Real-Time Aspects

1. Introduction

Introduction (Outline)

- When is a Computer System "Real-Time"
- Classification of Real-Time Systems
- Temporal Requirements
- What is a "System Architecture"?

What is a «real time» system?

When is a Computer System 'Real-Time'?

A *real-time computer system* is a computer system in which the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time, when these results are produced.

The point in time when a result has to be produced is called a *deadline*.

Deadlines are dictated by the environment of the real-time computer system.

Definitions

If the result has utility even after the deadline, we call the **deadline *soft***.

If the result has no utility after the deadline has passed, the **deadline is called *firm***.

If a catastrophe could result if a strict deadline is missed, the **deadline is called *hard***.

- A real-time computer system that has to meet at least one hard deadline is called a ***hard real-time system***.

The design of hard- and soft real-time are fundamentally different.

Outline

- When is a Computer System "Real-Time"
- Classification of Real-Time Systems
- Temporal Requirements
- What is a "System Architecture"?

Classification of RT Systems

On the basis of the external requirements

Hard Real-Time versus Soft Real Time

Fail-Safe versus Fail-Operational

Classification of RT Systems

On the basis of the implementation

- **Guaranteed Timeliness** versus **Best Effort**
- **Resource Adequacy**
- **Event Triggered** versus **Time Triggered**

Hard Real Time versus Soft Real Time

Characteristic	Hard Real Time	Soft Real Time
Response time	hard	soft
Peak-Load	perform. predictable	degraded
Error Detection	system	user
Safety	critical	non-critical
Redundancy	active	standby
Time Granularity	millisecond	second
Data Files	small/medium	large

Fail-Safe versus Fail-Operational

A system is *fail-safe* if there is a safe state in the environment that can be reached in case of a system failure, e.g., train signaling system.

- In a fail-safe application the system must to have a high *error detection coverage*.
- Fail safeness is a characteristic of the application

A system must be *fail operational*, if the application does not allow to identify a safe state thus such systems in case of a failure must continue to be operational e.g., a flight control system aboard an airplane.

- In fail-operational applications the computer system has to provide a minimum level of service, even after the *occurrence of a fault*.

Classification of RT Systems

On the basis of the external requirements

- Hard Real-Time versus Soft Real Time
- Fail-Safe versus Fail-operational

On the basis of the implementation

- Guaranteed Timeliness versus Best Effort
- Resource Adequacy
- Event Triggered versus Time Triggered

Guaranteed Timeliness versus Best Effort

A system implementation provides *guaranteed timeliness* if (within the specified load- and fault-hypothesis) the temporal correctness can be substantiated by analytical arguments.

A system implementation is *best effort*, if such an analytical argument for the temporal correctness can not be made.

- The temporal verification of best effort systems relies on **probabilistic arguments**, even within the specified load- and fault hypothesis.

Hard real-time systems should be based on guaranteed timeliness.

Resource Adequacy

If a system has to provide guaranteed timeliness, there must be sufficient computational resources to handle the specified peak load and fault scenario.

In the past, there have been many applications where resource adequacy has been considered *too expensive*.

The decreasing cost of hardware makes the implementation of resource adequate designs economically viable.

- In hard real-time applications, *there is no alternative to resource adequate designs*.

Predictability in Rare Event Situations

A *rare event* is an important event that occurs very infrequently during the lifetime of a system, e.g., the rupture of a pipe in a nuclear reactor

A rare event can give rise to many correlated service requests (e.g., an alarm shower)

In a number of applications, the utility of a system depends on the predictable performance in rare event scenarios, e.g. flight control system

In most cases, workload testing
will not cover the rare event scenario

State and Event

A *state* is a condition that persists for an interval of real time, i.e., along a section of the timeline

An *event* is an occurrence at an instant.

State information informs about the attributes of states at the point of observation (itself an event).

Event information informs about the difference in the attributes of the states immediately before and after the occurrence of the event and an estimation of the point in time of event occurrence

- Only the consequences of an event can be observed.

Time Triggered vs. EventTriggered

A Real-Time system is *Event Triggered* (ET) if the control signals are derived solely from the occurrence of events, e.g.,

- termination of a task
- reception of a message
- an external interrupt

A Real-Time system is *Time Triggered* (TT) if the control signals, such as

- sending and receiving of messages
- recognition of an external state change

are derived solely from the progression of a (global) notion of time.

Outline

- When is a Computer System "Real-Time"
- Classification of Real-Time Systems
- Temporal Requirements
- What is a "System Architecture"?
- Example of TTA

Temporal Requirements

Temporal accuracy of real-time data:

The data elements that are displayed to the operator must be *temporally accurate*.

Maximum response time:

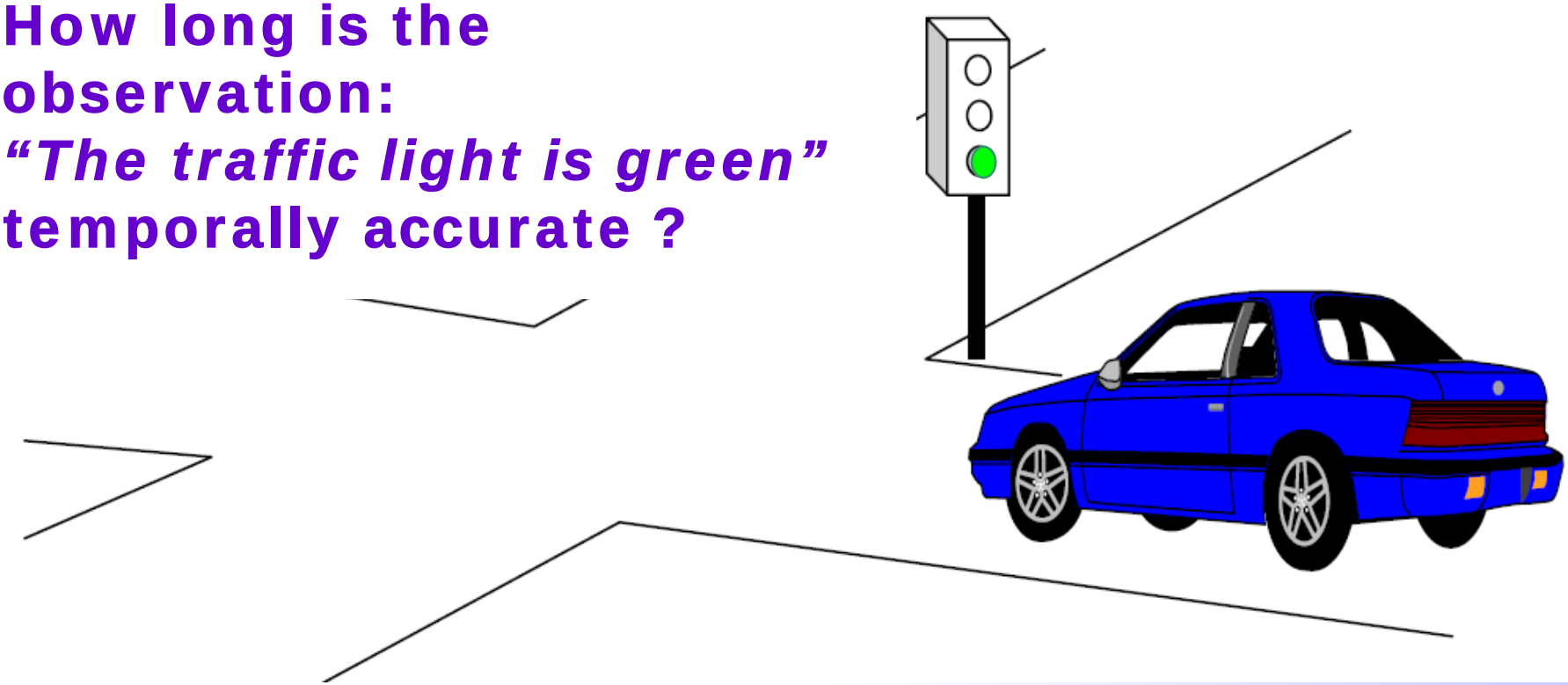
The maximum real-time interval between a *stimulus* and the *response* must be known and bounded.

Predictability:

The temporal behavior must be predictable, even in a rare event scenario.

Validity of Real-Time Information

How long is the observation:
"The traffic light is green"
temporally accurate ?



Temporal parameters are associated with real-time data.

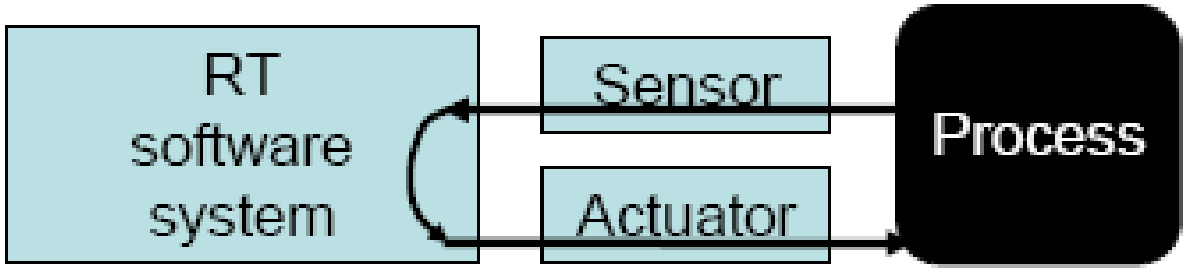
Interaction of RT system with the environment

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

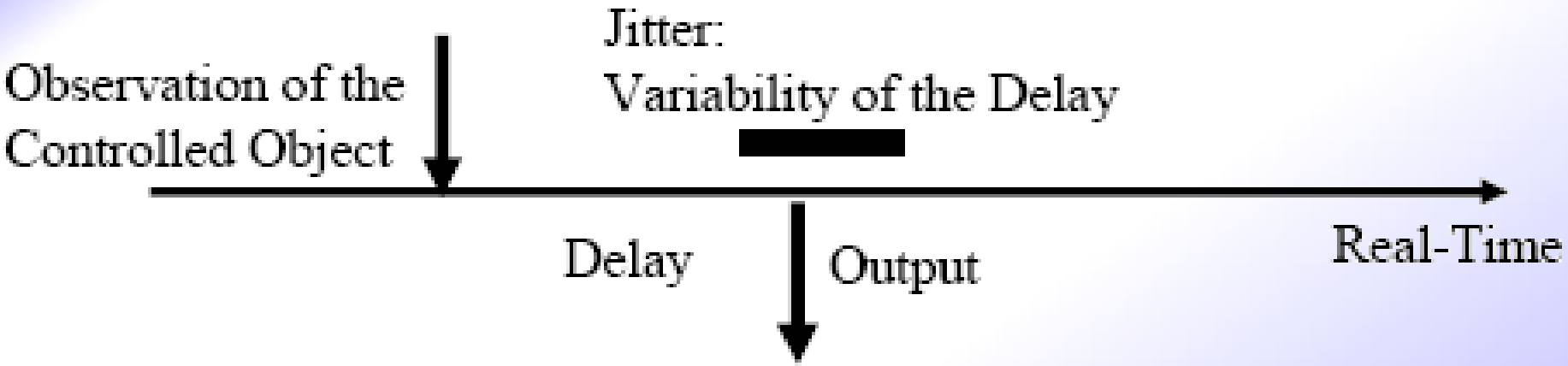
- Examples: thermometer, microphone, video camera

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

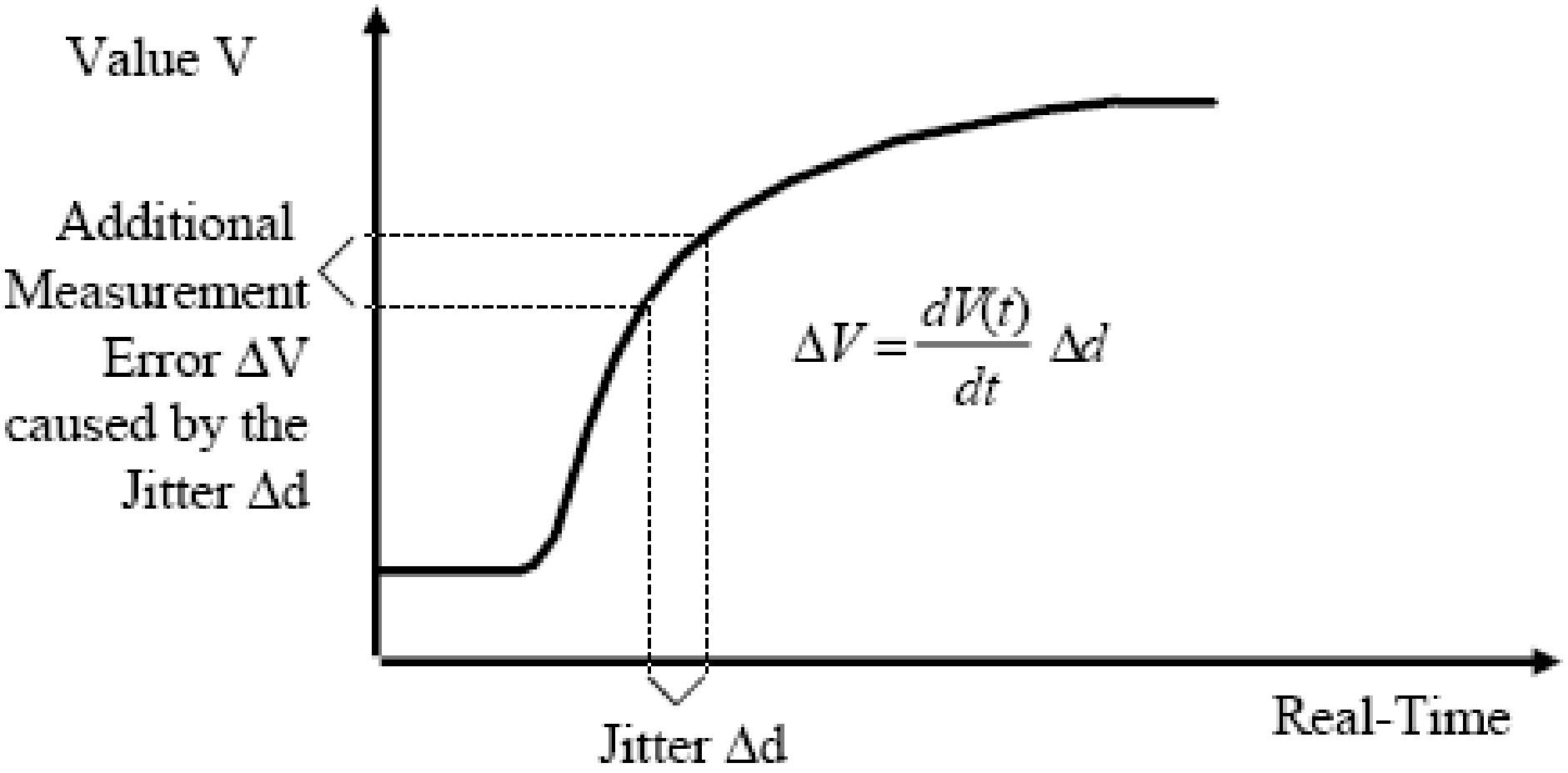
- Example: motors, pumps, machines…



Jitter at the Application Level



The Effect of Jitter: Measurement Error



Jitter in Control Loops causes a degradation of control quality.

Protocol Execution Time in ET Systems qui

The execution time of an **event-triggered** protocol between two tasks depends on:

- the scheduling decisions by the operating system of the sender
- the buffer management of the sender
- the data link protocol
- the media access strategy
- the buffer management at the receiver
- the scheduling strategy at the receiver

Outline

- When is a Computer System "Real-Time"
- Classification of Real-Time Systems
- Temporal Requirements
- What is a "System Architecture"?
- Example of TTA

Need for a "Sound" System Architecture for Real-Time Distributed Systems

For (critical) *hard real-time* systems we need to base the development of our system on a "sound" technical system architecture

A *technical system architecture* is a framework for the construction of a system that constrains an implementation in such a way that the ensuing system is

- understandable,
- maintainable,
- extensible, and
- can be built cost-effectively

Technical System Architecture (II)

Architectural style:

- An architecture must provide rules and guidelines for the partitioning of a system into subsystems and for the design of the interactions among the subsystems.

Composability:

- An architecture must provide a framework for the systematic construction of a system out of subsystems (components).

Elegance:

- An architecture must *constrain* an implementation in such a way that the system is understandable, maintainable, extensible, and can be built cost-effectively--in other words, it is *elegant*.
  - ➢ Architecture Design is Interface Design

Don't Touch it Again

Consider a system that is developed **without** a documented technical architecture by a group of experts :

- The system can achieve a high level of functionality within a relatively short time because an *architecture exists in the mind of the experts.*
- As soon as the original developers leave, nobody really understands the architectural style and the internals of the system anymore.
- The system is very difficult to maintain - nobody wants to touch it.

The heart of the US Air Traffic Control system is an IBM 360/65 computer that has been installed 30 years ago!

Dead End Road

The approach to "trial and error" system building without an underlying architecture is noteworthy a dead-end road.

Fundamental changes are happening/have happened:

- Systems on Chip (SOC)
- Inherent distribution and Cloud
- Mass market (embedded) applications (e.g., automotive)
- Internet appliances (Door lock on the Web?)

Real-time systems: What are key requirements in the future?

Grand Challenge

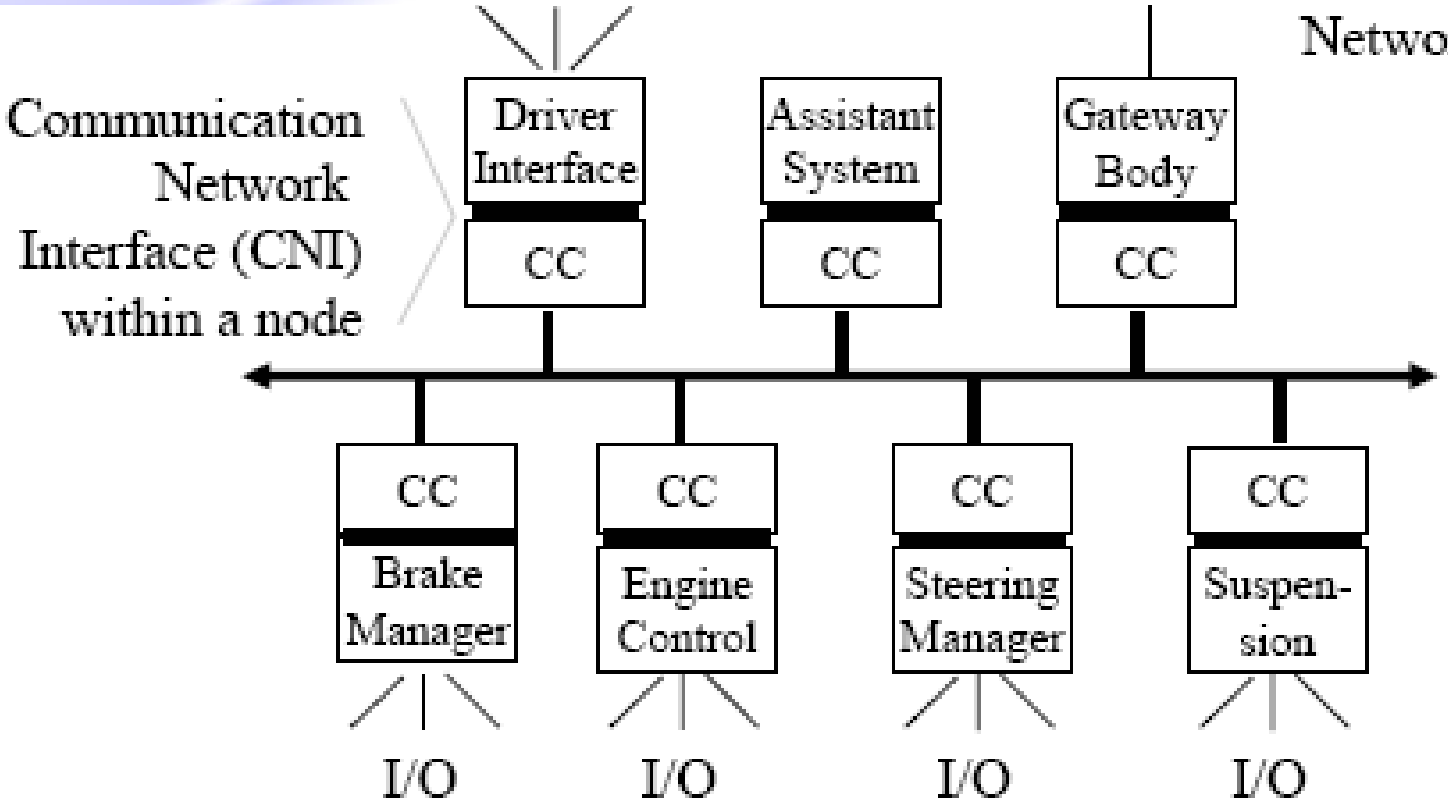
The grand challenge in the design of large distributed real-time systems is the management of the ever-growing complexity

- Partition the system into nearly autonomous subsystems that can be understood, developed and tested independently
- Separate the interactions among the subsystems (system level) from the functions within the subsystems (subsystem level).

What is needed is a *distributed architecture* for embedded realtime systems that supports a two level design methodology

- system level design and the
- subsystem level design

Example of an Integrated Drive-by-Wire System - qui



Composability

An architecture is said to be *composable* with respect to property P_{IK} of a subsystem if this property also holds at the system level after the integration.

Examples of such properties for which we want composability:

- Timeliness
- Testability

Architecture based approach to RT system design

An architecture-based approach to real-time system design must support:

- Composability
  to build systems constructively out of prevalidated components.
- Two-level design methodology
  to be able to separate architecture design from component design.
- Generic fault-tolerance
  to implement fault-tolerance without any change in the application software.
- Flexible configuration
  to support the reuse of existing components

Architecture Design *is* Interface Design

A good interface within a distributed real-time system

- must *precisely* be specified in the value domain and in the temporal domain,
- must *hide* the irrelevant details,
- must lead to *minimal coupling* between the interfacing subsystems,
- must conforms to the established architectural style and thus introduces *structure* into a system.