ES623 Networked Embedded Systems

Modeling Real-Time Systems

05th March 2013

Outline…

- Purpose of a model
- Assumption Coverage
- Temporal properties
- Structural elements

Purpose of the Model

- Reduced representation of the world ***Model***
- Variety of models:
  - Physical-scale model of a building
  - Simulation model of a technical process
  - Mathematical model of a quantum physics phenomena
  - Logical model of the security in a computer system
- Abstractions of reality
- Model that introduces a set of well-defined concepts and their interrelationships is ***conceptual model***
- Can be informal or formal
- Formal model has advantage of precise notation and rigorous rules that support the properties of system

Model Construction

- Focus on the essential properties--eliminate the unnecessary detail (purpose, viewpoint important).
- The elements of the model and the relationships between the elements must be well specified.
- Understandability of the structure and the functions of the model is important.
- Formal notation to describe the properties of the model should be introduced to increase the precision.
- Model assumptions must be stated explicitly.

Assumption Coverage

- Every model/design is based on a set of assumptions about the behavior of the components and the environment.
- Assumption coverage: The probability that the assumptions cover the real-world scenario.
- The dependability of a *perfect design is limited by the* assumption coverage.
- Specification of the assumptions is a system engineering task.

Load and Fault Hypothesis

- Two important assumptions that must be contained in the requirements specification:
  - Load Hypothesis
  - Fault Hypothesis
- Load Hypothesis: Specification of the peak load that a system must handle.
- Fault Hypothesis: Specification of the number and types of faults that a fault-tolerant system must tolerate.
- The fault hypothesis partitions the fault space into two domains: those faults that must be tolerated and those faults that are outside the fault-tolerance mechanisms.

Temporal Properties

- Physical time is important in any real-time system
- Actions: execution of a statement
- Duration of a computational action on a given hardware between the occurrence of the stimulus and the associated response – execution time
- For a given action a
  - Actual duration
  - Minimal duration
  - Worst-case execution time (WCET)
  - Jitter

Temporal Properties

- Actual duration or actual execution time
  - $d_{act}(a,x)$ Number of time units of the ref clock z that occur between the start of action a and the termination of action a, for a given concrete input data set x
- Minimal duration
  - $d_{min}(a)$ Smallest time interval it takes to complete the action a, quantified over all possible input data
- Worst-case execution time (WCET)
  - $d_{wcet}(a)$ Maximum duration taken to complete the action under the stated load and fault hypothesis, quantified over all possible input data
- Jitter
  - Difference between $d_{wcet}(a)$ and $d_{min}(a)$

Temporal Properties

- Maximum number of activations of an action per unit of time – frequency of activations

- A resource can meet its temporal properties if the frequency and temporal distribution are strictly controlled

Structural Elements

- Real-time System: Computer System + Controlled Object + Operator
- Cluster: A subsystem of the RT-system with high inner connectivity
- Node: A hardware software unit of specified functionality
- Task: The execution of a program within a component

Task

- execution of a sequential program Task
- The software of a component is structured into a set of tasks that run in parallel
- The OS provides the execution environment for each task.
- Tasks are cooperative, not competitive.
- Stateless versus stateful tasks

Task

- *Simple tasks (S-Task)* – execute from the beginning to the end without any delay, given the CPU has been allocated.
- *Complex tasks (C-Task)* – may contain one or more WAIT statements in the task body.

Structural Elements (contd)

07th March 2013

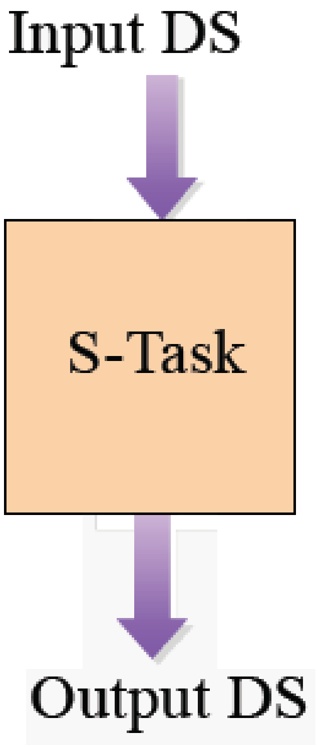
Task

- *Simple tasks (S-Task)* – execute from the beginning to the end without any delay, given the CPU has been allocated.

- *Complex tasks (C-Task)* – may contain one or more WAIT statements in the task body.

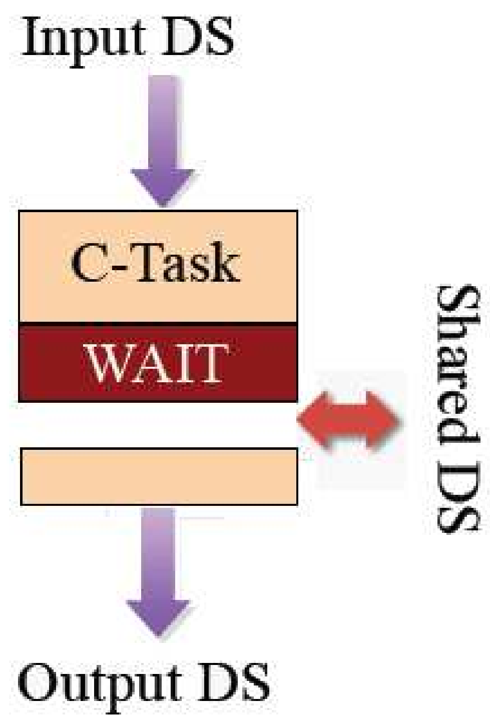
Simple Tasks (S-Task)

- Can execute from beginning to end without delay, given the CPU has been allocated to it
- No blocking (synchronization, communication) inside
- Progress is independent from other tasks
- Inputs available in input data structure at the task start
- Outputs ready in the output data structure at the task end



Complex Tasks (C-Task)

- May contain one or more WAIT operations in its code
- Possible dependencies due to synchronization, communication
- Progress is dependent on other tasks in node or environment
- Task timing of a C-task is a global issue

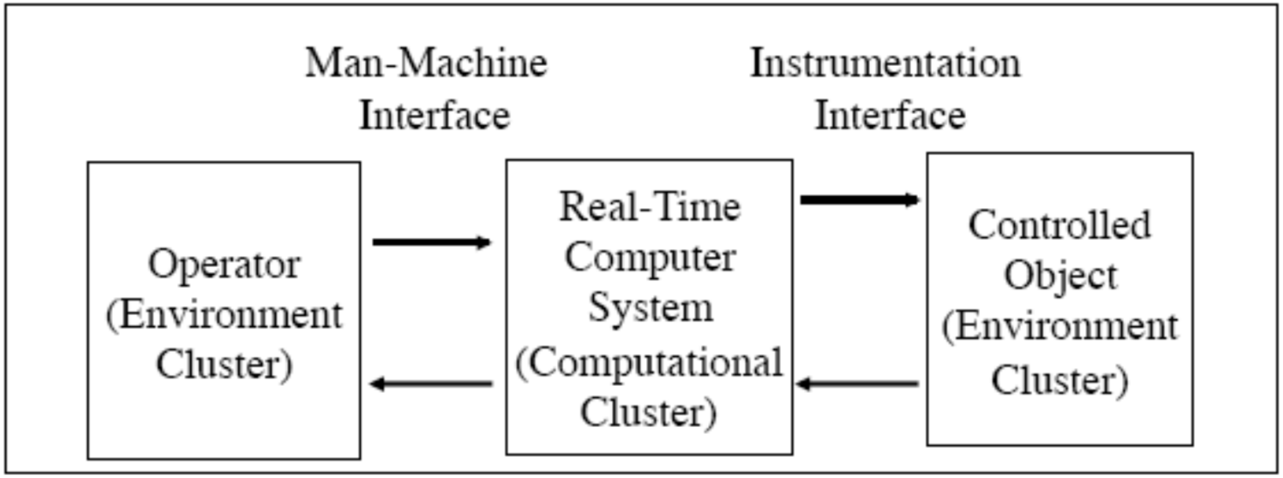


Node

- Self-contained computer with its own hardware and software, which performs a set of well-defined functions within the distributed computer system.
- Hardware – processor, memory, communication interface, interface to the controlled object
- Software – application programs, operating systems
- *A node is the most important abstraction in a distributed real-time system because it binds software resources and hardware resources into single operational unit with observable behavior in the temporal domain and in the value domain.*

Structure of a Node

- Node hardware consists of a host computer, a communication network interface (CNI), and a communication controller



Structure of a Node

- Node software resides in the memory of the host
- Divided into two structures
  - initialization state (***i-state***)
  - history state (***h-state***)
- *i-state* is a static data structure that comprises the reentrant program code and the initialization data of the node, and can be stored in ROM.
- h-state is the dynamic data structure of the node that changes its contents as the computation progresses, and must be stored in read/write memory, RAM

Structure of a Node

- In many applications, node is the ***smallest replaceable unit (SRU)*** that can be replaced in case of a fault

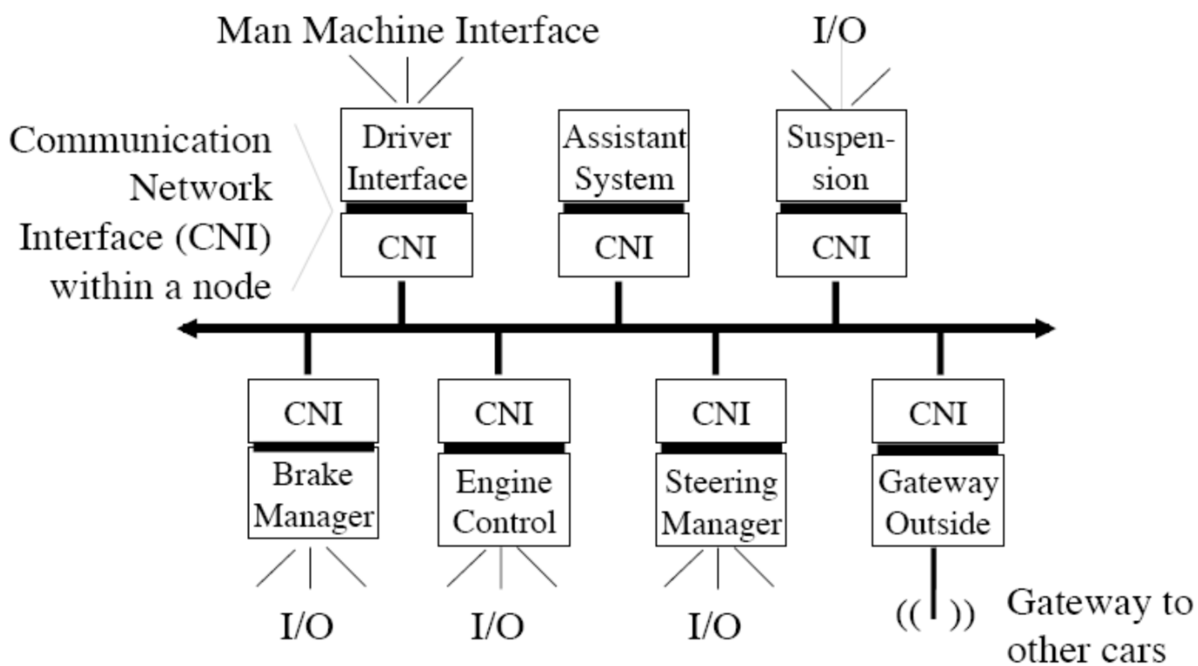
- Execution of concurrently executing tasks within a node is controlled by the node operating system

Fault-Tolerant Unit (FTU)

- Abstraction unit introduced for implementing fault tolerance by active replication
- Consists of a set of replicated nodes intended to produce replica determinate result messages – same results at approx. the same points in time
- If one of the nodes of FTU produces an erroneous result, a judgment mechanism detects the erroneous result and ensures that only correct results are delivered to the client of FTU

Computational Cluster

- Comprises a set of FTU
- Interface between a cluster and its environment are formed by the gateway nodes of the cluster
- Interconnected by the gateway nodes in the form of mesh network

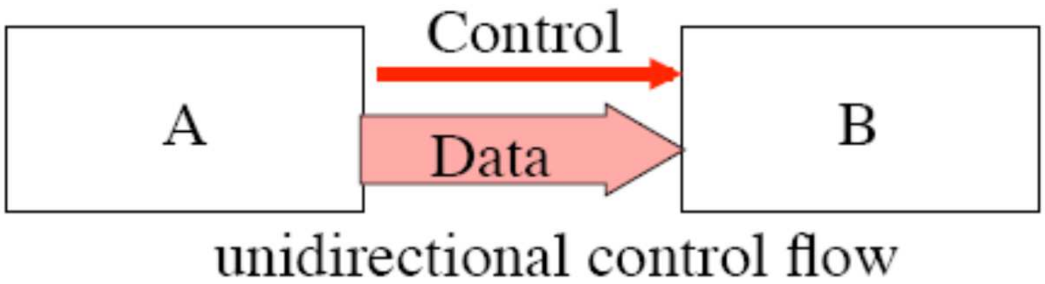


Interfaces

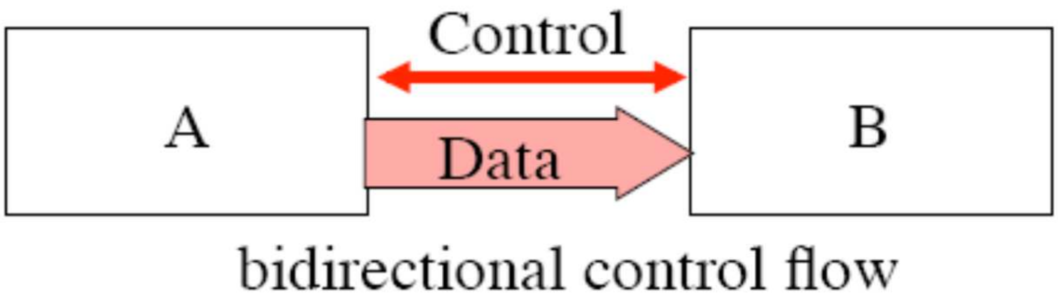
- Common boundary between two subsystems
- Characterized by
  - ***data properties***, i.e., the structure and semantics of the data items crossing the interface.
  - The semantics include the ***functional intent***, i.e., the assumptions about the functions of the interfacing partner
  - Its ***temporal properties***, i.e., the temporal conditions that have to be satisfied by the interface: e.g., update rate and temporal data validity
  - Its ***control properties***, i.e., strategy used to control the data transfer between reader and writer

Elementary Vs Composite interface

Elementary Interface:

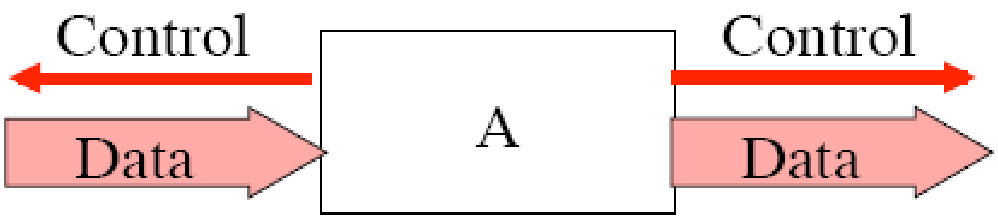


Composite Interface:



Temporal-Firewall

- Interface that does not allow to execute external control over the component



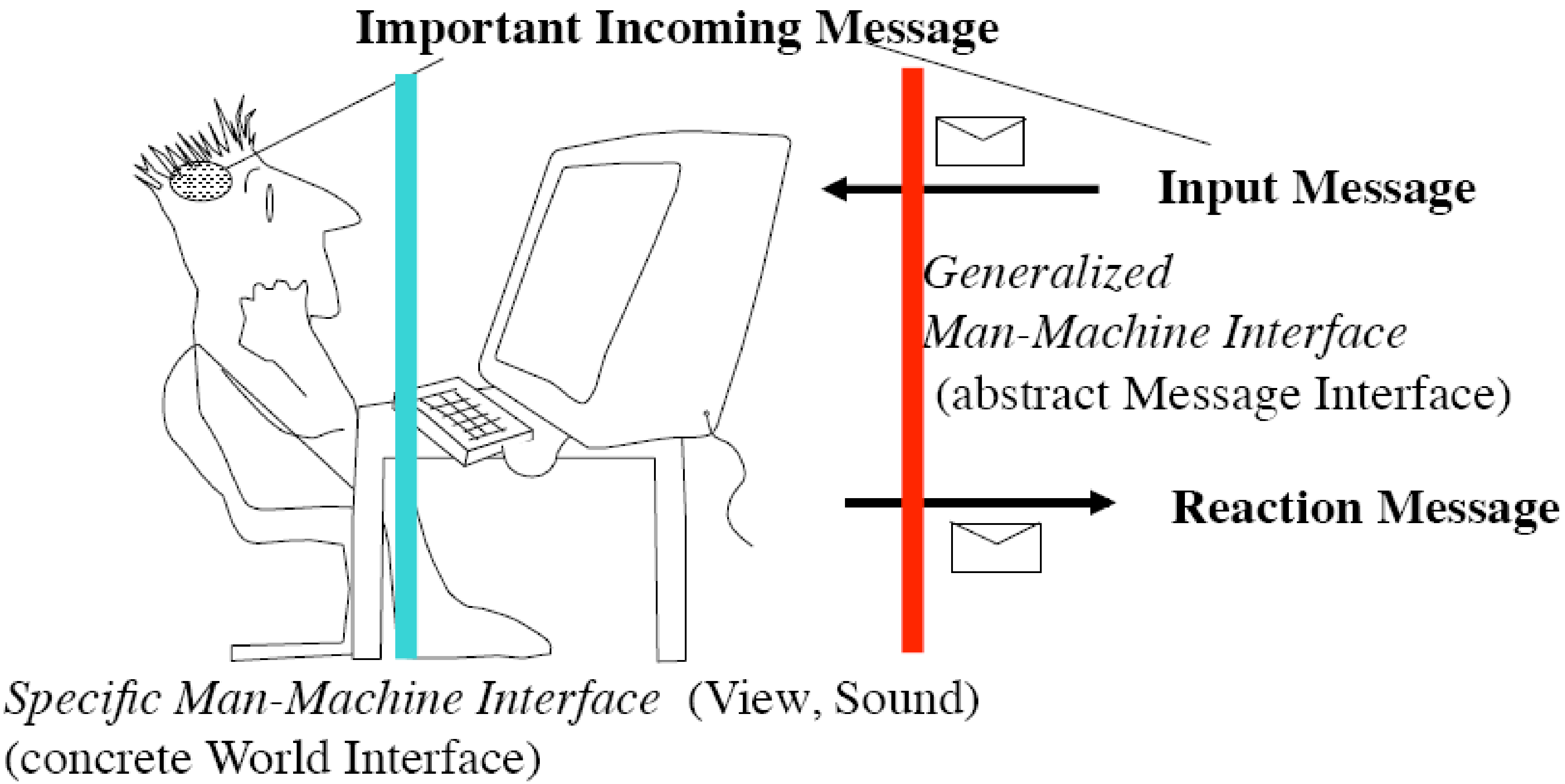
World and Message Interfaces

- low level interface the *world interface*
- internal abstract message-based the *message interface*
- interface component between the message and the world interface acts as an “information transducer” and is called *resource controller*

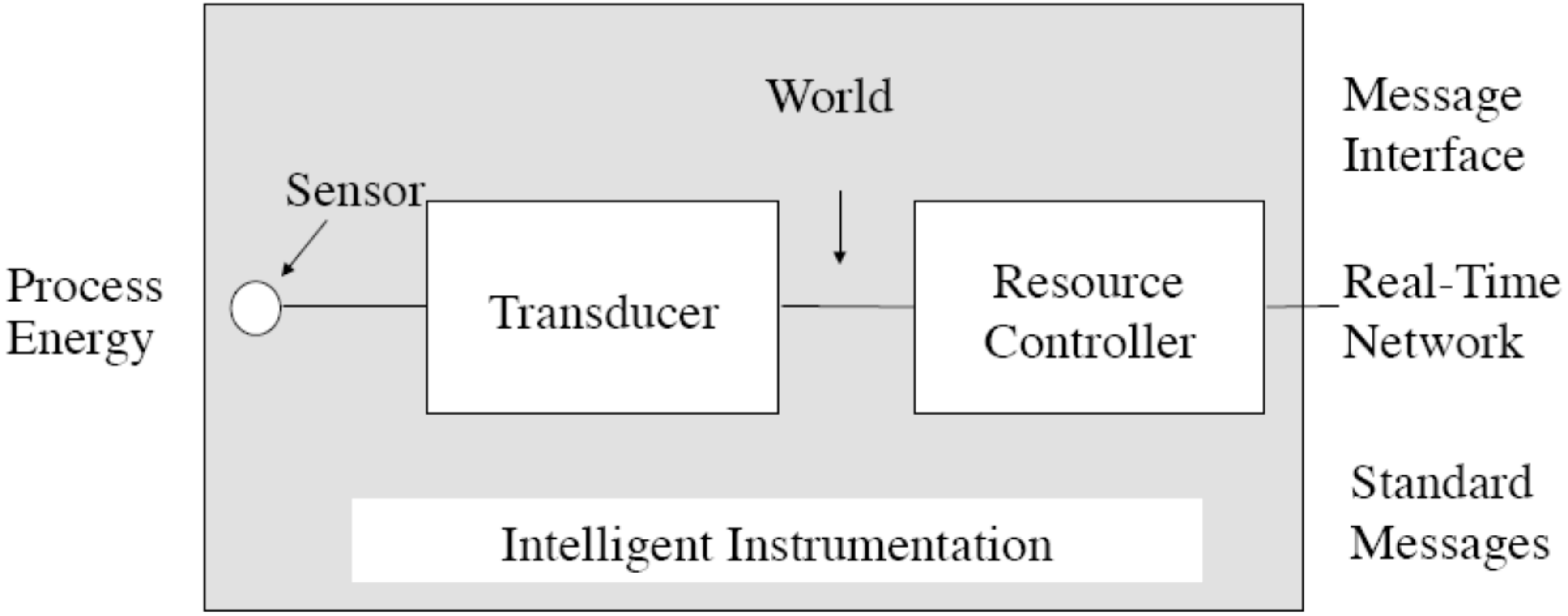
World and Message Interfaces

- Man-Machine interface (MMI)
  - *Specific man-machine interface (SMMI)*
  - *Generalized man-machine interface (GMMI)*
- SMMI
  - Concrete world interface
  - between machine and human operator
- GMMI
  - Abstract message interface
  - between MMI and rest of distributed system

Example: Man-Machine Interface



Example: Intelligent Interface



World and Message interface in distributed system