Using RTAI/LXRT for Partitioning in a Prototype Implementation of the DECOS Architecture

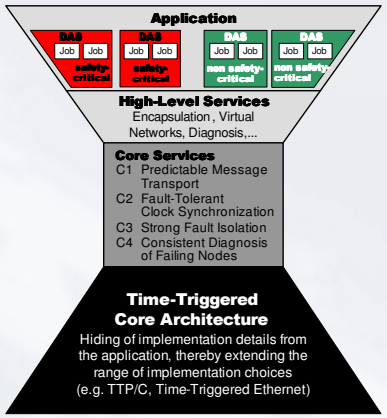
B. Huber, P. Peti, R. Obermaisser, and C. El-Salloum

Overview

- DECOS Integrated Architecture
- DECOS Component Model
- Two Dimensions of Partitioning
- Prototype Implementation
- RTAI/LXRT Execution Environment
- Results

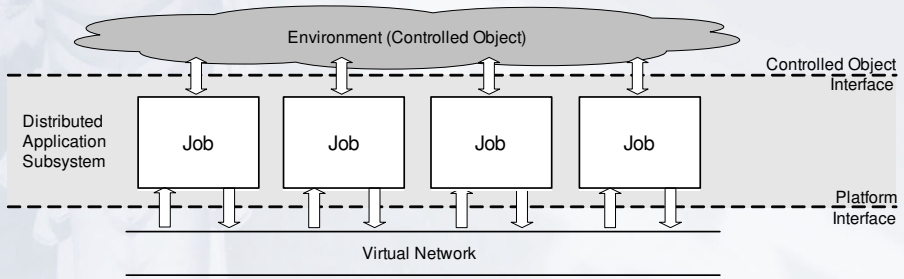
The DECOS Integrated Architecture

- **D**ependable **E**mbedded **CO**mponents and **S**ystems
- Research project founded by the European Commission under FP6
- Architecture for distributed embedded real-time systems mainly aimed at automotive and avionics domain
- An integrated architecture that combines the benefits of integrated and federated architectures



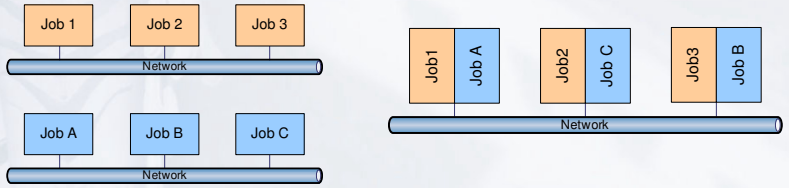
Distributed Application Subsystem

- Nearly independent distributed subsystem
- Exploit specific platform services
- Infrastructure tailored to the needs of the DAS (e.g., TT or CAN communication)

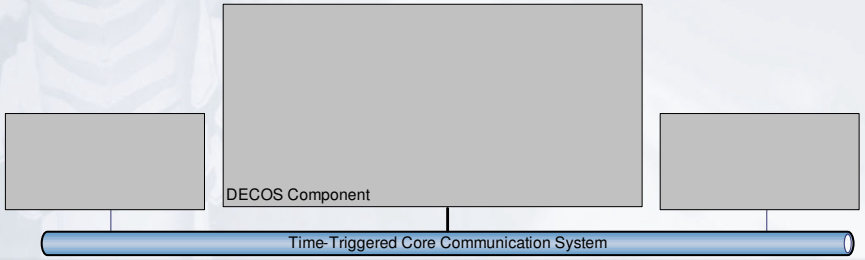


Federated System vs. Integrated System

- **Federated Architecture**: Each DAS has its own distributed computer system, i.e. a dedicated communication infrastructure, dedicated hardware elements etc.
- **Integrated Architecture**: Multiple DASs (possibly with different criticality levels) are integrated within a single distributed computer system.

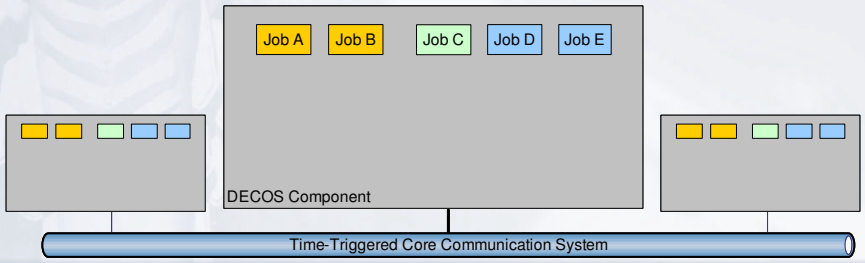


The DECOS Component Model



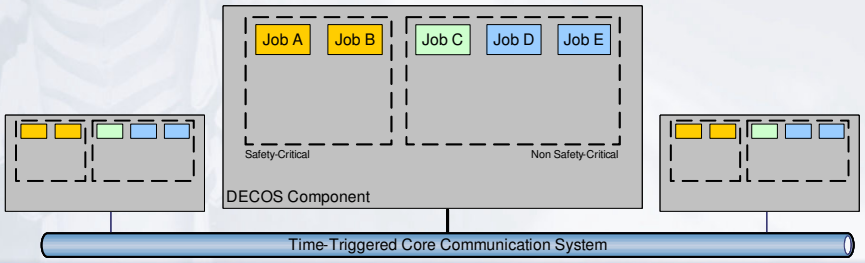
The DECOS Component Model

- Jobs of different DASs hosted on the same component



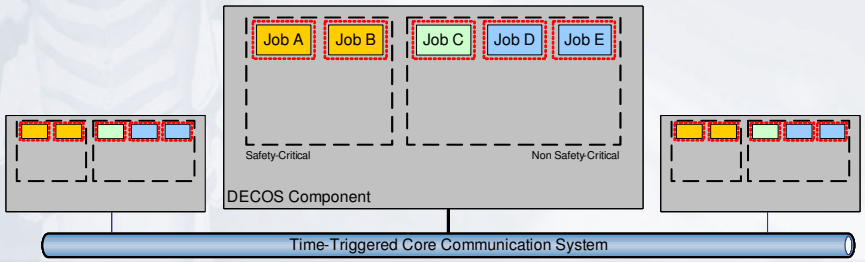
The DECOS Component Model

- Jobs of different DASs hosted on the same component
- Support for mixed criticality



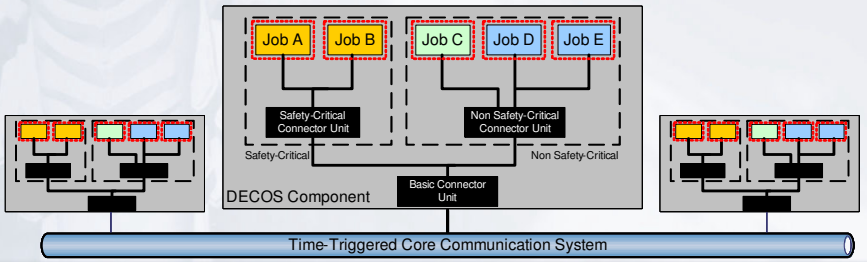
The DECOS Component Model

- Jobs of different DASs hosted on the same component
- Support for mixed criticality
- Encapsulated Execution Environment for each Job



The DECOS Component Model

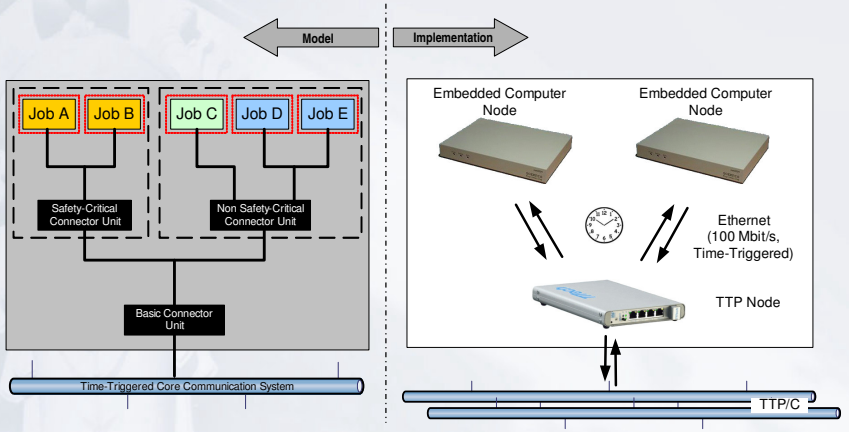
- Jobs of different DASs hosted on the same component
- Support for mixed criticality
- Encapsulated Execution Environment for each Job
- Encapsulated Virtual Communication Service for each DAS



Two Dimensions of Partitioning

- **Spatial Partitioning**
  - Preventing jobs from overwriting memory elements of other jobs (data and code)
  - Preventing jobs from interfering with other jobs in the access of devices
- **Temporal Partitioning**
  - Preventing jobs from disturbing the timing of other jobs (e.g. by holding a shared resource like the CPU)

Prototype Implementation



RTAI/LXRT Execution Environment

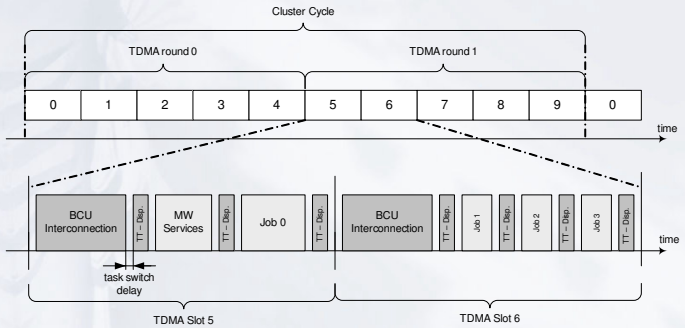
- Real-Time Application Interface (RTAI)
  - Deprives the Linux Kernel the control of the hardware
  - Linux kernel and applications are relegated to “idle tasks”
- Linux Real-Time (LXRT)
  - Enhances RTAI in order to provide real-time functionality in user mode
  - Standard Linux protection mechanisms are still available (e.g., memory protection)

RTAI/LXRT Execution Environment

- Spatial Partitioning
  - Jobs and middleware services are executed in user mode (LXRT)
  - Linux memory protection mechanisms can be exploited
  - Sharing of I/O devices is prevented by design
- Temporal Partitioning
  - Time-triggered dispatcher handles in combination with the RTAI real-time scheduler the execution of jobs
  - Dispatching is based on a static dispatcher tables (synchronized to the communication on the core network) that specify explicit time budgets for each job.

RTAI/LXRT Execution Environment

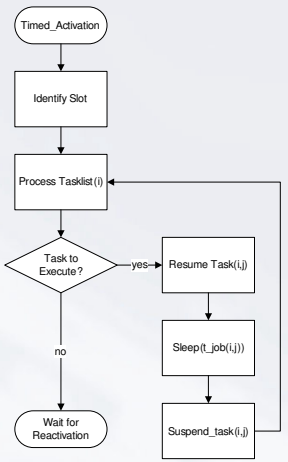
- Time-Triggered Dispatching



RTAI/LXRT Execution Environment

- Time-Triggered Dispatching
  - *n* dispatcher tables (one for each slot of the TDMA round)
  - *m* tasks per slot (jobs and middleware)
  - Accumulated execution time of all jobs (t_{job}) must not exceed slot length

$$t_{slot,i} \geq t_{comm} + n_i \cdot (t_{dispatch} + t_{tswd}) + \sum_{j=1}^{n_i} t_{job,i,j}$$



Results

- Independence between Partitions
  - No constraints on execution time of jobs – they cannot exceed their assigned time budget
  - Memory Protection is ensured by operating system mechanisms
  - Unexpected termination of any job has no influence on other jobs or middleware
- Overhead
  - Minimal and constant overhead introduced by the time-triggered dispatcher (18µs)
  - Low overhead introduced by RTAI task switch (40µs to 65µs)
- Transparency to Jobs and Middleware
  - Jobs and middleware are developed as “usual” Linux applications
  - Intellectual property (IP) protection is supported
  - RTAI/LXRT specific functionality is linked at system integration

Vienna University of Technology

Thank you for your attention!

Any Questions?