REAL-TIME COMPUTING - BASIC CONCEPTS

H KOPETZ

Rapporteur: R de Lemos
A Saeed

A DRS CONSISTS OF AUTONOMOUS COMPONENTS

- A component is a hardware software unit of specified functionality and performance

- Components communicate by the exchange of RT messages only

- A component is a unit of information hiding and intelligent under fault conditions

- At the reintegration point components should contain minimal internal state

- Components should support reasonable abstractions for fault tolerance e.g. failsilent.

What is included in the architecure design of a DRS?

* Specification of the DRS Requirements (in the form of RT transactions)

* Allocation of the RT transactions to the components of the DRS

* Specification of the functions, the external interfaces and the relevant internal states of the components.

* Specification of the messages between the components.

The specification must cover the 'deep' value and the timing properties.

Load and Fault Hypothesis

Load Hypothesis:
Specification of the peak load the system has to handle

Fault Hypothesis:
Specification of the Faults the system has to tolerate

The load hypothesis and the fault hypothesis must be contained in the requirements specification document.

Global versus Local Properties

Global properties:

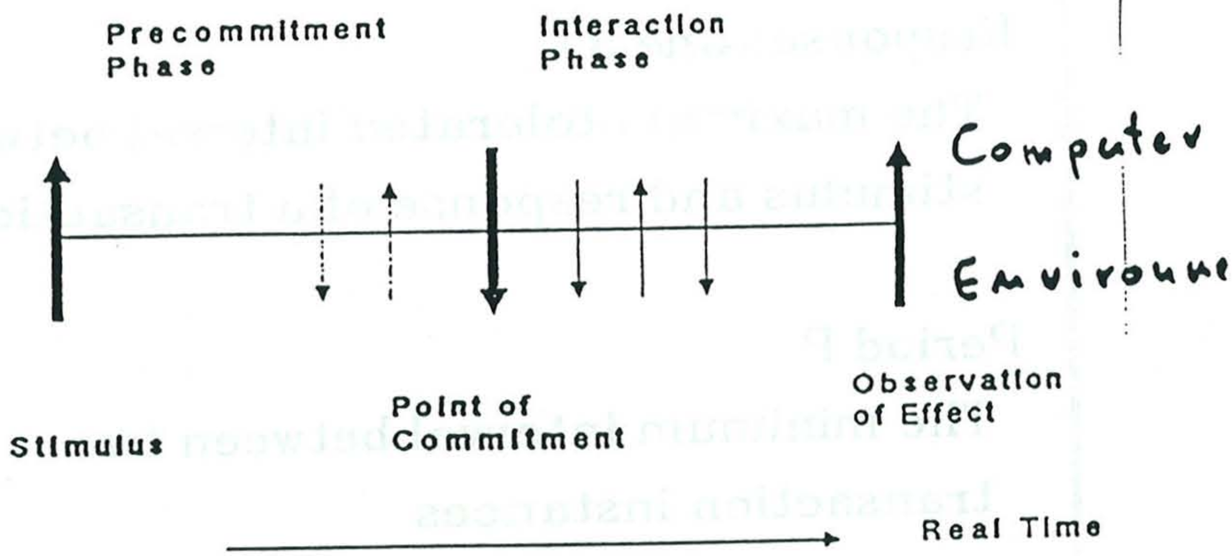
* Meaning of a message
* Function of a component
* Timing between messages

Local properties:

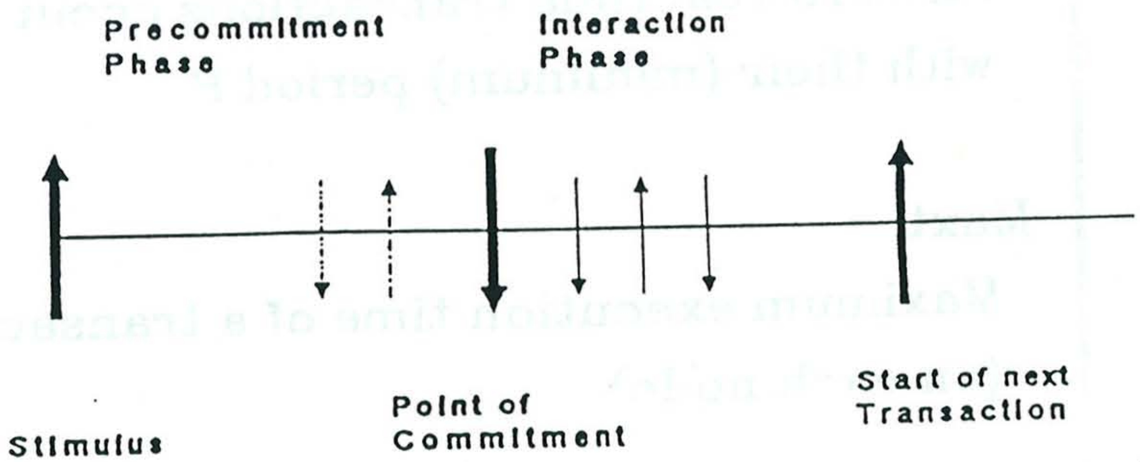
* Representation of information
* Timing within a component
* Timing between an interface component and the associated environment

At the architectural level, only the global properties have to be considered.

Real Time Transaction



Periodic RT-Transaction



Response time RT:

The maximum tolerated interval between stimulus and response of a transaction

Period P

The minimum interval between two transaction instances

Peak load:

All hard real time transactions occur with their (minimum) period P

Maxt

Maximum execution time of a transaction (on each node)

Transaction classes:

Emergency Transactions
immediate service

Hard Real Time Transactions
guaranteed service

Soft Real Time Transactions
Best effort service

Stimulus:

External Transactions
external stimulus

Internal Transactions
internal stimulus

Performance evaluation of DRS:

* Peak load is a 'rare event'

* The maximum, not the mean response times are of interest.

* Peak load is highly correlated by a catastrophic external event e.g. by a ligthning stroke

Therefore:

* It is difficult to follow arguments based on 'stochastics'

* Design must be based on deterministic mechanisms

Time rigid scheduling

Assumptions:
Global time base (<100 usec) available
TDMA Protocol on LAN

Scheduling:

A task is started at a predetermined absolute global point in time modulo the known cycle time P

Before system initialization these time rigid schedules are calculated for all hard real time tasks

Simplification
Cycle durations 2**n of basic slot times

Concurrency Control of RT Transactions

* Semantic Conficts
  Immediate conflict resolution

* Schedule Conflicts

  - Implicit Synchronization
    at compile time

  - Explicit Synchonization
    at run time

Direct Transaction Delay

The delay of a transaction provided that this transaction has immediate access to all required resources

Indirect Transaction Delay

The delay of a transaction caused by resource coflicts (buffer, Media Access etc.)

Total delay

Direct plus Indirect delay

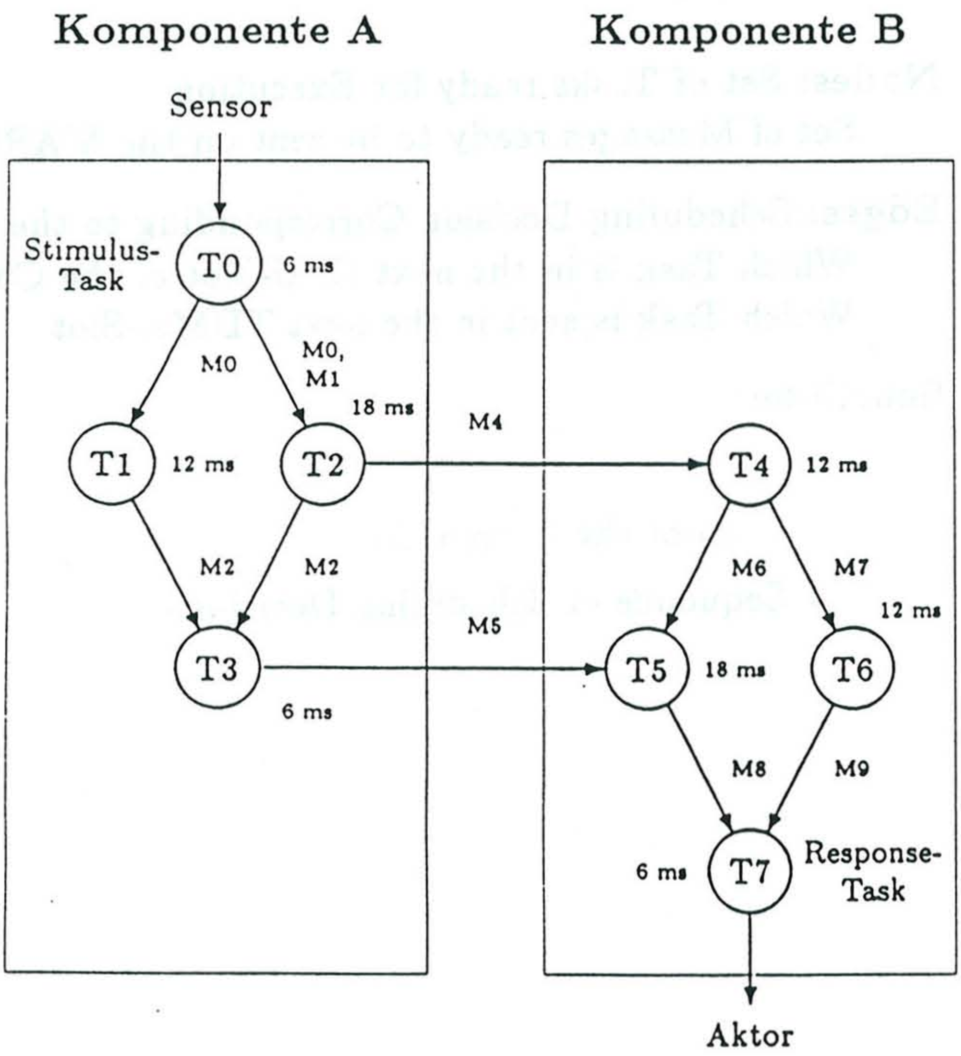
Design goal:

Minimize indirect delay of hard real time transactions.

Timing Analysis

- It is analyzed, whether the Timing Requirements of a Transaction are met
- A Transaction is the Implementation of a Stimulus-Response-Action
  - It is a directed acyclic Graph
  - Nodes represent Tasks
  - Edges represent Messages that are exchanged between Tasks
- Timing Analysis comes up with a Schedule that meets the Timing Requirements
- The Schedule is a constructive proof that all Timing Requirements are met at Runtime under all load circumstances

Sample of a Transaction executed on two Components



Search Tree

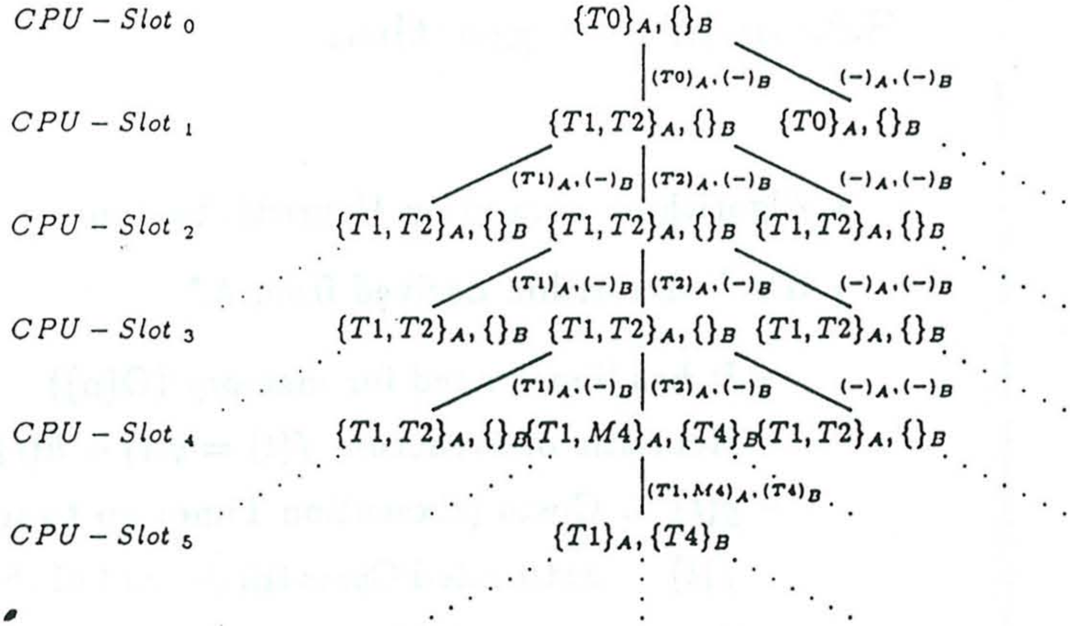
Nodes: Set of Tasks ready for Execution
Set of Messages ready to be sent on the MARS-Bus

Edges: Scheduling Decision Corresponding to the CPU-Slot
Which Task is in the next CPU-Slot of the Components
Which Task is sent in the next TDMA-Slot

Schedule:

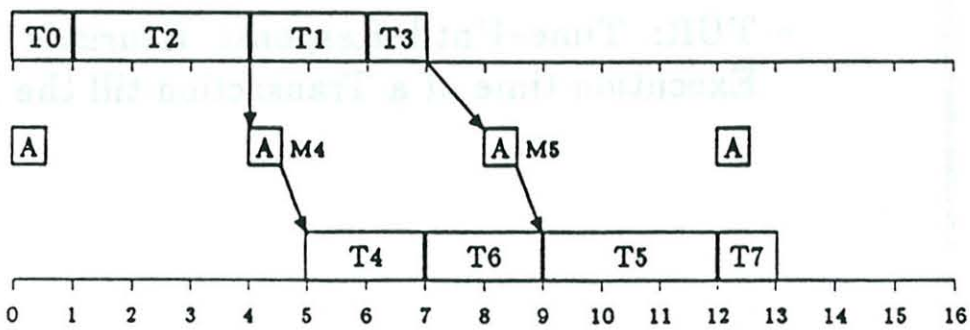
- Path of the Search-Tree
- Sequence of Scheduling Decisions

Sample of a Search Tree



Sample of a Schedule

Component A



Component B

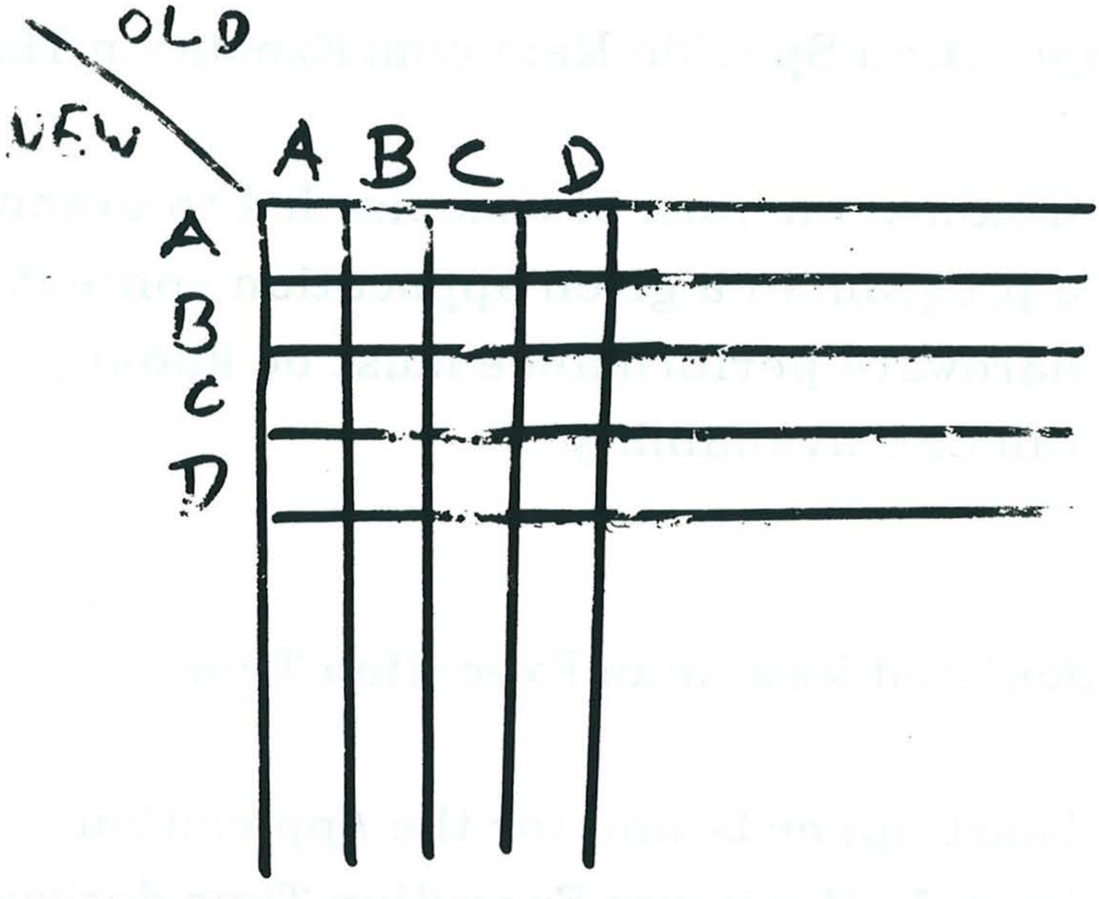
[A] ... TDMA-Slot of Component A

Scheduling–Algorithm

- Algorithm is based on Heuristic Search
- IDA*–Algorithm derived from A*
  - It has linear need for memory (O(n))
  - Heuristic of Structure $f(t) = g(t) + h(t)$ used
  - $g(t)$ ... Costs (Execution Time) up to now
  - $f(t)$ ... Estimated Costs till the end of the Transaction
  - Requirement of A*:
    $f(t)$ has to be Optimistic
    i.e. $f(t)$ must underestimate costs till the end of the Transaction
- TUR: Time–Until–Response Heuristic to estimate the Execution time of a Transaction till the Response

EMERGENCY TRANSACTION

SCHEDULE SWITCH



Definitions:

Application Specific Maximum Execution Time

maximal amount of time needed to execute a program in a given application context; hardware performance must be known; full CPU availability

Calculated Maximum Execution Time

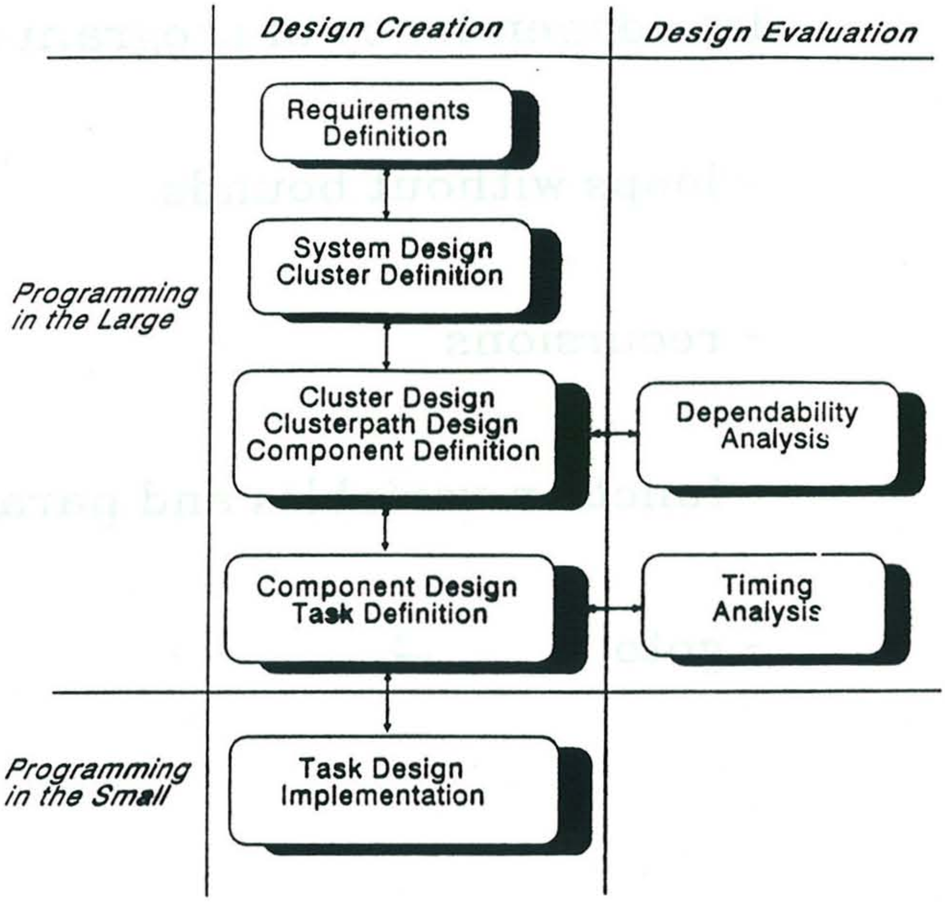
Least upper bound for the Application Specific Maximum Execution Time derived from program code

Goal: small difference MAXT_C - MAXT_A

Problems for the MAXT Calculation in Existing Programming Languages

- data dependency of program execution
- loops without bounds
- recursions
- function variables and parameters
- goto ...

The Distributed Toolset



Contract Description Language (CDL)

* Representation for the technical specification

* It has been tried to make CDL representations readable for man and machine

* Technical specification is generated in CDL by the client from the design data base

Server can parse the CDL representation and generate its local data base

The result of the server is coded in CDL

Example of a contract:

DOCUMENT: ralph/thomas.1/ORDER.1

ORDER

Project..: PROJECT.1
Contract.: ralph/thomas.1
Document.: ORDER.1
Reference:

HEADER:

Title......: timing analysis order

Sender.....: ralph
Addressee..: thomas

Duetime....: Aug. 25, 1988 at 17: 00

MANAGEMENT SPECIFICATION:

check the timing behaviour of the even designed "car-control" cluster. If the scheduling can be solved, deliver the results as usual in two ways:
(1) sorted by the passing of time
(2) sorted by tasks

TECHNICAL SPECIFICATION:

```
cluster car-control size=2 tdma-slot=1msec
   component calc-throttle location=0
       import wheels-rotation, car-status
       export throttle-setting
       task current-speed bc=32 met=8 nonpret=1
            input wheels-rotation
            output current-speed
       end task
       task calc-desired-speed bc=16 met=6 nonpret=1
            input car-status
```

page 01 of 03

DISCUSSION

Rapporteur: Rogério de Lemos
Amer Saeed

Professor Ercoli asked Professor Kopetz what was his definition for a safe state, and how it could be checked whether the system was in a safe state or not. Professor Kopetz answered that a safe state is always defined in the context of a particular application, and a safe state cannot have an abstract definition without looking at the requirements of the application. Also Professor Kopetz said that it was possible to consider a situation where a safe state could be considered as a bad state, but normally this generalization did not make a big difference.

Professor Turski questioned if it was possible to design systems that could respond in real-time peak load, in accordance with the definition, by the occurrence of two lighting striking in the power net. Professor Kopetz answered that the specification of peak load must be part of the requirements specification. It was always difficult to prove real world properties, we could only prove in a mathematical system by setting them out as mathematical problems.

Based on Professor Kopetz answer Professor Turski continued exposing his thoughts stating that there are unpredictable things and the rest is predictable by definition, so for predictable things there are very well designed tools without all the variables that consider time, and the rest was unpredictable anyhow. He continued stating that if the time splitting will have to continue undefinedly there were always subunits of time that events could happen and become unobservable. In his view, there are things which we are unable to cope with and the rest is just relations of objects where time has no importance. At this point Professor Anderson asked whether Professor Turski was presuming that system design was then trivial. Replying to Professor Turski's arguments, Professor Kopetz said there were always certain assumptions which a designer must make and which are related, for example to fault hypothesis - what are the faults which can be tolerated by the system, and will "real" real life system exhibit only these properties which the system can tolerate. There are delicate assumptions that must be made at the specification of the requirements and a similar set of assumptions which must be made in relation to design properties of the system: the peak load that is to be handled and considered, and the peak load that cannot be handled. And this is not only a question of peak or probability, or sometime, in real world situations, the question of economics.

Professor Nehmer asked how resource conflicts will be handled if the system is going to run without a real-time operating system. Professor Kopetz answered in two parts. In the first part he said that the set of processes were restricted to those which the execution time could be determined if they were executed in one processor, considering that interruptions did not occur - this task could be done efficiently off-line. In the second part, he said that their operating system has only one interrupt - the clock interrupt, which determines the frequency and allocates to every task a slot during which the task runs on the processor in an uninterruptible manner, and therefore, since there is no need to consider the interruption of execution tasks by operating system tasks, they can have a reasonable estimate of the time.