
- CBSE for different types of embedded systems
- Needs and challenges, research directions

# Sources of information

CBSE net
Component Based Software Engineering network

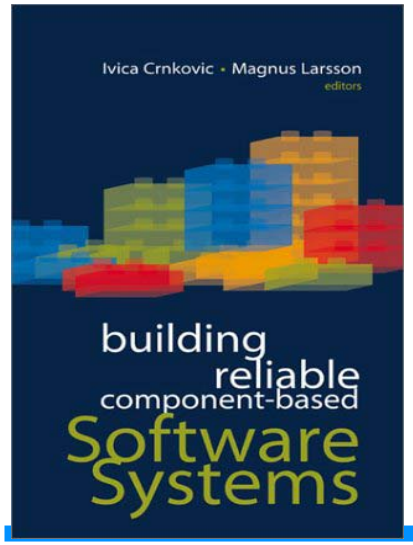
http://www.cbsenet.org/pls/CBSEnet/ecolnet.home

ARTIST

http://www-artist.imag.fr/Overview/

SAVE

http://www.mrtc.mdh.se/SAVE/




Ivica Crnkovic and Magnus Larsson:

Building Reliable Component-Based Software Systems

Artech House Publishers, ISBN 1-58053-327-2

http://www.idt.mdh.se/cbse-book/

MRTC
ARDALEN REAL-TIME
ESEARCH CENTRE

# Component-based approach

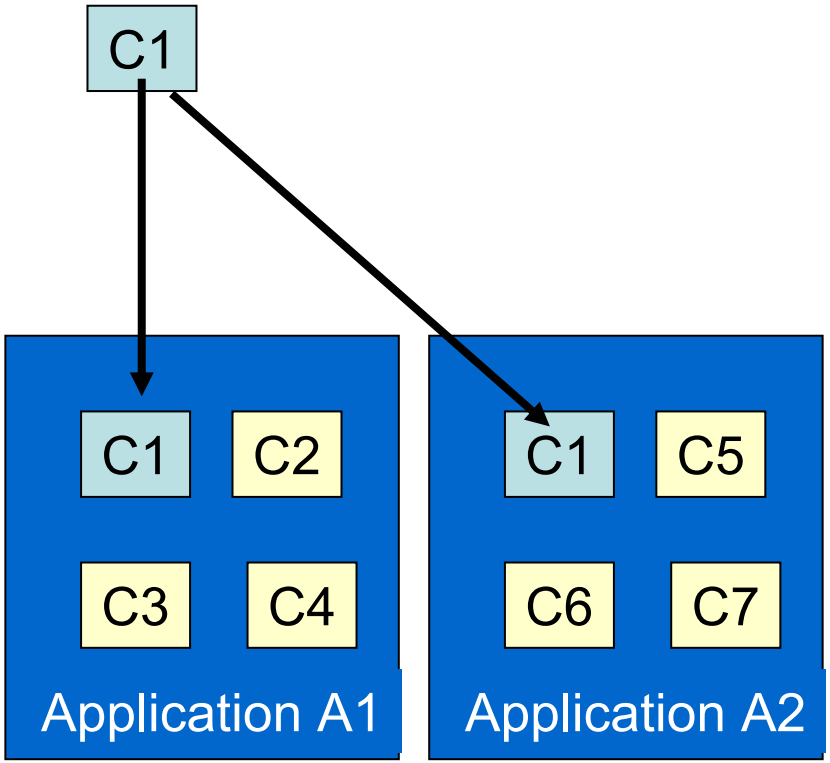
- Building systems from (existing) components
  - Providing support for the development of systems as assemblies of components
  - Supporting the development of components as reusable units
  - Facilitating the maintenance and evolution of systems by customizing and replacing their components
- Component-based Software Engineering
  - Provides methods and tools supporting different aspects of component-based approach
    - Process issues, organizational and management issues, technologies (for example component models), theories (component compositions),…

# Implications

- Component development is separated from system development process
  - Less programming efforts to build systems
  - System verification and validation  more difficult and more important
  - Different requirements management
- A combination of a bottom-up and top-down approach
- Many explicit and implicit assumptions
  - Architectural styles (middleware, deployment,..)

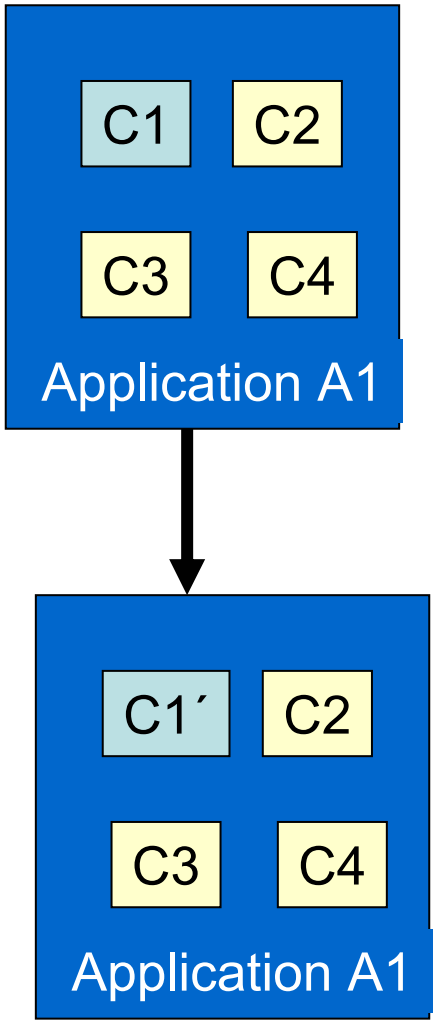
# Main principles: (1) Reusability

- Reusing components in different systems
- The desire to reuse a component poses few technical constraints.
  - Good documentation (component specification…)
  - a well-organized reuse process
  - Similar architecture
  - ….



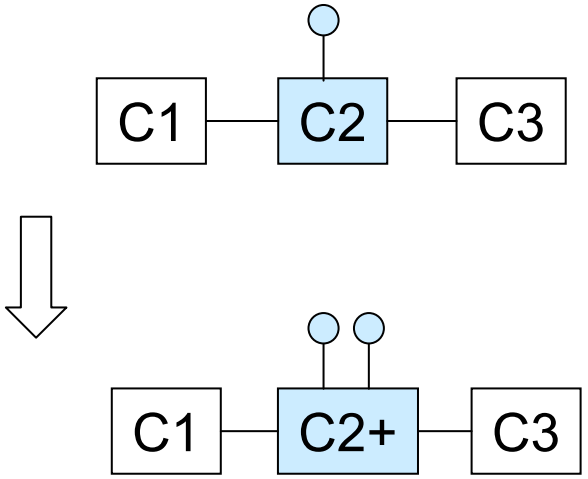
# Main principles: (2) Substitutability

- Alternative implementations of a component may be used.
- The system should meet its requirements irrespective of which component is used.
- Substitution principles
  - Function level
  - Non-functional level
- Added technical challenges
  - Design-time: precise definition of interfaces & specification
  - Run-time: replacement mechanism

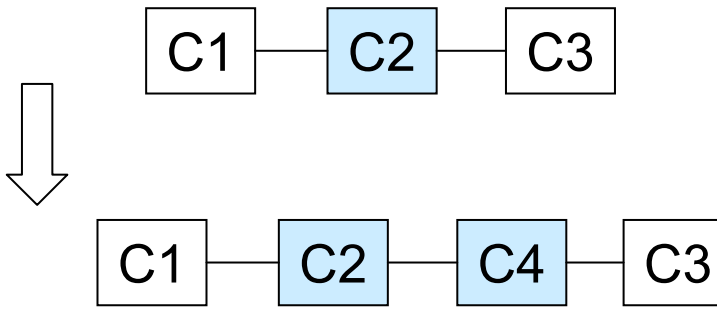


# Main principles: (3) Extensibility

- Comes in two flavors:
  - extending components that are part of a system
  - Increase the functionality of individual components

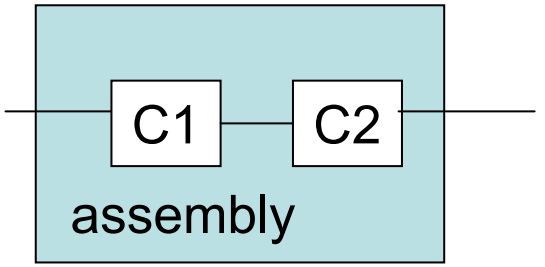



- Added technical challenges:
  - Design-time: extensible architecture
  - Run-time: mechanism for discovering new functionality



# Main principles: (4) composability

- Composition of components
  - F(c1 o c2) =F1(c1) o F2(c2)
  - Composition of functions
  - Composition of non-functional properties




- Many challenges
  - How to reason about a system composed from components?
    - Different type of properties
    - Different principles of compositions

# 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.

# Another definition

- **The software architecture of a program or computing system is the structure or structures of the system,** which comprise software components **[and connectors], the** externally visible properties of those components **[and connectors] and** the relationships among them**.”**

  **Bass L., Clements P., and Kazman R., *Software Architecture in Practice*,**

# Implications of Szyperski's Definition

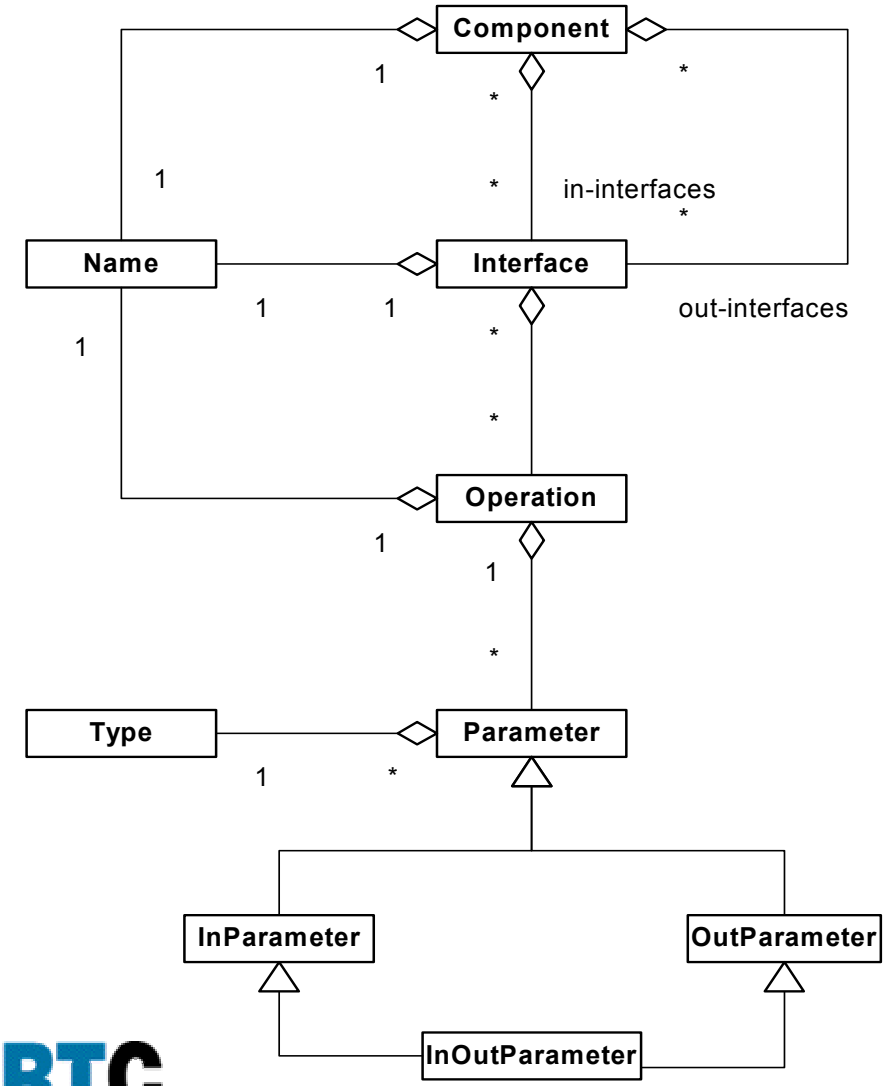
- The following implications arise as a result of Szyperski's definition:
  - For a component to be deployed independently, a clear distinction from its environment and other components is required.
  - A component must have clearly specified interfaces.
  - The implementation must be encapsulated in the component and is not directly reachable from the environment.

  (Black box nature)

# Describing a Component

- To be able to describe a component completely the component should consist of the following elements:
  - A set of interfaces provided to, or required from the environment.
  - An executable code, which can be coupled to the code of other components via interfaces.

# 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**

# IDL Example

```
interface ISpellCheck : IUnknown
{
    HRESULT check([in] BSTR *word, [out] bool *correct);
};

interface ICustomSpellCheck : IUnknown
{
    HRESULT add([in] BSTR *word);
    HRESULT remove([in] BSTR *word);
};

library SpellCheckerLib
{
    coclass SpellChecker
    {
            [default] interface ISpellCheck;
            interface ICustomSpellCheck;
    };
};
```

# 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?

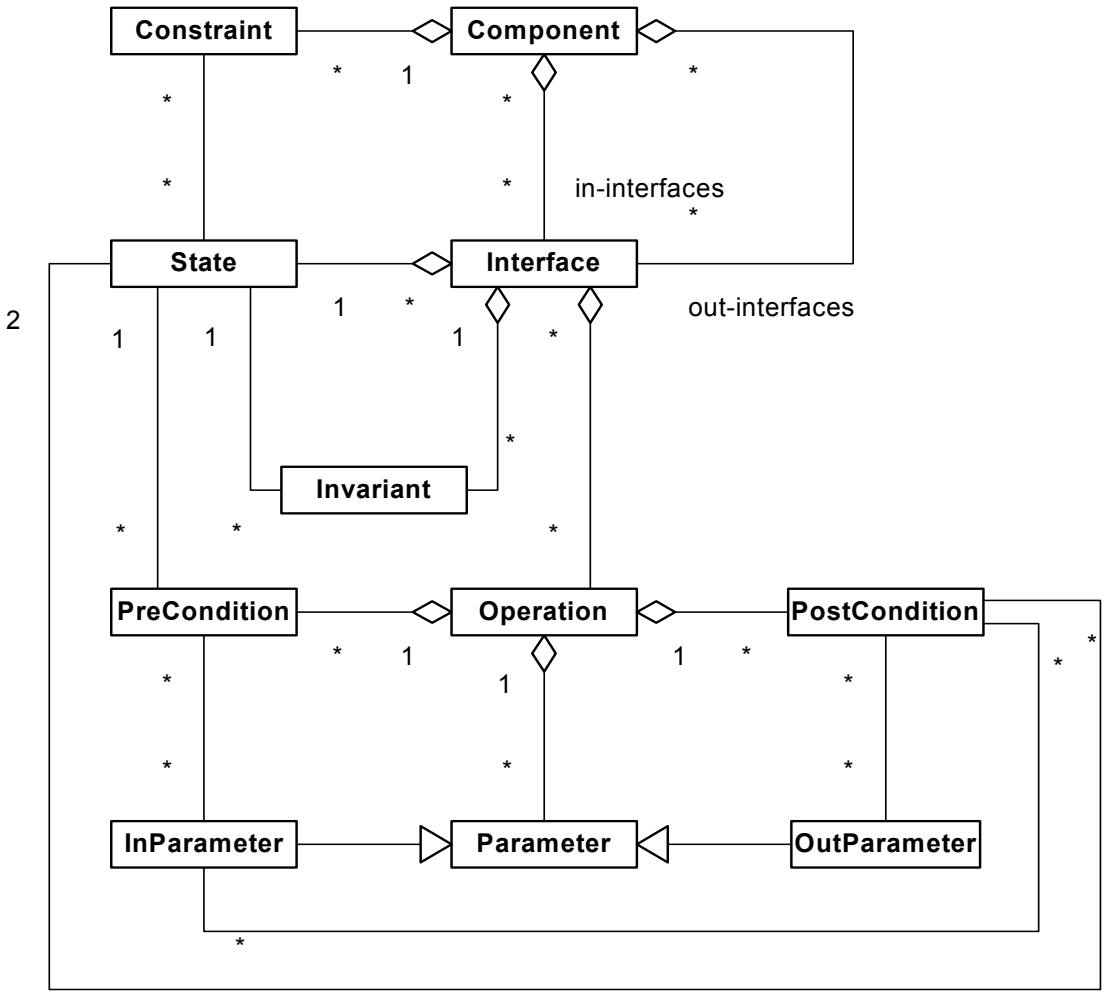
# Specifying the Semantics of Components

- 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



# Semantic Interface Specification

```
context ISpellCheck::check(in word : String, out correct :       Boolean):
RESULT
  pre:
  word <> ""
  post:
  SUCCEEDED(result) implies correct = words->includes(word)

  context ICustomSpellCheck::add(in word : String) : HRESULT
  pre:
  word <> ""
  post:
  SUCCEEDED(result) implies  words = words@pre->including (word)

  context ICustomSpellCheck::remove(in word : String) : HRESULT
  pre:
  word <> ""
  post:
  SUCCEEDED(result) implies words = words@pre->exluding(word)
```

# Extrafunctional properties

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

# CBSE questions

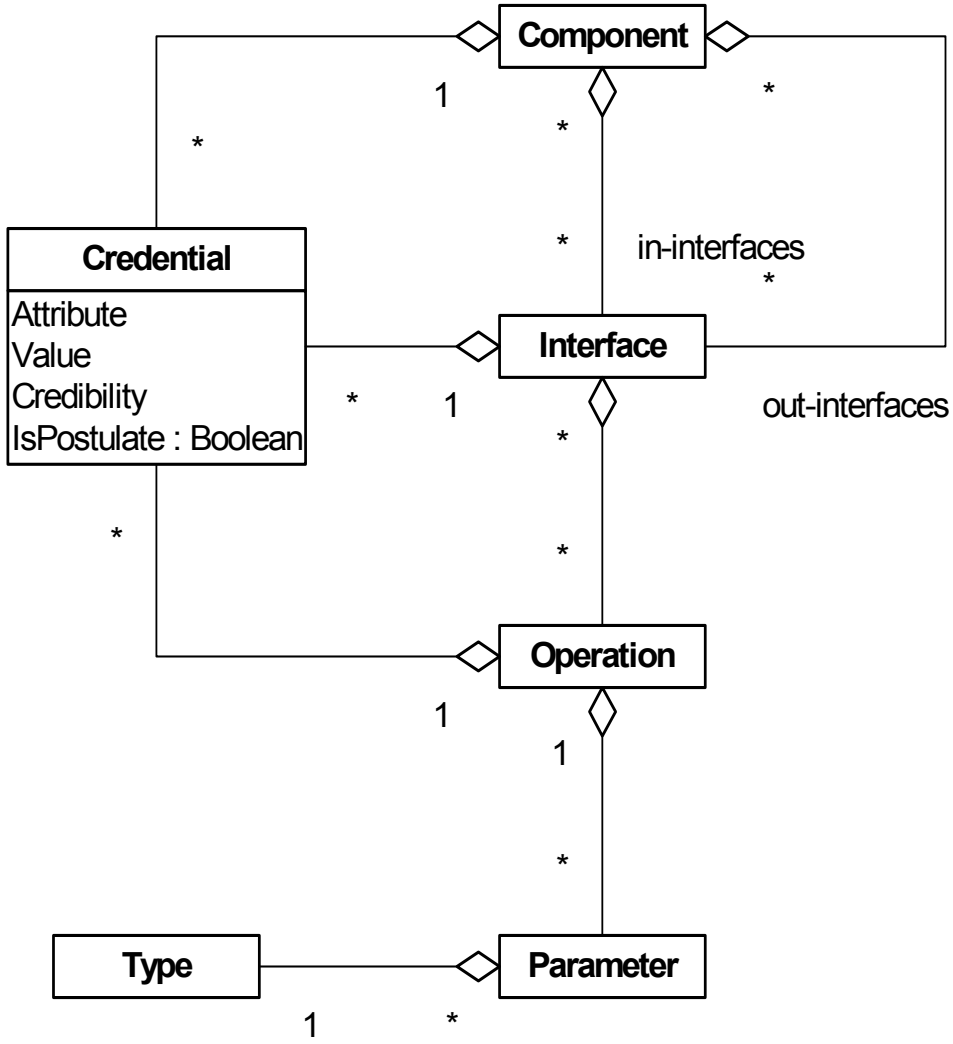
- Relation between system and component properties
- Ability to predict the system properties from the component properties
  - What type of system properties can be predict from component properties?
  - What types of analysis  techniques can be used?
  - How are the component properties specified, measured and certificated?

# Extrafunctional properties specifications

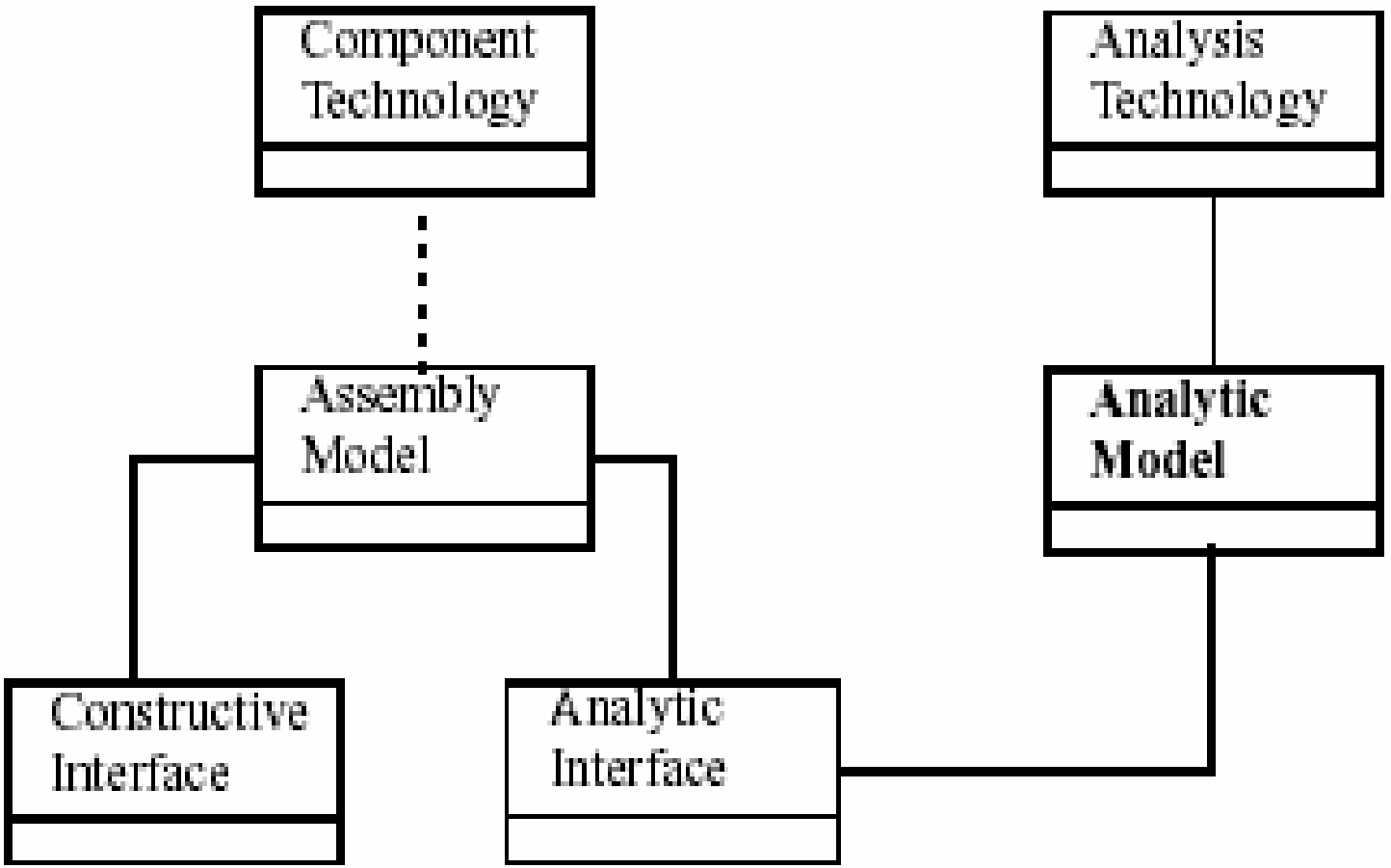
Credentials (Mary Shaw)

- 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

# Extra-functional Properties



# Generalization of a component model



# How much is CBSE attractive for different domains?

- Advantages from a business point of view:
  - Shorter time-to-market, lower development and maintenance costs
- Advantages from technical and engineering point of view
  - Increased understability of (complex) systems
  - Increased the usability, interoperability, flexibility, adaptability, dependability…
- Advantages from strategic point of view of a society
  - Increasing software market, generation of new companies
- CB-approach has been successful in many application domains:
  - Web- and internet-based applications
  - Desktop and office applications, Graphical tools, GUI-based applications
  - In certain segments of telecommunication, consumer electronics…

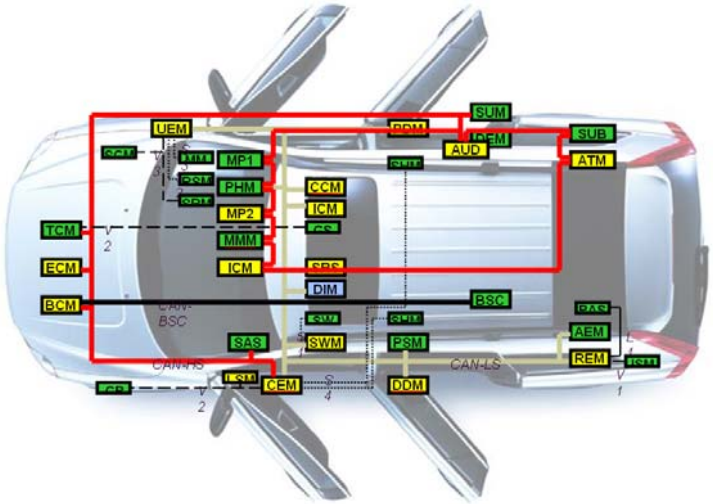
# Do existing component technologies meet the requirements of different domains?

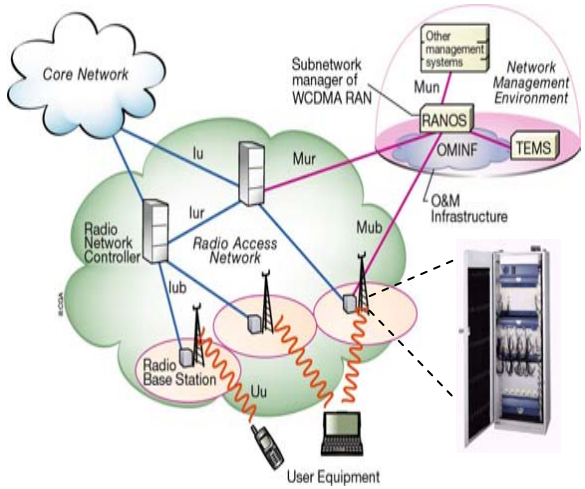
- Widely-used component models (Microsoft COM/DCOM and .NET, Sun EJB, OMG CCB,…)
  - Focus on functionality, flexibility, run-time adaptability, simpler development and maintenance
  - Do not consider non-functional requirements
    - Timing properties (performance), resource consumptions
    - Reliability, availability, quality of services…

***Important questions for CBSE feasibility:***

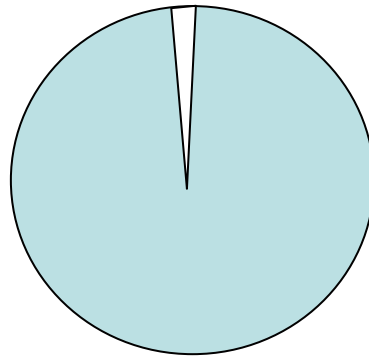
- Which are the primary requirements in different domains?
- Can CBSE provide solutions that meet these requirements?

# What are embedded systems?



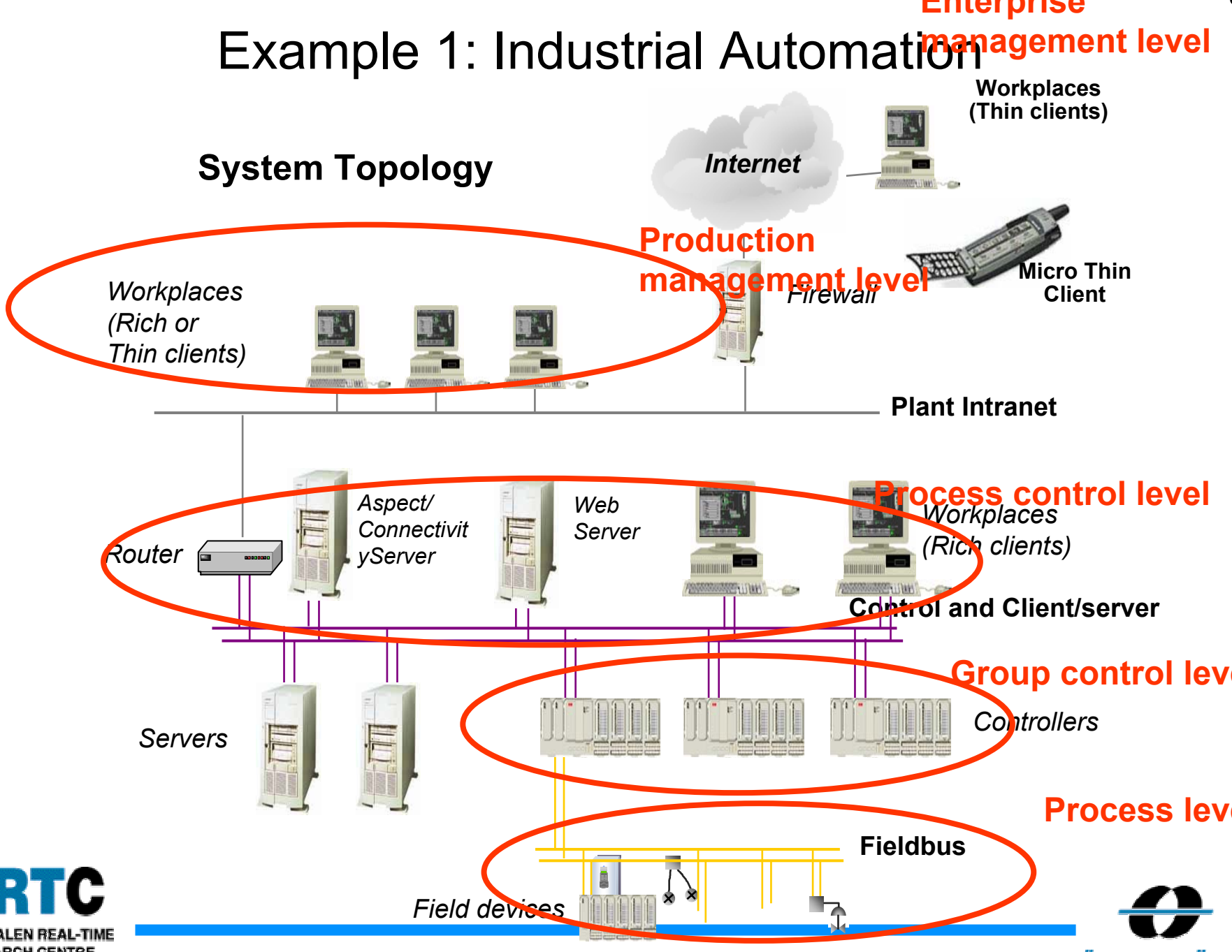




*An Embedded Computer System:*
*A computer system that is part of a larger system and performs some of the requirements of that system. (IEEE, 1992).*

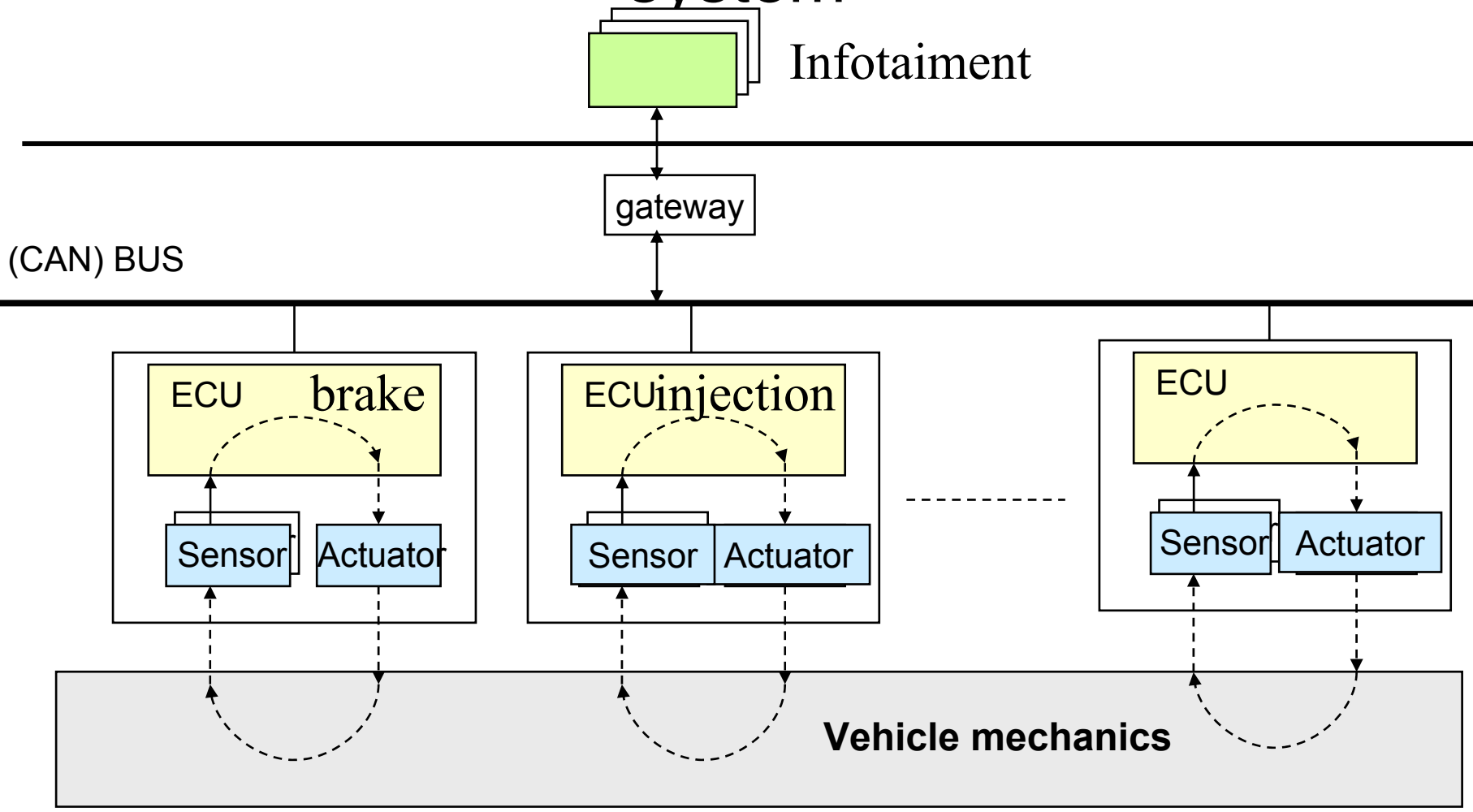


99% of computer systems are embedded systems (DARPA 2000)

# Example 1: Industrial Automation

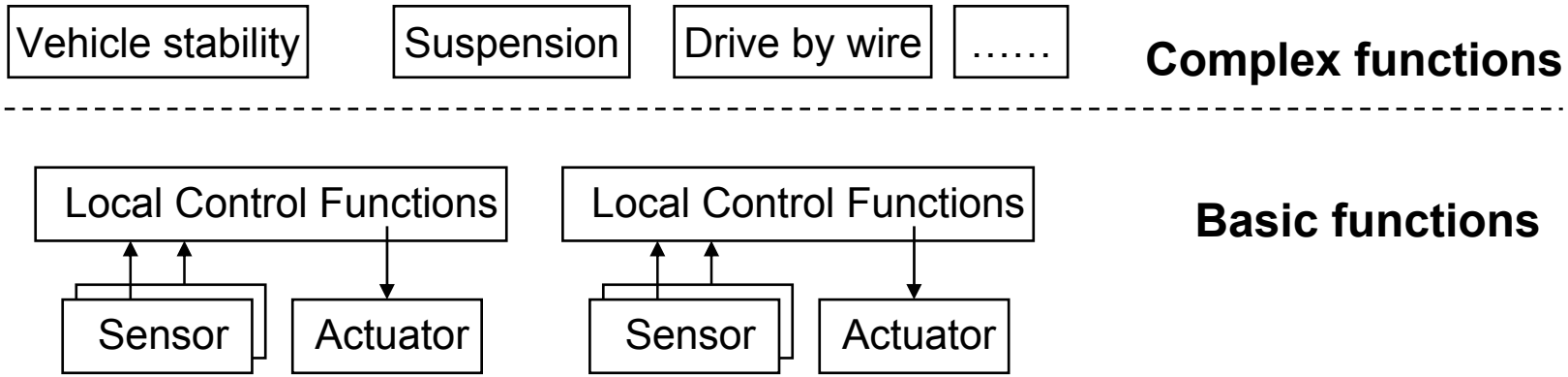


# Example 2: The architecture of a car control system



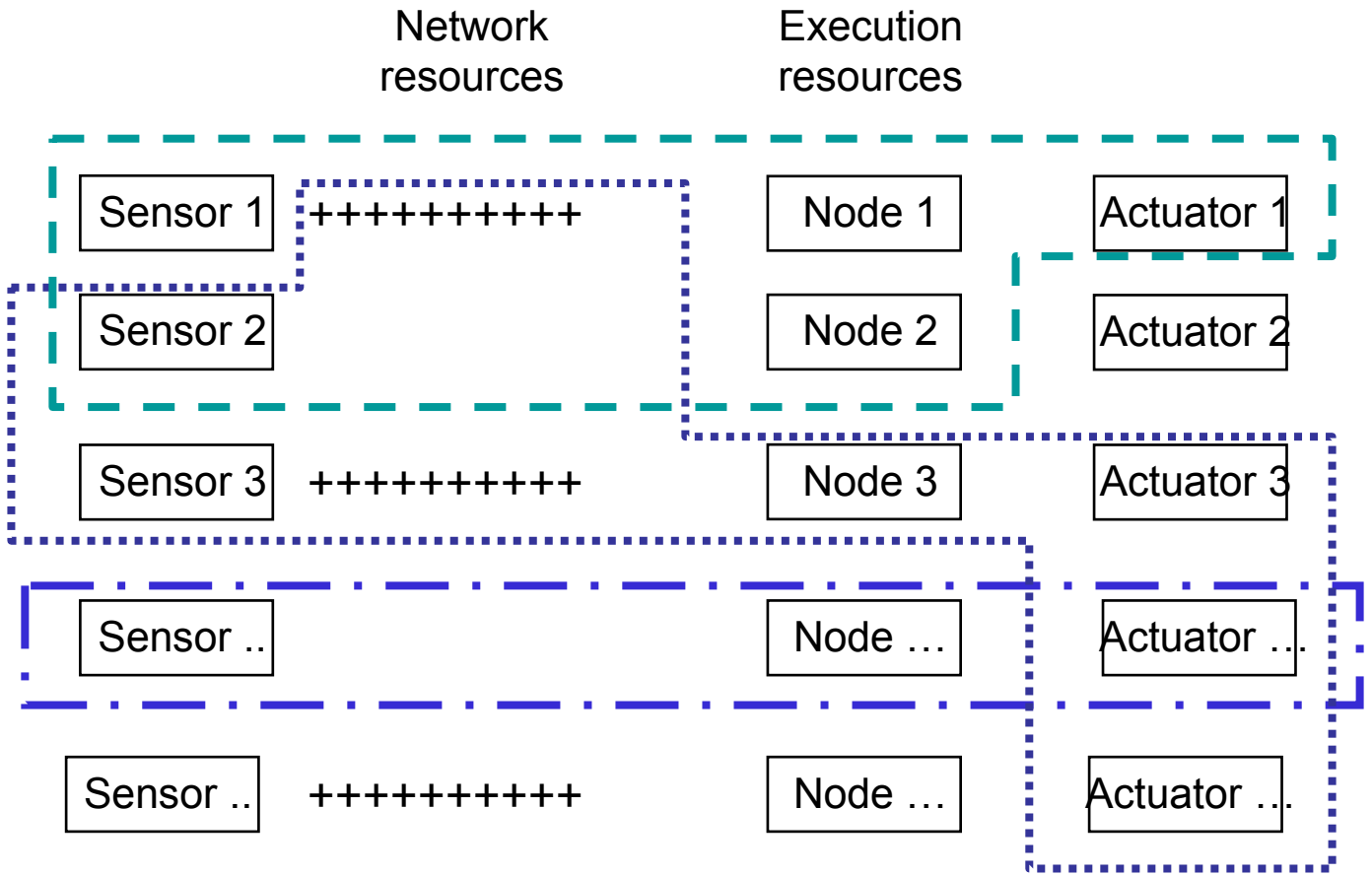

ECU – Electronic Control Unit

# The architectural design challenge



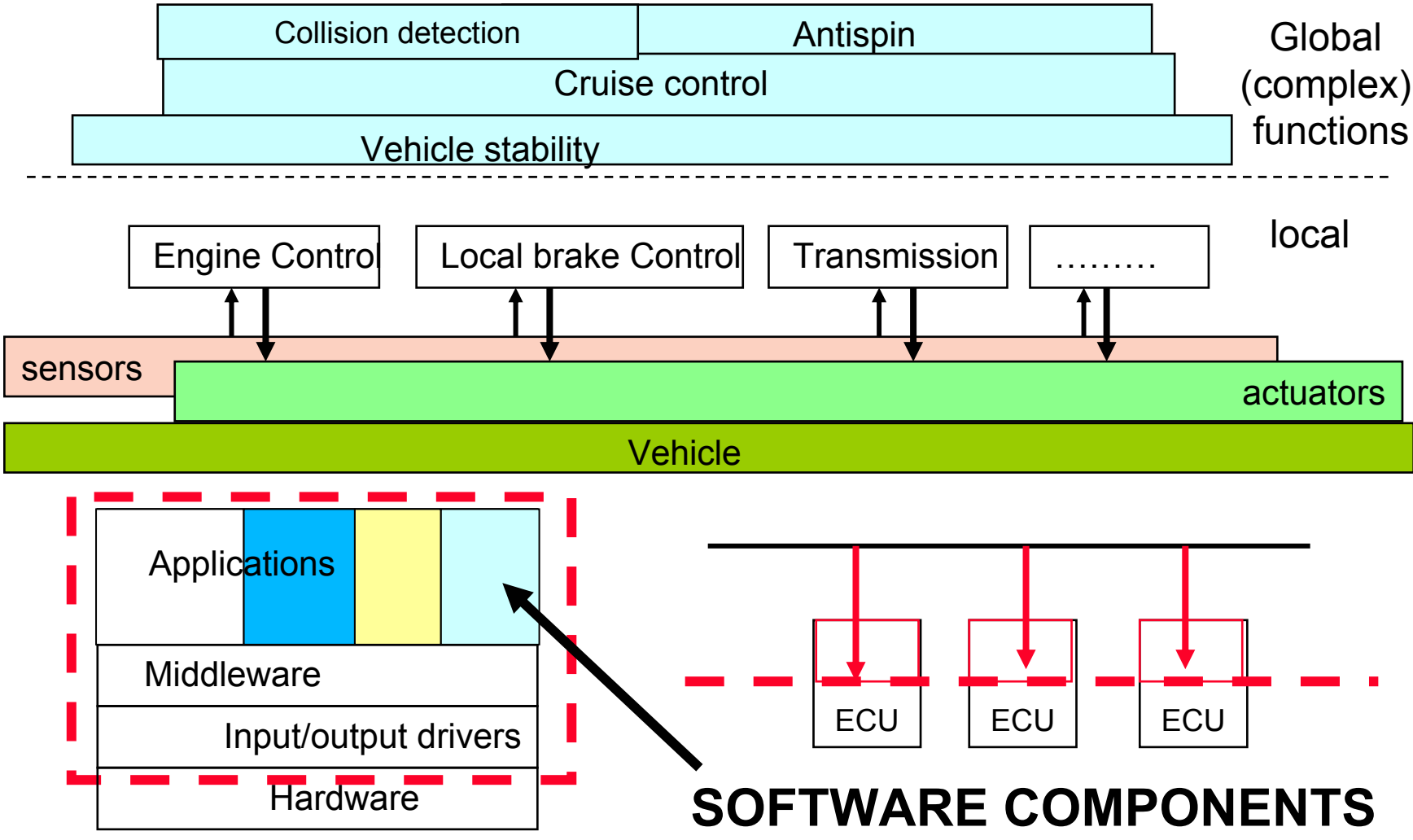

**How to implement complex functions based on local control functions?**

# Problem: resource sharing




**Can functions of different criticality be allowed to share resources?**

# Challenge – open and dependable platform



# Specific requirements of embedded systems

- Real-time requirements
- Resource consumption
  - CPU, Memory, Power, Physical space
- Dependability
  - Safety, reliability, availability
- Life-cycle properties (long life systems)
  - Maintainability, expandability
  - Portability
- Increasing interoperability

# Basic concepts for Component-based Embedded Systems

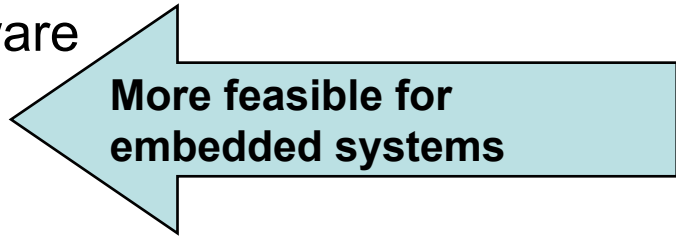
- Main concern
  - Predictability of different properties (on account of flexibility)

- Difference between small and large embedded systems

# Unit of composition and independent deployment

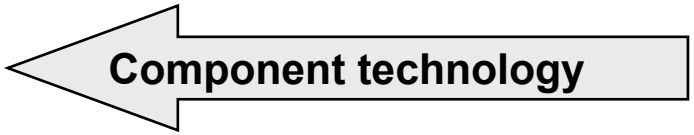
- Run-time composition
  - Component lifecycle,
  - Run-time environment,
  - Dynamic composition (late binding)


- Configuration composition
  - Capable of generating monolithic firmware from component-based design,
  - Optimization
    - Re-configuration of the components
    - Direct references

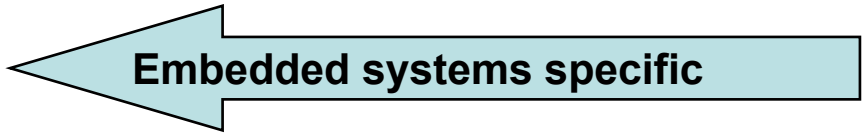


# Explicit context dependencies

- Other components and interfaces
  - required & provided interfaces
  - (Contractual-based interfaces)
  - Set of interfaces

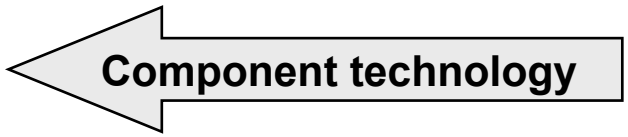



- Run-time environment
  - CPU,
  - RTOS,
  - Resource constraints
  - Component implementation language

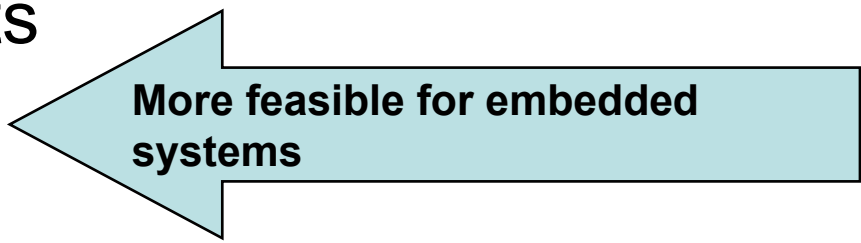


# Component granularity

- Coarse-grained components
  - "Bags" with many (partially unused) functions
  - Not resource-usage aware
  - Often distributed components



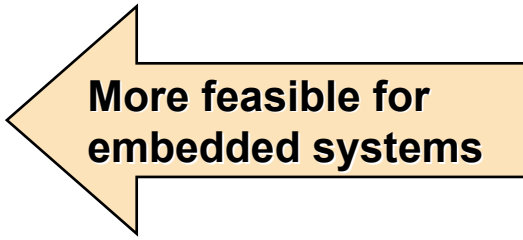

- Fine-grained components
  - unneeded functionality removed,
  - Scarcer uses of resources



# Reuse

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

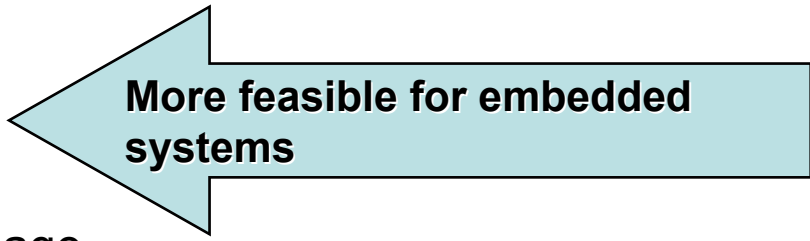

- White-box reuse
  - From composition environment point of view
- Gray-box reuse (glass-box)
  - If clear conventions for knowledge about implementation are introduced



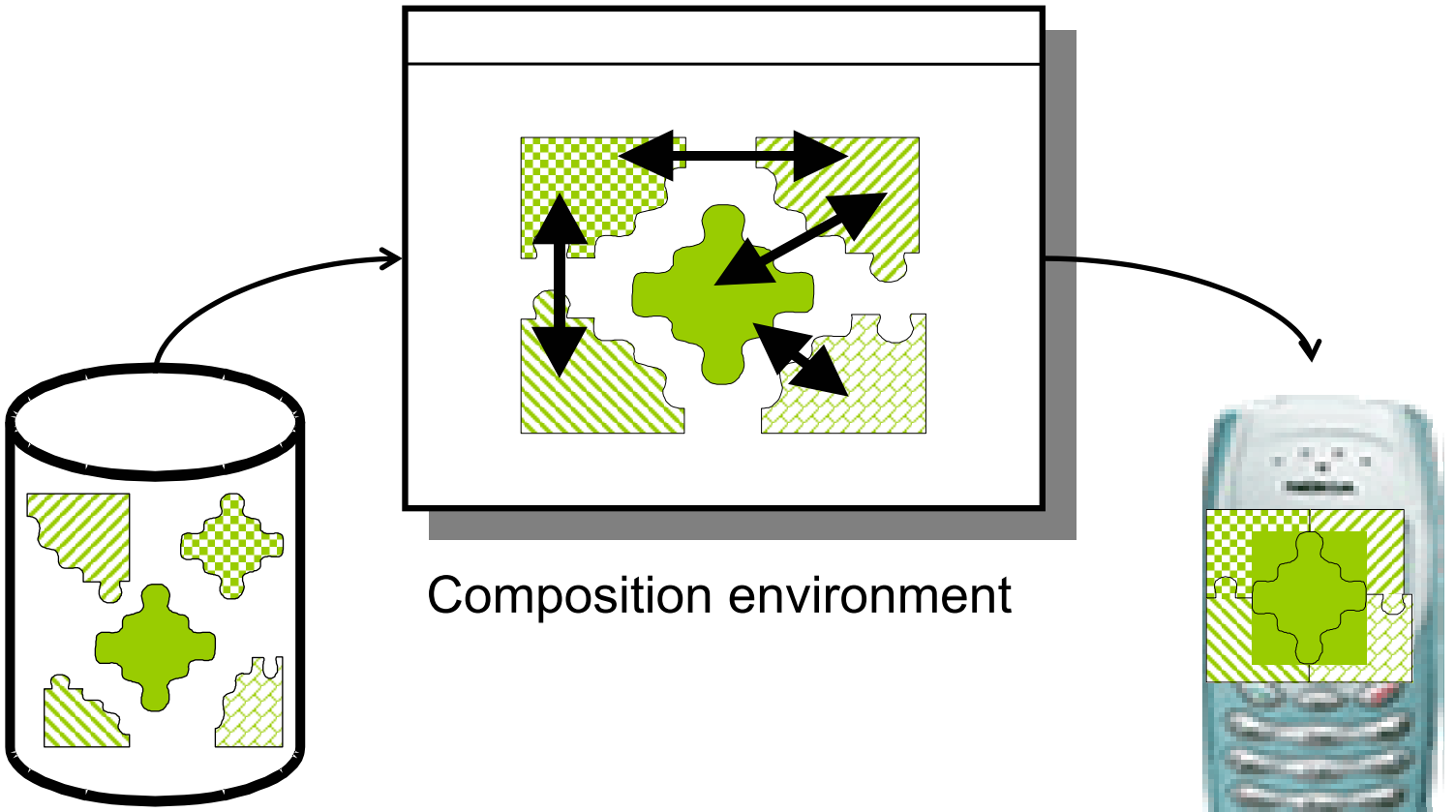
# Portability, Platform independence

- Binary independence


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

# However…


- The day after tomorrow…
  - Requirements on flexible upgrading
    - Part of a system
    - Updating software components
    - Separation of software from hardware

  Binary standards will become important

# Component-based approach for LARGE embedded systems

- the resource constraints are not the primary concerns.
- The complexity and interoperability important
- Minimizing the development costs

- *For this reason general-purpose component technologies are of more interest than in a case for small systems.*

# Widely-used component models and embedded systems

- Direct use of component models
  - CORBA (telecommunication)
  - COM/DCOM, .NET – process industry
- Improved component-models (with added functionalities)
  - OPC (OLE process control Foundation)
- Restricted (use of) component-models to achieve predictability
  - Using only specification (IDL) , no multiple interface, etc.

# The needs and priorities

- Need for component models and frameworks for embedded systems.
  - the run-time platform must provide certain services, which however must use only limited resources.
- Obtaining extra-functional properties of components in particular timing and performance properties.
- Component certification
- Platform and vendor independence
- Component noninterference  applications, (in terms of memory protection, resource usage, etc).
- Tool support: The adoption of component technology depends on the development of tool support.
- Component-based platforms

# References

- Conferences and Workshops
  - Component-based Software engineering Symposium:
    - http://www.sei.cmu.edu/pacc/CBSE7/
    - http://www.sei.cmu.edu/pacc/CBSE8/
  - Euromicro conference, CBSE track
    - http://www.idt.mdh.se/ecbse/2004/
    - http://www.idt.mdh.se/ecbse/2005/
  - WCOP International Workshop on Component-Oriented Programming
    - http://research.microsoft.com/~cszypers/events/wcop2004
  - International Working Conference on Component Deployment
    - http://cd04.cs.ucl.ac.uk/