DECOS: Dependable Embedded Components & Systems

Neeraj Suri
(with inputs from the entire DECOS team)

www.deeds.informatik.tu-darmstadt.de
The “Why & How” Outline

• Federated & Integrated Dependability approaches
• DECOS objectives
• DECOS foundations (PIM, PIL, PSM, ...)
• DECOS SP’s and demonstrators
ES ➔ DES Context

• **Embedded Systems (ES) involve** computing elements
  - “... that are part of a larger system”
  - “... whose primary task is not standalone computations”
  - ... are often resource constrained

• **Dependability:** The trust we (*are willing to*) put into a system to provide for “sustained” delivery of desired services in the presence of perturbations (internal or external)

- ES utility is a consequence of our perception and trust for the degree of dependence (D) we put into them: ES ➔ (D)ES
...the federated approach to dependability

- identify safety critical and non-safety critical functions
- identify relevant fault/error hypothesis
- target containment of errors by separation (ECR's)
  - physical separation/containment
  - temporal separation/containment
  - logical separation/containment
Federated
Automotive/Aerospace (Federated Systems)

**Applications**
- Navigation
- Steering
- Commn.
- Multi Media

**Middleware**
- Engine/Flight Control
- Dist. Resources
- Nodes + Commn.

**Resources**
- User I/O
- Diagnostics
- Env. Ctrl.
- Braking
- Body Elec.

multiple nodes, varied criticality buses, clusters, bridges (HW, SW), ...
System Design...

- We certainly **know** how to develop systems that are:
  - application specific
  - platform specific
  - domain specific
  - technology specific
  - federated
  - expensive & customized! 😊

- Do we know how to build them as generic, open & component based infrastructures that are **NOT**:
  - application specific
  - platform specific
  - domain specific
  - technology specific
  - network specific

Can we either (a) afford or (b) technologically keep re-designing ever increasingly complex systems and still expect stable dependable services?
... the base complexity & over changes + the re-design question?

Middleware Services
(Exist as stable, formally analyzed & validated form!!!)

Do we need to keep re-inventing the core services for each new system or should we rather define the basic interfaces linking apps and impl. to the core services?
The case for partitioning?

Do we re-design over each app change?
Do we re-design over each upgrade?
Do we re-design over each imp. change?
Do we re-verify/certify over each change?

If the core and partitioning can be developed and sustained, does the need for re-... still apply?
Case for interface specs to hide the impl. dependencies?

Component
- OS
- Middleware
- Prog. Support
- Communication
- RT/Scheduling
- Mem. Mgmt.
- Etc.

Linking Interface Specification
(In Messages, Out Messages, Temporal, Meaning--Interface Model)
DECOS: The **Integrated** Composition Approach

- **Basic Services**
  - Transport
  - Co-ordination
  - Membership
  - Fault Isolation

- **Resources**
  - Shared, Distributed/Networked

- **Higher-level services**
  - Layer Transport
  - OS/Net Interface
  - Diagnosis
  - Gateway

- **Set of varied criticality apps/sub-systems**

- **Interface**

- **(Validated) Core Middleware Services**
  - Maintain stable core
  - Maintain clean interfaces
  - Maintain partitioning
  - Apps change, impl. change
  - Designer sees a virtual “federated” design env!!!
If we can define clean partitions and interfaces, can’t we logically progress from “Federated” to “Integrated” Compositions?
**DECOS Basics**

**Objective:**

Development of fundamental enabling technologies to facilitate shift from *federated* to *integrated* design of dependable real-time embedded systems

Ideally: Domain Independent  
Application Independent  
Platform Independent

**Start:** July 1\(^{st}\), 2004, **Duration:** 3 Years, **Budget:** 14.3 Mio €, **EU Funding:** 9 Mio €
DEEDS: Dependable Embedded Systems & SW

DECOS

DECOS Objectives

Facilitate systematic design & deployment of “integrated” systems in DES via:

- **Electronic Hardware Cost Reduction** (ECU’s, connectors, networks, MW …)
- **Enhanced Dependability by Design** (clear partitioning of safety-critical and non safety-critical subsystems by design)
- **Reduced Development Costs** (modular certification, reuse of software components, structured integration for communication & computational elements)
- **Diagnosis and Maintenance** (diagnosis of transient and intermittent component failures)
- **Intellectual Property (IP) Protection**
DECOS Partners

• **Co-ordinator:** ARCS

• **Industrial Partners:**
  - Audi, Airbus, EADS, Infineon, TTTech, Fiat, Profactor, Hella, Liebherr, SP, Thales, Esterel

• **Universities:**
  - TU Vienna, TU Darmstadt, TU Hamburg, Uni Kassel, Uni Kiel, Uni Budapest
The DECOS Architecture

DAS | DAS | DAS | DAS
---|---|---|---
Gateway Service | Gateway Service | Gateway Service | Gateway Service
Virtual Network Service | Virtual Network Service | Virtual Network Service | Virtual Network Service
Encapsulation Service | Encapsulation Service | Encapsulation Service | Encapsulation Service
Fault Tolerance Service | Fault Tolerance Service | Fault Tolerance Service | Fault Tolerance Service
Diagnostic Service | Diagnostic Service | Diagnostic Service | Diagnostic Service

Core Services
- C1 Predictable Message Transport
- C2 Fault-Tolerant Clock Synchronization
- C3 Strong Fault Isolation
- C4 Consistent Diagnosis of Failing Nodes

Base Architecture
Hiding of implementation details from the application, thereby extending the range of implementation choices

Technology invariant mapping of functionality to resource platforms

Distributed Application Subsystems

DAS-Specific High-Level Services

Core Services
DAS-Independent
**Platform/Tech./Domain Independent Component Oriented Flow**

- **DAS**: Distributed Application Subsystem
- **PIM**: Platform Independent Model
- **PSM**: Platform Specific Model
- **PI(L)**: Platform Interface
DECOS Overview: PIM - PI - PSM Flow

**DAS Requirements**

- **PIM Gen. Tool**
  - UML, SCADE, Matlab/Simulink
  - LUSTRE (data excg)

**DAS modeling and HW/SW integration**

- **PIM**
  - Resource Layer/PI Specs
  - HW/SW Intg.
    - Dependability Partitions, Scheduling, Res. Constraints (Hard & Soft)

**DAS SW development environment**

- **DAS-Model**
  - DAS(-SW) Modeling-Tool
  - Code Generation

**Simulation / SIL**

- DAS Code
  - Deployment
  - final PSM executables

**Feedback**

- PIM API's, Meta Models?
- HW/OS/Middleware, Protocols, Gateways, Comm. Links, …
Outline

1. Specific Platform
2. Platform Interface Layer (PIL) and Middleware
3. Platform Specific Model (PSM)
4. Platform Independent Model (PIM)
   Distributed Application Subsystem (DAS)
The Base Platform/Core Architecture

The platform is not predicated on any architectural paradigm, but on its ability to support the core and high-level services reqd. of it! Any platform satisfying them suffices!!!
Mixed-Criticality Node

- Integrated Architecture requires mixed-criticality node

- Node hosting safety-critical and non safety-critical jobs of several Distributed Application Subsystems (DASs)

- Node contains a commn. controller that is connected to one or more app. computers via connector units

- From Kopetz, H., lecture slides, distributed RT systems engineering, 2004
The DECOS Component Model

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

- **Spatial Partitioning**
  - Preventing overwriting memory elements of other jobs (data and code)
  - Preventing interfering with other jobs in the access of devices

- **Temporal Partitioning**
  - Preventing disturbing the timing of other jobs (e.g. by holding a shared resource like the CPU)
DECOS Component Structure

- Layers perform a stepwise abstraction of the underlying platform.
- Communication between adjacent layers occurs via ports.
HW Model

- Safety-Critical Subsystem
  - Application Computer (Processor Core)
  - Safety-Critical Connector Unit
- Non-Safety-Critical Subsystem
  - Application Computer (Processor Core)
  - Complex Connector Unit
- Basic Connector Unit
- Communication Controller
  - e.g., DECOS Component

- HW Node
  - HW Node
  - HW Node
  - HW Node
  - HW Node

Physical Network (e.g., TTP/C)
The DECOS Component

Application Computer

Safety-Crit. Connector Unit
Adaptor Layer
VN, Gateways, Diagnosis
Allocation Layer

Complex Connector Unit
Adaptor Layer
VN, Gateways, Diagnosis
Allocation Layer

Core Comm. Service
TT State Message Interface

Communication Controller

Basic Connector Unit

Hardware Element
Software Module
Partition

App. Code + App. MW = Job
API
time & value info

e.g., CAN, CAN on TT Ethernet
e.g., TT Ethernet

Interconnection Protocol

HL Services
DECOS Component Application Types

- **Legacy CAN Node**
  - FPGA
  - CAN
  - **Connector Unit**
    - Adaptor Layer
    - VN, Gateways, Diagnosis
    - Allocation Layer

- **Application Computer**
  - TT App.
  - CAN App.
  - IP App.
  - App. MW

- **Connector Unit and Application Computer**
  - App. Code
  - App. MW
  - VN, Gateways, Diagnosis

a) b) c)
Scope of the Hardware Specification Model (HSM)

• Specifies the structure of a DECOS cluster. (Cluster Model)
• Specifies the available resources (CPU, memory, bandwidth, etc.) of a DECOS cluster. (Hardware Resource Model)
• Resource Constraint input for SW-HW integration
Hardware Specification Models - Levels

- Cluster Model
- Hardware Model
- Architectural Service Model
Hardware Specification Model - Overview

Cluster Model

Represents concrete assembly of DECOS cluster via:

- the number of components, the type and internal setup of components,
- the physical networks that are used for inter-component and intra-component communication.
Hardware Specification Model - Overview

Hardware Model

Model used to specify resource “building blocks” for the

- Cluster Model, which forms the platform of DECOS components.
- Represented resources are: computational resources, communication hardware and I/O elements
Hardware Specification Model - Overview

Architectural Service Model

 Represents the resource requirements of the architectural services.

 Depending on the jobs hosted on a component, a particular set of architectural services has to be realized. Thus, the determination of the overall resources required for the realization of the architectural services is an iterative process related to the software-hardware integration leading to the PSM.
Outline

1. Specific Platform
2. Platform Interface Layer (PIL) and Middleware
3. Platform Specific Model (PSM)
4. Platform Independent Model (PIM)
   Distributed Application Subsystem (DAS)
Platform Interface Layer & Base Services

PI(L): Architecture/Platform Level Interface
## Core & PIL Services

<table>
<thead>
<tr>
<th>Service</th>
<th>Core Services</th>
<th>PIL Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport Service</td>
<td>TT Communication Network</td>
<td>TT, ET, Multimedia</td>
</tr>
<tr>
<td>Global sparse time base</td>
<td>Clock Synchronization</td>
<td>DAS-specific time format</td>
</tr>
<tr>
<td>Membership</td>
<td>Component-level</td>
<td>Job-level</td>
</tr>
<tr>
<td>Fault Isolation</td>
<td>Component-level</td>
<td>Job-level, Virtual-network level</td>
</tr>
</tbody>
</table>
Platform Interface

• Description of a stable interface for PI services:
  - Generic time-triggered virtual network service (Temporal Firewall Interface)
  - Generic event-triggered virtual network service (event-triggered send and receive)
  - Component membership service
  - Generic global time service

• Accessed by application via interconnection protocol

• Definition of Generic Platform Interface
  - Abstract specification of services provided by PI, based on “buffers”
  - “PI Bindings” specify how a concrete implementation (e.g. software, hardware) of PI is accessed
Generic PI and PI Bindings

```c
et_Status et_send_msg(
    et_msg* msg,
    port_id receiver)
```

Diagram showing the relationship between Global Time, Node-Membership, Generic TT ports, and Generic ET ports.
Abstraction Layers: PIL $\rightarrow$ API

- Implementation of DECOS base architecture transparent to jobs
- PIL = interface at arch. level
- API = interface at appl. level
  - Application Middleware
  - Application Code
Application Middleware

• Examples for specific services realized in the application middleware:
  
  - Specific virtual network services (e.g. specific CAN APIs like BasicCAN, FullCAN, …)
  - Provision of the global time in specific time-formats (e.g. NTP)
  - Operating system specific services for accessing I/O, task management, and inter process communication inside a job
  - End-to-end job-level membership service

• API:
  
  - Defined individually for each service implemented in MW
Application Programming Interface (API)

• Provided by application middleware

• Utility service (not part of architecture) that simplifies application development by providing higher level protocol services (TCP, UDP etc)

• Purposes
  - **Adaptor**: hides component-internal protocol between application computer and connector units (e.g., CAN on TT Ethernet)
  - **Higher protocols** (e.g., TCP) on top of the basic protocols provided by the PIL (e.g., CAN, IP)
  - **Operating system services** (e.g., interprocess communication, memory management)
Related PI(L) Issues

PI (and High-Level Services) as Resources

- **PIM→PSM-mapping** (HW/SW-integration) needs specification of available resources, in particular PI (and high-level services) for
  - selecting required elements according to specific PIM
  - configuring/tailoring them (e.g. gateways)

  ⇒ a notation is essential for describing properties (and 'costs') of those elements usable during HW/SW-integration

  ⇒ it needs to be provably established that specifications correctly describe available resources and fn. bindings
Outline

1. Specific Platform
2. Platform Interface Layer (PIL) and Middleware
3. Platform Specific Model (PSM)
4. Platform Independent Model (PIM)

Distributed Application Subsystem (DAS)
The DAS Model

- **Distributed Application Subsystems**
  - nearly independent distributed subsystem
  - utilize specific platform services
  - partition as encapsulated execution environment for jobs
  - virtual network as an encapsulated communication service for a DAS
DAS Internals

![DAS Internals Diagram]
DAS → PIM: What should it contain?

• Description of DAS functionality
  - black-box like functional input-output specification
  - state description for linking interfaces

• Definition of DAS non-functional requirements
  - dependability requirements
  - performance requirements

• Definition of DAS out-of-norm behavior
  - fail-safe / fail-silent / degradation exception handling

• Marking: Bindings, Inter-DAS relations, Resource Constraints etc
The PIM metamodel
PIM metamodel

PIM metamodel 0.11w

- Functionality
- Performance
- Dependability
Modeling workflow

- PIM metamodel
  - domain specific PIM metamodel
    - language specific PIM metamodel
      - PIM
        - marked PIM
          - PSM
            - platform files

conceptual view

engineering view
Outline

1. Specific Platform
2. Platform Interface Layer (PIL) and Middleware
3. Platform Specific Model (PSM)
4. Platform Independent Model (PIM)

Distributed Application Subsystem (DAS)
The PIM – PI – PSM Flow

**DAS Requirements**

- **PIM Gen. Tool**
  - UML, SCADE, Matlab/Simulink
  - LUSTRE (data excg)

- **Resource Layer/PI Specs**
  - PI API’s, Meta Models?
  - HW/OS/Middleware, Protocols, Gateways, Comm. Links, …

- **HW/SW Intg.**
  - Dependability Partitions, Scheduling, Res. Constraints (Hard & Soft)

- **PIM**

**DAS modeling and HW/SW integration**

- **DAS-Model**

**DAS SW development environment**

- **Simulation / SIL**
  - feed-back

- **PSM**

**Code Generation**

- **DAS(-SW) Modeling-Tool**

**Deployment**

- **final executables**

**DAS Code**
Automotive/Aerospace (Federated Systems)

Embedded Arena

*SW based functionality*

- Diverse perf., RT & criticality + resource constraints

*SW assimilation?*

30-40% “glue” SW & also results in majority of bugs

*(Boeing, Daimler, NASA)*
PSM/HW-SW Integration Dimensions

- **Practical Process**: Transformation Approaches (local: 1 constraint at a time)

- **MVO Process**: Constrained multi-variable optimization (global: multiple constraints)

- Allocation
- Scheduling
- Configuration Data
Transformation Overview

- Inputs (e.g., PIM, PI, HW model)
- Interactive allocation
- Partitions housing the jobs for scheduling
- Scheduling
- PSM
- Configuration file: Mapping, scheduling, resource usage

- HW platform constraints
- Dependability Constraints
- Code info (SCADE)
The Constraints

• Fixed Bindings (Jobs to Nodes)
• Dependability
  – SC & non-SC partitioning
  – Separation over replication
• Computing Node Constraints
  – Computational Capability
  – Memory
• Communication Constraints
  – Bandwidth
  – Allocation/Sharing Protocol
• Timing Constraints
  – Precedence Relations
  – Deadlines
The Basic Integration Process
The Transformation Process
Transformation development

Metamodels

Graph transformation rules

Transformation control structure

Transformation development & testing

Transformation trace, debug information

Source model

Transformation design time

Native transformation plugin

Transformation runtime

Target model
TTA development (via DECOS)

- UML CASE tool
- PIM
- Transformation
- PSM
- TTP-Plan
- cluster plan
- TTP-Build
- code generation
- source
- OS config
- FT layer
- PIL description
- host plan
VIATRA/SCADE Transformation Approach

- Modeling tools?
- Scheduling?
- Generalizations?
- Inputs?
- PIM/PI constraints?
VIATRA Transformer

- VIATRA = VIual Automated model TRAnsformations

  - general-purpose model transformation (transware) framework
  - supports the entire life-cycle for transformations
    - specification
    - design
    - execution
    - validation
    - maintenance
  - within and between various modeling languages
The VIATRA 2.0 framework

Eclipse framework

VIATRA 2.0 Model Transformation Plug-in

VPM Metamodelling Core

Source

Source model

Xform. rules
(UML/QVT)

Xform engine
(ASM+GraTra)

Target

Target metamodel

Target model

Source metamodel

Meta XForm

Native

Source model

Native XForm Plugin

Native

Target model

DECOS tool
DEEDS: Dependable Embedded Systems & SW

DECOS

DECOS2SCADE Architecture

DECOS PIM model
Rational Rose XDE

XDE IMPORT

VIATRA transformation

Code format

SCADE source code
Mapping from DECOS PIM to SCADE

<table>
<thead>
<tr>
<th>DECOS PIM</th>
<th>SCADE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAS</td>
<td>Project</td>
</tr>
<tr>
<td>Job</td>
<td>Node</td>
</tr>
<tr>
<td>Interface</td>
<td>-</td>
</tr>
<tr>
<td>Port</td>
<td>Port</td>
</tr>
<tr>
<td>Message type</td>
<td>Datatype</td>
</tr>
</tbody>
</table>
DECOS PIM to SCADE - Sample transformation program

// Pattern for recognizing jobs
pattern jobs(Z) =
{
    uml.metamodel.'Foundation'. 'Core'. 'Class'(Z);
    uml.metamodel.'Foundation'. 'Extension'. 'Stereotype'(X);
    uml.metamodel.'Foundation'. 'Core'. stereotype_ModelElement_Stereotype(Y,Z,X);
    datatypes.'String'(STR);
    uml.metamodel.'Foundation'. 'Core'. 'ModelElement'.name(NAME,X,STR);
    check (name(STR)=="Job")
}
//main rule (entry point) of transformation
rule main() seq
{
    log(info,"PIM2SCADE started...");

    {
    
    forall C with pattern jobs(C) do
    seq
    {
        print("//Found job : "+fqn(C));
        print("//!!FILE="+name(C)+".saofd");
        print("node "+name(C)+"(");
        ...
        ...
        print("//!!ENDFILE");
    }
    
    log(info,"PIM2SCADE ended...");
    }
}
Status - Viatra Framework

- Visual design + programming
- Reusable transformations
- Separation of the design and execution environments
  - Native transformation plugins
- Certification
  - Transformation plugins (needs adoption to DO178B)
  - Viatra will not be certified
- Still in academic prototype phase
Theoretical PSM Basis: Allocation, Scheduling, Config.

Timing, Safety critical, Replicas (need to be mapped in distinct resources), Physical resources (Processor, Memory, I/O, Transducers), Network Resource (bandwidth) + …

Issues:
- Criticality of constraints
- Mapping process
- Heuristics/algorithms
- Trade off of integration
- Tool support (SCADE, TTTplan, TTTbuild)
Sw Model

HW Model

Clustering SW modules, Constraints

Mapping Algorithms/Heuristics

Feasible Assignment (one or more solution)

Assessment Framework

Replication Influence

Good/Optimal Mapping

Multi Variable/Attribute Optimization (MVO)

Dependability - the primary concern
Feasibility Test (constraints vs. features)

Infeasible Assignments

Interactive Allocation

tentative (partial) allocation

App. MW Selection

APP. MW Pool

MW Resource Requ. Calculation
(assisted by SCADE)

Scheduling & Feasibility Test
(assisted by SCADE)

Completeness Test

PSM

UML 2 SCADE Transformation

SCADE Model

HW Specification Model

computational resources

Architectural Service Model

constraint list

(feature list)

constraint list

(marked) PIM

identified jobs

Job Resource Requ. Calculation

Job Code

(possibly generated by SCADE)
Allocation and Scheduling

- Requires consideration of a multitude of constraints (and it is probably impossible to find a single appropriate formalism to express all of these constraints)
- Done iteratively in progressive steps
- Must be easily extensible (user might want to add new constraint classes that were not considered when allocation algorithm was designed)
- Must be configurable (e.g., change of underlying base architecture might require adaptation of scheduling algorithm)
- Must be proven to be sound → monolithic algorithm is inappropriate
A&S is a Constraint Satisfaction Problem

- Conjunction of constraints over variables (probably in different formalisms): $c_1$ and $c_2$ and ...

- Find ground instances (i.e., satisfying assignment)

→ Divide and Conquer: Let specialized constraint solvers solve $c_1$, $c_2$, ... separately, cooperate to solve the whole problem
Cooperating Constraint Solvers and Conflict Propagation

- **Control**
  - Controls search process (backtrack, backjump, case-split, variable order, optimization)

- **Blackboard**
  - Contains tentative solution(s)
  - Contains conflict information

- **Constraint Solver**
  - Provides conflict information
  - Specialist for a single class of constraints
  - Considers value order

- **Node Allocator**
- **I/O Allocator**
- **Scheduler**
Specialists for Classes of Constraints

• Tries to find a valid variable assignment

• Report conflicts
  - ideally *symbolic* information, e.g.,
    - processor (job₁) <> processor (job₂),
    - \( t₂ < t₅ \)
  - or explicit conflicts, e.g.
    - \( t₂ <> 10 \)

• Makes suggestions for case splits, e.g.
  - \( t₂ = 5 \) or \( t₂ < 3 \)

• Chooses appropriate values with respect to optimal value order
  - Most promising value vs. fail first strategy
Control Class

- Controls the search process (e.g. search depth, case split)

- Considers optimal variable order (i.e., which constraints are considered first) based on information provided by specialists (“assessCost” method)

- Worst Case: Search with Backtracking (if no symbolic conflicts information can be provided)

- If sufficiently good conflict information is available:
  - Back[jumping, checking, marking]
  - Forward checking/Look Ahead
  - Hill-Climbing/consider neighborhoods
Advantages

- Constraint classes that are not considered now can be added later

- Constraints can be specified in different formalisms (but there has to be a simple formalism to exchange conflict information)

- Existing tools can be integrated (using a bridge), provides a plug-in mechanism

- Parallelizable
Disadvantages and Problems

• How to compute “symbolic conflict information”?  

• How should a “least common denominator” formalism for conflict propagation look like?  

• Do the constraint formalisms have to be disjoint?  

• Proof of completeness of the approach?
Federated to Integrated!

- SW-HW co-design/integrations: that’s just the process!

- But, what are guidelines (applied/conceptual) to
  - integrate varied functionality and dependability,
  - maintain delineation – functional and especially dependability
    (high/low criticality separations), and
  - follow constraint driven (RT, performance, power, cost …)
    composability principles?

- How do we identify “trusted”, “open”, “adaptable” building blocks?
- How do we specify (operational/meta) componentized services and interfaces?
- Is safety or resilience/dependability or time composable per se?
  Modular/incremental? What are conceptual/applied limits?
DECOS Subprojects

- **SP 1**: Architecture Design (TU Darmstadt + TU Vienna)
- **SP 2**: Component Design and Implementation (TTTech)
- **SP 3**: Silicon Infrastructure (Infineon)
- **SP 4**: Validation and Certification (ARCS)
- **SP 5**: Application Automotive (Audi)
- **SP 6**: Application Aerospace (Airbus)
- **SP 7**: Application Control (Profactor)
- **SP 8**: Training, Dissemination and Standardization (ARCS)
- **SP 9**: IP Management and Assessment (ARCS)
Subproject 1 - Architecture Design Methods

- Technology invariant interfaces
  - Platform Independent Model (PIM)
  - Platform Specific Model (PSM)
  - Hardware-Software Integration
  - Platform Interface Layer (PIL)
  - Middleware Services
  - Distributed Application Subsystem (DAS) modelling
Subproject 2: Component Design and Implementation

- Encapsulation / Diagnosis / FT-Layer
  - Encapsulated Execution Environment (partition OS)
  - Virtual Commn. Links incl. Gateways
  - Diagnostic Services
  - Optimized FT-Layer
Subproject 3 - Silicon Infrastructure (Middleware)

- Fault-Tolerance Layer (HFTL) / Event Layer (HEVL)
Development Workbench

Applications 1, 2: Safety critical; Applications 3, 4 and 5: None-critical

Board Support Package

<table>
<thead>
<tr>
<th>Appl. 1</th>
<th>Appl. 2</th>
<th>Appl. 3</th>
<th>Appl. 4</th>
<th>Appl. 5</th>
</tr>
</thead>
</table>

Time-triggered operating system supporting encapsulation

TT-Bus Controller Driver

HFTL Drivers

HEVL Drivers

TriCore-Board

SDRAM

Flash

HMI

Peripherals

Interface to FPGA-Board

Human Machine Interface

TT-Bus-Controller + HFTL + HEVL

Interface

FPGA

Flash

Physical layer

Work bench

TT-Bus

protocol dependant
Subproject 4: V&V Framework

- NOT Testing/V&V techniques by themselves - we (really should) know them by now 😊 But,

  - Integrated use of varied tools: Modular, Incremental, Composable
  - Limits of each technique
  - Do we understand the dimensions? Random FI for SW, middleware/open services?
  - “…presence of bugs vs. absence (by design/proof)”?
  - Do we even know what V&V means for open systems & services?
    Do we know how to specify open services?
V&V:
- of the design process
- of each layer (PIM,...)
- of the integration
The V&V Environment

Test-bench Set-up

Candidate Methods & Tools

Method / Tool Evaluation and Selection

Method / Tool Improvement and Development

Integration

Test-bench Methods & Tools

selected methods/tools

Design/Model
Software Tool
Software Comp.
Hardware Comp.

Verification
Validation (Verification)

Test Env. Set-up

Test Environment
Testing (in-the-loop) (Verification?)

Test-bench Usage

Reports
Certification
Certificates
## Tools & Techniques

### The DECOS Test Bench

<table>
<thead>
<tr>
<th>Methods &amp; Tools</th>
<th>System Analysis &amp; Evaluation</th>
<th>Simulation, Modelling</th>
<th>Fault Injection Techniques</th>
<th>EMI</th>
<th>Formal Verification</th>
<th>Test Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hazard Analysis, HAZOP, FMECA, FTA, ETA</td>
<td>UML, Simulink, various Techniques</td>
<td>SWIFI, EMFI, HIFI</td>
<td>Simulation and measurement</td>
<td>Proof-checking, model checking tools</td>
<td>White Box, black box, coverage, complexity, HW testing</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Partners Lead</th>
<th>ARCS, SP</th>
<th>BUTE, TUDA, TUVI, EST</th>
<th>TUVI, SP, BUTE, TUDA, ARCS</th>
<th>ARCS, SP</th>
<th>TUVI, TUDA, BUTE, Rushby (SRI)</th>
<th>ARCS, SP</th>
</tr>
</thead>
</table>

86
DECOS Application Areas

- Automotive
- Aerospace
- Railways
- Industrial Control
- Medical Systems
- Autonomous Systems

- DECOS will develop structured guidelines for domain-independant and technology independent integration.
DECOS Application (SP5): Aerospace
Flap Control Demonstration System for Airbus Outer Flap System
Integrating with DECOS Technology
DECOS Application (SP6): Automotive
Driver Assistance and Crash Warning and Avoidance Demonstration Systems
Integrating with DECOS Technology
DECOS Application (SP7): Industrial Control
Vibration Control Demonstration System for Nano Imprinting Machines

**Objectives:**
Suppression of critical vibrations in high-end nano-imprinting machines for next-generation Sensors, Microoptics, Bio- and Nanotechnology.
DECOS Timeframe

- SP1 SP2: R&D of methods and tools (prototypes)
- SP1 SP2 SP3: R&D of methods and tools (final)
- SP4: Verification & Validation of methods and tools
- SP5 SP6 SP7: Basic set-up of application subproject demonstrators / test-bench
- SP5 SP6 SP7: Integration of R&D results into demonstrators / test-bench
- SP4: V&V of demonstrators / test-bench
- SP8 SP9: Management, Dissemination, Training, ...

Timeline:
- July, 2004
- Evaluation phase (EC)
- July, 2007
DECOS Results

- Methodologies + Tools for “Composable & Integrated” Design of Systems
- Re-usable SW, HW & middleware components
- Integrated distributed execution platform
- Diagnostics concept
- Component Oriented V&V Test Bench
- Application development support

**DECOS Technology is platform independent**

- Set of certifiable HW and SW components in order to *significantly reduce* the design, deployment, and life cycle *cost* of *dependable embedded applications*.
- *Fundamental enhancement of EU’s competence in design, analysis, applied techniques and tools for integrated dependable, real-time embedded systems!*
www.decos.at

www.deeds.informatik.tu-darmstadt.de