# Chapter 4
# Modeling Real-Time Systems

## Overview

- Introduction of a conceptual model for real-time systems
- Tasks, nodes, fault-tolerant units, clusters
- Simple and complex tasks
- Interface placement and interface layout
- Temporal control and logical control
- The history state

## 4.1 Appropriate Abstractions

A model is a reduced representation of the world.
A conceptual model is a set of well-defined concepts and their interrelationships to obtain an abstracted representation of a real-world entity.

We have to consider two important assumptions when designing a model for a fault-tolerant real-time system:

- Load hypothesis: load offered to a computer system is below a maximum or peak load
- Fault hypothesis: a statement about the assumptions that relate to the type and frequency of faults the computer system is supposed to handle.

Statements on the response time of a computer system can only be made under the load hypotheses.

When designing a fault-tolerant system, only conditions described by the fault hypothesis will guarantee a fault-tolerant behavior. Outside this hypothesis, the system can fail.

### Notion of Physical Time:

To a real-time system model, the notion of physical time is of central importance. In our model, we assume that there is a omniscient external observer with a precise reference clock z, and that all real-time clocks in the nodes are synchronized within a precision $\Pi$.

### Duration of Actions:

The execution of a statement constitutes an action.

We consider the following temporal quantities to describe temporal behavior of an action a:

1. Actual duration (actual execution time) $d_{act}(a, x)$: the number of time units of reference clock z that occur between start of action with input parameters x, and the termination of action a.
2. Minimal duration $d_{\min}(a)$: smallest time to execute action $a$, for all possible input parameters $x$.
3. Worst-case execution time (WCET) $d_{wcet}(a)$: the maximum duration of action a for all conceivable input parameters.
4. Jitter: the difference between worst-case execution time $d_{wcet}(a)$ and minimal execution time $d_{\min}(a)$.

### Frequency of Activations:

The maximum number of activations of an action per unit of time we call the frequency of activations.

Every processor (or computational resource) has a finite capacity. A processor can meet its temporal obligations only if the frequency and temporal distribution of the activations are controlled.

### What is not Part of the Model:

- Representation of values (e.g. 4-20 mA, data format, etc.) Representation differences can be hidden behind gateways.
- Details of data transformation (e.g. control algorithms, representation transformations, internal program logic).

### 4.2 The Structural Elements

A distributed fault-tolerant real-time system can be decomposed into a set of communication clusters. A computational cluster can be partitioned into a set of fault-tolerant units (FTU), connected by a real-time communication network. Each FTU consists of one or more node computers. Within a node computer, a set of concurrently executing tasks run jobs, i.e. perform the intended functions (see figure on board).

### Jobs and Tasks

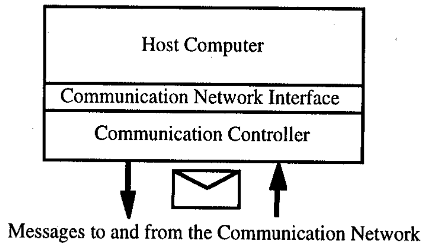
A task is a sequential program. The execution of one or more tasks to perform some function we call a job.

The time interval between the start of a job and its termination, given an input data set x, is called the actual duration $d_{act}(job, x)$ of the job on a given target machine.
A task that does not have an internal state at its point of invocation is called a stateless task. Otherwise it is called a task with state.

**Simple Task (S-task):** a task with no synchronisation point within, e.g. no semaphore “wait” operation, mutexes or other constructs that could make the task wait or block. Execution time can be determined in isolation, i.e. without knowledge of behaviour of other tasks.

**Complex Task (C-task):** a task with blocking synchronization statements, e.g. semaphore “wait” operation. Execution time is a global issue, it can depend on progress of other tasks within the node.
**Node**

A node is a self-contained computer with its own hardware and software.
A node accepts input messages and produces the intended and timely output messages via the communication network interface (CNI).


Host Computer
Communication Network Interface
Communication Controller
Messages to and from the Communication Network

Data structures on the node can be categorized into initialization state (i-state) and history state (h-state).

The i-state is a static data structure comprising the program code and initialization data and can be stored in ROM.

The h-state is the dynamic data structure and is stored in RAM.

### Fault-Tolerant Unit (FTU)

A fault tolerant unit consists of a set of replicated nodes that are intended to produce replica determinate result messages.

In case one of the nodes of an FTU produces erroneous results, a judgement mechanism detects this situation, and ensures that only correct results are delivered to the clients of the FTU (e.g. voter which takes majority of results).

From a logical point of view, an FTU can be considered as a single node.

### Computational Cluster

A computational cluster comprises a set of FTUs that cooperate to perform the intended fault-tolerant service.

Interfaces between a cluster and its environment (controlled object, operator, other computational clusters) are formed by the gateway nodes of the cluster.

Clusters can be interconnected by gateway nodes in form of a mesh network.

## 4.3 Interfaces

### 4.3 Interfaces

System architecture design is primarily interface design.
An interface between two subsystems of a real-time system is characterized by

- The control properties (control signals crossing interface, what do they do)
- The temporal properties (temporal constraints to be satisfied by control signals and data crossing the interface)
- The functional intent (intended functions of the interface partner)
- The data properties (structure and semantics of data elements crossing the interface)

In case events can occur any time, we speak of a dense time base.
In case events are permitted to occur only in certain time intervals $\pi$, we speak of a sparse time base.
In a dense time base it can be difficult to establish temporal order.

Example: The functional intent of a node in an engine controller is to guarantee the car conforms to environmental standards. As the standards change, e.g. new laws are passed, the function of the node may have to change, but not its functional intent.

The functional intent is thus at a higher level of abstraction than the function.

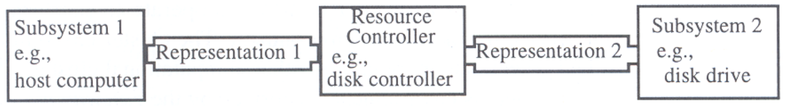
The functional intent can be related to a “goal” in requirements engineering.

## 4.3 Interfaces

### Resource Controllers

In many cases, the interfacing partners use different syntactic structures and incompatible coding schemes to represent information that must cross the interface.

An intelligent interface component must be placed between the interface partners to transform the different representations. This we call a resource controller.

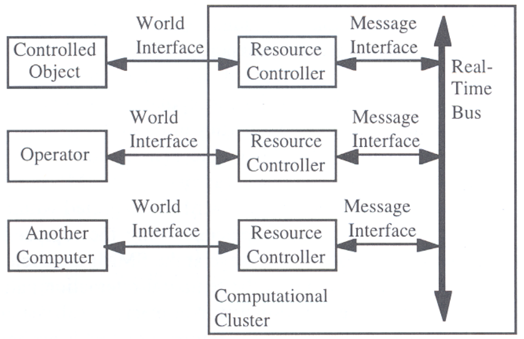


Resource controllers act as gateways between two different subsystems with different representations.

### World and Message Interfaces

We distinguish between real world interfaces, and message interfaces.
Example: specific man-machine interface (SMMI) would be a concrete world interface, with touchpad, screen, buttons. A generalized man-machine interface (GMMI or abstract message interface) would just consider the messages that cross the interface, and their temporal properties.

## 4.3 Interfaces



### Standardized Message Interfaces

To improve compatibility between systems designed by different manufacturers, some international standard organizations have attempted to standardize message interfaces. In automotive systems, the interfaces are typically bus interfaces.

## 4.3 Interfaces

### Automotive bus systems (OSI level 0 to 2)

| **Type of bus** | **Application** | **European Standards** | **US Standards** |
|---|---|---|---|
| *Character based (UART)* | | | |
| K/L-Line | Diagnostics | ISO 9141 | |
| SAE J1708 | Diagnostics, Class A On Board | | SAE J1708 (truck and bus, 9.6 kBit/s) |
| LIN | Class A On Board | Manufacturer syndicate, 20 kBit/s | |
| *PWM based* | | | |
| SAE J1850 | Diagnostics, Class A/B On Board | | SAE J1850 (PWM Ford, VPWM GM), 10.4 and 41.6 kBit/s |
| *Bit stream based* | | | |
| CAN | Class B/C On Board | ISO 11898 1-3<br>Bosch CAN 2.0 A,B<br>47.6 ... 500 kBit/s<br><br>ISO 11992<br>CAN for trucks and trailers<br><br>ISO 11783 ISOBUS<br>CAN for agricultural vehicles | SAE J2284<br>(passenger cars)<br>500 kBit/s<br><br>SAE J1939<br>(truck and bus)<br>250 kBit/s |
| TTCAN | Class C(+) On Board | ISO 11898-4<br>Bosch, 1 MBit/s | |
| FlexRay | Class C+ On Board | Manufacturer syndicate, 10 MBit/s | |
| TTP | Class C+ On Board | Manufacturer syndicate | |
| MOST | Multimedia | Manufacturer syndicate, 25 MBit/s | |

### Automotive transport protocols (OSI level 4)

## 4.3 Interfaces

| Transport Protocol | Application | European Standards | US Standards |
|---|---|---|---|
| ISO TP | CAN busses | ISO 15765-2 | |
| SAE J1939 | CAN busses | | SAE J1939/21 |
| TP 1.6 TP 2.0 | CAN busses | Manufacturer standard VW/Audi/Seat/Skoda Base is OSEK COM 1.0 | |

### Automotive application protocols (OSI level 7)

| Protocol | Application | European Standards | US Standards |
|---|---|---|---|
| ISO 9141-CARB | Diagnostics US OBD | ISO 9141-2 Outdated US diagnostics interface | |
| KWP 2000 Keyword protocol | Diagnostics (general and OBD) | ISO 14230 KWP 2000 diagnostics on K-Line ISO 15765 KWP 2000 Diagnostics on CAN | |
| UDS Unified Diagnostic Services | Diagnostics (general and OBD) | ISO 14229 UDS Unified Diagnostic Services | |
| OBD | Diagnostics US OBD European OBD | ISO 15031 Identical to US standards | SAE J1930, J1962, J1978, J1979, J2012, J2186 |
| CCP and XCP CAN and Extended Calibration Protocol | Application | ASAM MCD 1 ASAM association | |

## 4.3 Interfaces

### Temporal Obligations of Clients and Servers

In the client-server model, a request (message) from a client to a server causes a response from the server at a later time. Three temporal parameters characterize such a client-server interaction:

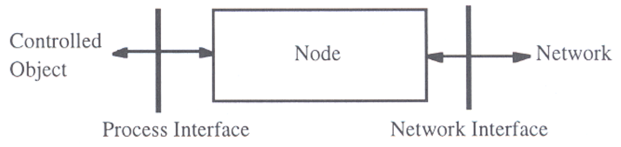
1. The maximum response time, RESP, that is expected by the client
2. The worst-case execution time, WCET, of the server
3. The minimum time, MINT, between two successive requests by the client

The WCET is in the sphere of control of the server, and the MINT is in the sphere of control of the client.

In a hard real-time environment the implementation must guarantee

$$\text{WCET} < \text{RESP}$$

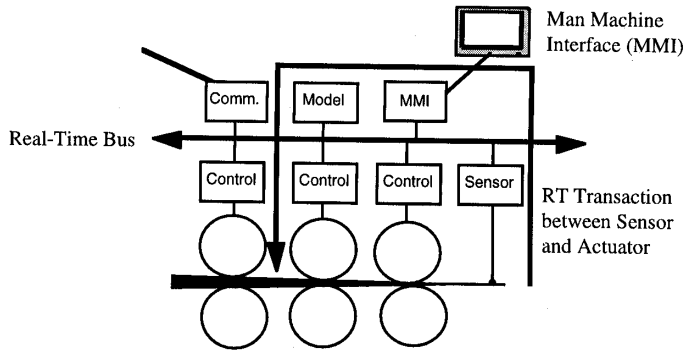
as long as the client respects the specified MINT (e.g. no interrupt overload).



## 4.4 Temporal Control versus Logical Control

### 4.4 Temporal Control versus Logical Control

Example rolling mill with three drive controllers and associated controller nodes and pressure sensors:


Man Machine Interface (MMI)
Comm.
Model
MMI
Real-Time Bus
Control
Control
Control
Sensor
RT Transaction between Sensor and Actuator

```
when ((p1 < p2) ^ (p2 < p3))
    then everything ok
    else raise pressure alarm;
```

But...

## 4.4 Temporal Control versus Logical Control

1. What is maximum tolerable time difference between occurrence of the alarm condition in the controlled object and the triggering of the alarm at the MMI?
2. What are the maximum tolerable time differences between the three pressure measurements in the different control nodes?
3. When do we have to activate the pressure measurement tasks at the drive controller nodes?
4. When do we have to activate the alarm monitoring task at the MMI node?

The when statement intermingles two aspects: the point in time when the alarm must be raised, and the logical condition that needs to be met to raise the alarm.

Logical control is concerned with the control flow within a task, in order to achieve the desired data transformation.

Temporal control is concerned with determining the points in time when a task must be activated, or when a task must be blocked, because some conditions outside the task are not satisfied at a particular moment.

For an S-task, the only temporal control issue is when to activate the task.

A C-task combines issues of logical control and temporal control, for example with a semaphore “wait” statement to delay program execution until a temporal condition outside the task is satisfied.

Different approach: Synchronous languages like LUSTRE, ESTEREL. Task finishes computation immediately, output is synchronous with input.

## 4.4 Temporal Control versus Logical Control

### Event-triggered versus Time-triggered

A system where all control signals are derived from event triggers is called an event-triggered (ET) system. Event examples: button pushed, activation of limit switch, arrival of new message at a node, completion of task within node.

A system where all control signals are derived from time triggers is called a time-triggered (TT) system. A time trigger is a control signal derived from the progression of time, e.g. the clock within a node reaches a preset point in time.

Example: computer system controlling an elevator.

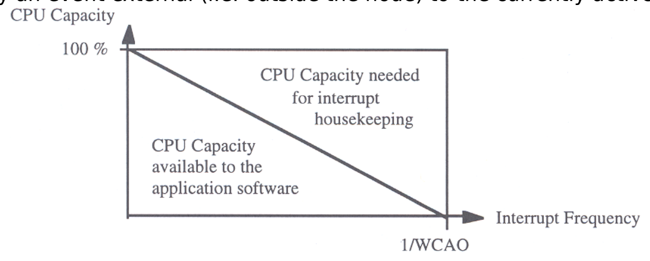
Many real-time systems use time triggers as well as event triggers, but most favour one or the other.

- Advantage event-triggered systems: flexibility
- Advantage time-triggered systems: predictability

## 4.4 Temporal Control versus Logical Control

### Interrupts

In an ET system, the external event triggers are often relayed to the computer system via interrupts. An interrupt is an asynchronous hardware-supported request for a specific task activation caused by an event external (i.e. outside the node) to the currently active computation.



Interrupts cause an overhead (worst case administrative overhead, WCAO), mostly due to the required context switches.

If interrupt frequency is too high, no CPU capacity remains for actual computations. Interrupts are outside the sphere of control of the node, which makes it difficult to handle such overload conditions.

### Trigger Task

## 4.4 Temporal Control versus Logical Control

In a TT system, to observe state changes outside the computer the state of the environment is regularly captured by a trigger task.

The result of a trigger task can be a control signal that activates other tasks.

Only states with a duration greater than the sampling period of the trigger task can be observed reliably. Short lived states (push of button) have to be stored in a memory element within the interface.

The periodic trigger task generates an administrative overhead in a TT system, regardless of any events that may occur. The administrative overhead becomes large if the sampling period of the trigger task gets small (< 1ms). The period of a trigger task must be smaller than the laxity (difference between deadline and execution time) of an RT transaction caused by an event in the environment.

(Illustration).

## 4.5 Worst Case Execution Time

### 4.5 Worst Case Execution Time

Deadlines can only be guaranteed if worst case execution (WCET) times of all application tasks are known a priori.

In addition, the worst case delays caused by administrative functions (e.g. operating system services, context switches, scheduling) need to be known. These delays we call the worst case administrative overhead (WCAO).

### WCET of non-preemptive S-Tasks

WCET depends on

- source code of task
- properties of object code generated by compiler
- characteristics of compiler

**Source code analysis:**

Problem is to find the longest path through the code (critical path). Number of paths increases exponentially with size of program. Annotations provided by the programmer can assist a tool to extract the critical path, by providing additional semantic information.

**Compiler analysis:**

The execution time of source code statements depends heavily on the object code generated by the compiler. The compiler generated object code timing analysis must be related to the source program by means of statement-level annotations.

**Microarchitecture timing analysis:**

## 4.5 Worst Case Execution Time

The execution time of the object code generated by the compiler depends on the microarchitecture of computer that executes it.

For small, simple microprocessors it is possible to predict the time required to execute a single machine language construct by looking it up in the specifications.
For larger microprocessors, with their pipelining and caching mechanism, it is very difficult to predict reliably an upper bound on the execution times of object code. Dependencies among instructions can cause pipeline hazards and cache misses.

The best that has been achieved to date is WCET bound 100% above the measured value.

**WCET of preemptive S-Tasks**

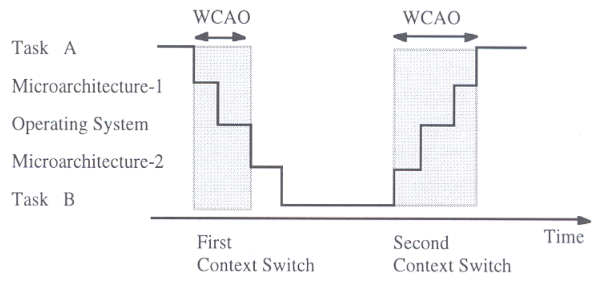
The execution time of an S-task is extended by three terms in case of preemption:

- The WCET of the interrupting task
- The WCET of the operating system for context switching
- The time required for reloading the instruction and data cache after the context switches

The sums of the last two terms we call the worst case administrative overhead (WCAO).

For modern microprocessors, the last component contributes significantly to the WCAO.

## 4.5 Worst Case Execution Time



**WCET of C-Tasks**

The WCET of C-tasks does not only depend on performance of the task itself, but also on thebehaviour of other tasks and the operating system within a node.

WCET analysis of a C-task is thus a global problem involving all interacting tasks within a node. In addition to preemption, intended task dependencies (mutual exclusion, precedence) have to be considered.

## 4.5 Worst Case Execution Time

**State of Practice**

In practice, it is usually not feasible to obtain a tight upper bound on the WCET for any but the most simple system. However, since bounds for WCETs are required in a hard real-time application, the problem is solved by combining some of the following techniques:

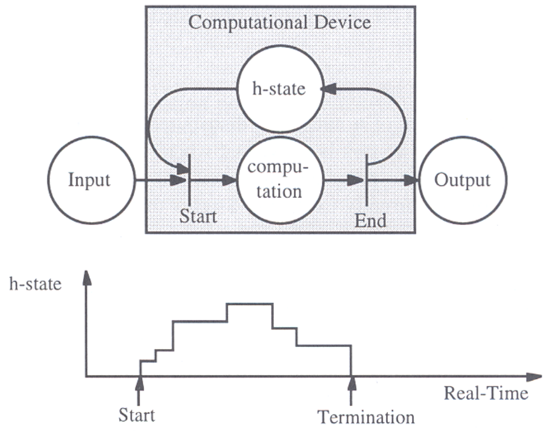
1. Gather experimental data for WCET analysis by measuring a similar actual implementation.
2. Restrict architecture to reduce interactions among tasks and to facilitate the a priori analysis of the control structure. Minimize explicit synchronizations that require context switches.
3. Analyze subproblems (e.g. maximum execution time analysis of the source program) to obtain a set of test cases biased towards the worst case execution time.
4. Do extensive testing of the complete implementation to measure the safety margin between the assumed WCET and actual measured execution times.

In practice, since it is close to impossible ascertain that an assumed upper bound on the WCET is the actual WCET, one relies on estimates based on experience and extensive testing.

## 4.6 The History State

**4.6 The History State**

The h-state of any point of interruption can be defined as the contents of the program counter and of all data structures that must be loaded into a “virgin” hardware device to resume operation at the point of interruption.



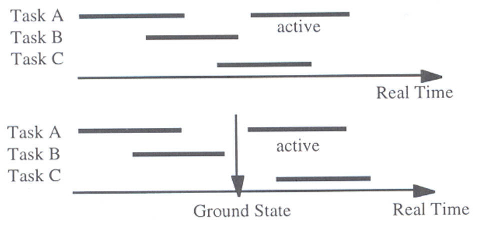
Example: pocket calculator calculation a sine(x). H-state is initially empty, and when result is displayed, is empty again. In between the series expansion of the sine takes place. If the computational device was halted anywhere between start and end, we could observe that the h-state is not empty.

Size of the h-state depends on the level of abstraction, and the time of observation. The size of the h-state can be reduced if granularity of observations is increased, and if observation points are selected immediately before or after an atomic operation.

A small h-state at the reintegration point simplifies reintegration of a failed component.

### Ground state

The ground state is a state where no tasks are active and all communication channels are flushed. Reintegration of a node after failure is simplified if a node periodically visits a ground state that can be used as a reintegration point.



### Points to Remember