**Overall Structure (Day 1)**

- **Introduction to modern embedded systems**
  - Ubiquitous computing as a vision for integrating future embedded systems
  - From embedded to resource constrained systems
  - Some basic techniques for constructing real-time embedded system software
- **Principled embedded software infrastructure**
  - Survey of real-time scheduling algorithms: static, dynamic priority, static priority dynamic
  - I/O processing and networking for embedded systems
Overall Structure (Day 2)

- Automotive embedded software architecture
  - Component-based software engineering
  - Case study on automotive embedded software
- Sampling of methodical optimization of embedded software
  - Specialization of system software
  - Code generation and translation
  - Aspect-oriented programming

Study of Embedded System

- Cars are reasonably stable systems
  - Several years of development, with several years of product half-life
  - Faster than airplanes (decades), slower than hand sets (months)
- Credit to Crnkovic (Malardalen University, Sweden) and their report.
Vehicular Control System

Figure 2.1 Basic elements of a control system

Functionality Evolution (99)

<table>
<thead>
<tr>
<th>Driver Assistance</th>
<th>Infotainment and Telematics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive cruise control</td>
<td>Dynamic navigation</td>
</tr>
<tr>
<td>Stop &amp; go control</td>
<td>Speech recognition</td>
</tr>
<tr>
<td>Parking assistance, Parking brake by wire</td>
<td>Multimedia entertainment</td>
</tr>
<tr>
<td>Blind spot detection</td>
<td>Internet access, E-mail</td>
</tr>
<tr>
<td>Video-based traffic sign, lane and obstacle recognition</td>
<td>Telediagnosis</td>
</tr>
<tr>
<td></td>
<td>Roadside assistance</td>
</tr>
</tbody>
</table>

Electronic Control Unit (ECU)

Figure 2.3 Mechanical embedding exemplified by the Scania diesel engine controller.

Figure 2.4 Mechanical embedding of a braking control system illustrating power supplies, power cabling, controllers, communication between controllers, and some sensor and actuator elements.
Distributed Computer System

Electronic Control Unit (ECU)
- Processor: 25MHz, 16-bit
- 128KB RAM, 1MB flash, 64KB EEPROM
- Serial interface: RS232 or RS485
- I/O: digital and analog ports
- Controller Area Network (CAN)
  - Developed by Bosch for cars (mid-80’s)
  - High resilience to transmission errors
  - Up to 1Mbit/sec
Figure 2.6 Structure of a node, part of the embedded distributed computer system. The ECU (Electronic Control Unit) is a microcontroller with one or more microprocessors, memories, input/output devices, and communication interfaces. The ECU structure is composed of application software, supporting software, input/output drivers, and hardware.

Volvo XC90
(comp. MDX, RX, etc)
**XC90 Highlights**

- Software content: 10MB to 50MB (full options)
- Base configuration
  - Dynamic Stability and Traction Control (DSTC): sensors on wheels, steering, car movement
  - Roll Stability Control with a gyroscopic sensor, using DSTC
  - 4-Channel Anti-Lock Braking System (ABS) with weight sensors and Electronic Braking Assistance (skid sensors)
- Full options
  - Forward collision warning, Blind spot detection
  - Volvo Navigational System with DVD Map and Remote Control
  - Rear view camera, Telephone/telematics

---

**Complexity of Current Cars**

*Figure 2.8. The electronic architecture of Volvo XC90.*
Main goal: Integration of components from many suppliers

Important functionality
- Diagnostic kernel and services
- Network communications software: Volcano signals, CAN, LIN
- RTOS: OSEK/VDX
- I/O devices
**VCE Requirements**

- Large construction equipment
  - Articulated haulers, excavators, graders, backhoe loaders, wheel loaders
- Requirement highlights
  - Less complex electronic systems/networks
  - Focus on safety, reliability and functionality
  - Scalability of product variation, reusability, componentization to lower software costs
  - Accommodate cheaper mechanical parts

**VCE Network Architecture**

- Flexible allocation of control units
  - Map more of simple software components to a smaller number of control units
VCE Node Architecture

- Applications are control software
- System layer contains OS and middleware

Rubus RTOS

- Green: interrupt handler, highest priority
- Red: hard real-time, static scheduler
  - Verifiable timing properties
- Blue: software real-time, preemptive fixed priority scheduler
Train Network Architecture

Figure 2.13, the train communication network

Train Node Architecture

Application Code

Basic Software
- Standalone Download
- SiMON

Run-Time System

Boot code

Processor
Train Software Architecture

- Common Software Structures (hardwired)
  - Boot code
  - SiMon (signal monitor)
  - Standalone download

- Run-Time System
  - Run-time system initialization and startup
  - Run-time services
  - RTOS kernel: VxWorks

Industrial Robotics (ABB S4)

Figure 2.15 The system architecture of a robot controller.
Robot Software Architecture

- MTBF=60,000 Hours
- 2.5 MLOC
  - 400 to 500 components
  - 15 subsystems
- Isolation: VxWorks
- System language: C
- Robot language: RAPID

Robot Execution Model

- Event-driven IPC bus, similar to HW bus
- Successfully ported several times
Development Challenges

- System integration
  - Many architectures and components
- Model-based development and architecture
- Mission-critical requirements
  - Dependability: safety, reliability, security
  - Scalability, development and evolution costs
- Component-based software engineering
- Automation vs. human factors

Summary

- Software costs will dominate
  - Moore’s law and mechanical analogs (better materials) reduce hardware size, weight, costs
  - More functionality provided by software
  - Planned obsolescence of hardware need software glues to bridge the transition
- Different architectures and interfaces
  - Car, VCE, train, robotics
  - Bottom-up evolution
Discussion

- Vehicular embedded systems
  - Application requirements: time scale, cost/benefits, functionality, extra-functionality properties
- Other applications
  - How much of the discussion will apply to airplanes, and hand sets?

Component-Based Lifecycle

- Applied to automotive embedded systems
- Component-based software engineering is feasible for vehicular systems
  - System evolution not too fast
  - Critical safety requirements
Product Lifecycle

- Idea/product requirement specification
- Development, manufacturing, maintenance
- Intellectual content/value of products
  - Higher value/functionality by weight/size
  - From bigger-is-better to smaller-is-better

**Figure 3.1. Product lifecycle**

Generic product lifecycle

<table>
<thead>
<tr>
<th>Business Idea</th>
<th>Requirements Management</th>
<th>Development</th>
<th>Production</th>
<th>Operation and Maintenance</th>
<th>Disposal</th>
</tr>
</thead>
</table>

SW/HW Comparison

**Figure 3.2. Software and hardware development processes**
Waterfall Model

- Software development follows a well-defined sequence of stages
  - Assumes each stage can freeze its output
  - Actually, each stage is a feedback loop with continual refinements; propagate downstream, amplifying its effects
  - Failures such as FBI Virtual Case File project

Waterfall + Prototyping (V Model)
Component-Based

HW/SW Differences

- Hardware developments usually start new (before VLSI methods and tools)
  - Standard interfaces (110/220V, 50-60cycles)
  - Standard components (e.g., electronic systems)
- Software developments used to start new
  - Low software component availability (open source software is recent)
  - Cultural/legacy programming practices
- Component-based software development
  - Start with components and their composition
Component-Based Lifecycle

- Requirement analysis and definition
- Component selection and evaluation
- System design
- System integration
- Verification and validation
- System operation support and maintenance

Component Development Process

Figure 3.5 shows different environments in a component life cycle.
VCE Applications

- Application area: big machines
  - Volvo Construction Equipment
  - Embedded, Real-Time software
- Design and development method
  - Formal design specification language
  - Component-Based Software Engineering
  - Software tools using the design specs
  - Implementation by hand

ABB Robotics

Figure 3.6, the main process and the sub-processes at ABB Robotics
ECU Development (VCE)

Design Process

Component model
Model transformation
Real-time model
Synthesis
Target application

Design-Time
Compile-Time
Run-Time

Real-Time Analysis

QoS Semantics
Design Tools

Data flow specifications
- Data ports with types
- Connections only with compatible types
- Ordering, end-to-end deadline, start jitter, completion jitter

Control flow specifications
- Event-based
- Mandatory control sink (to activate the module)
- Optional source (to activate others)
Component Model

- Component model design
  - Translation into run-time model
  - Fixed-priority scheduling run-time model
- Real-time analysis (e.g., schedulability)
  - Based on classic RT techniques (see below)
- Compile run-time model into code
  - Synthesis of target application and run-time
Task Allocation

- Transactions divided into tasks
  - Worst Case Execution Time (WCET) analysis
  - Jitter requirements (Start and completion jitter)
- Synchronization among tasks
  - Data flow dependencies
- Time triggered and event triggered tasks
  - Period, jitter, WCET

Schedulability Analysis

- Given a schedule (from the task allocation)
  - Test whether the schedule is feasible (tasks completing before deadline) for FPS
Code Synthesis

- Code
- Data port connections
- Task Assignment
- Assignment of Run-time attributes
- Run-Time System
- RTOS characteristics and Run-Time system
- Executable Reliable Bundle For Embedded Systems

Task Communications

- Inter Task Communication
  - Within the same tasks: shared data spaces
  - Different tasks: IPC/RPC
- Links between sink and source
  - Scheduling of periodic tasks
  - Synchronization between source and sink
  - Middleware glue code
Portability from Middleware

Middleware

Operating System 1

Middleware

Operating System n

System specification → Top-level design → Detailed design → Architecture analysis → Scheduling / interface check → Obtain components timing behavior on target platform → System verification → Final product

Component library

Create specifications for the new components

Add new components to library

Implement and verify new components using classical development methods

Figure 3.9 Development steps at VCE
Development Method (1)

I. Requirement engineering
II. Requirement analysis
III. High-level system decomposition
   - Operational modes and transitions
IV. Function decomposition and structuring
   - Optimization using interrupt handlers as efficient replacement for periodic tasks

Development Method (2)

V. Mapping temporal constraints
   - Synchronization defined by precedence
   - Period, WCET, job start, deadline
VI. Defining execution time budget
   - By experienced engineers
VII. Feasibility check and automatic implementation
   - Configuration compiler: pre-run-time scheduler
Development Method (3)

VIII. Implementation and module testing
- Write programs (modules) by hand [This may be automated.]
- Module testing: functional/timing behavior

IX. System integration and verification
- Putting it all together

Findings (Development)
1. A design language is good
2. Design language supports RT analysis and code synthesis
3. Trade-off between design time and implementation time
4. Estimated execution time budgets useful for early schedulability analysis
Findings (Tech Transfer)

5. Tools, people, and support structure needed for successful RT technology transfer
6. Major roadblock is temporal requirements (need all participants writing specs)

Findings (Technical)

7. Task model too restrictive
   - Control jitters for simple controllers
   - Multirate controllers
8. Theoretical task models and schedulers need to be extended for real-world apps
   - Schedule representation
   - Interrupt overhead (interrupt handlers used as optimized processes)
Findings (Technical)

9. User feedback for schedulability analysis and schedule modifications
   - Identification of problem and suggestions for solution

10. Incremental scheduling to minimize verification effort
    - Minor updates should incur minor costs

Discussion

- Component-based approaches
Introduction to CBSE

- Component-based software engineering as “the newest mature method”
- Application of component-based software engineering to vehicular systems

Component-Based Software Development

- Component-Based Software Engineering
  - Design philosophy
  - Reusing off-the-shelf building blocks

- Business of heavy vehicles (VCE)
  - Need low cost hardware
  - High reliability in the total system
  - High degree of customization, small volume
  - Need flexible software development
Basic Concepts

Figure 4.1. Relations between basic concepts of component technology

CBSE Lifecycle

Figure 4.2 Component technology for embedded systems
**Vehicle Network Architecture**

![Vehicle Network Architecture Diagram](image)

**Technical Requirements**

- Analysable (composability)
- Testable and debuggable
- Portable (HW, RTOS, networking)
- Resource constrained
  - Low cost hardware
  - Efficient software (comparable to non-CBSE)
- Component modeling
- Computational model
Development Requirements

- Introducible (migration)
- Reusable
- Maintainable
- Understandable
  - For developers (simplify evaluation and verification)
  - For users (improve acceptance)

PECT

- Prediction-Enabled Component Technology

*Figure 4.3, UML class diagram of PECT concepts*
Prediction Enabled Component Technology

The component technology ensures that components & assemblies satisfy reasoning framework assumptions.

We use a **composition language** to specify well-formed assemblies:
- currently is a text-based language
- nearly isomorphic with UML 2.0
Koala

- Philips: consumer electronics
  - Light-weight component model
  - Mainly static composition of source code

- Interfaces
  - Component Description Language (bounding interface): Provides, Requires.
  - Interface Definition Language (API)
Rubus

- Rubus OS consists of three kernels achieving an optimum solution.
- The Red Kernel, which is very small, manages execution of pre-run-time scheduled time-triggered Red threads
- The Blue Kernel is dedicated for execution of event-triggered Blue threads
- The Green Kernel is dedicated for execution of event-triggered Green threads (External Interrupts)
- Basic Services contains common services for the three kernels
Rubus Meta-Model

Figure 4.6, A UML meta-model of the relations between different items

Rubus Visual Studio
Rubus OS Simulator

Application Interface

Port Based Objects

Figure 4.7, Architectural view of a port based object
**PBO Example**

![Diagram showing PBO example](image)

*Figure 4.8, a control application based on PBO*

---

**PECOS**

- Pervasive Component Systems
  - ABB (ASEA Brown Boveri)
  - University of Karlsruhe (Germany)
  - OTI (Netherlands)
  - University of Berne (Switzerland)
- EC-IST project
  - € 2.5M in two years (2000-2002)
Focus of Project

- Component Specification
- Technical Innovation
- Component Repository
- Composition Environment
- Process Innovation
- Product Innovation
- Ultra Light Component Environment
- Paradigm Shift

Figure 2 PECOS contribution to the different aspects of innovation

PECOS Compo. Structure

Figure 4.9, UML diagram showing the structure of a component
PECOS Example

Figure 4.10, A composite component

IEC 61131-3 Standard

4.11, a graphical view of the elements covered in the IEC 61131
IEC 61131 Example

```plaintext
FUNCTION_BLOCK
("External Interface "
EXAMPLE

BOOL - IN O OUT
TIME - DB_TIME - ET_OFF

("Function Block Body "

A - X Y
B - V T

A - DB_TIME - X Y

END FUNCTION BLOCK
```

CORBA Architecture

Client

Operation ()

Object (Servant)

IDL Skeletons

ORB Interfaces

Object Adapter

ORB Core

IDL Stubs
## Comparison & Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Analyzable</th>
<th>Testable and debuggable</th>
<th>Portable</th>
<th>Resource Guaranteed</th>
<th>Component Modelling</th>
<th>Computational Model</th>
<th>Introducible</th>
<th>Reusable</th>
<th>Maintainable</th>
<th>Understandable</th>
<th>Source Code Components</th>
<th>Static Configuration</th>
<th>Average</th>
<th>Number of 2's</th>
<th>Number of 0's</th>
</tr>
</thead>
<tbody>
<tr>
<td>PECT</td>
<td>2</td>
<td>NA</td>
<td>2</td>
<td>NA</td>
<td>NA</td>
<td>2</td>
<td>NA</td>
<td>NA</td>
<td>0</td>
<td>NA</td>
<td>1.2</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Koala</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>NA</td>
<td>1.3</td>
<td>7</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rubus Component Model</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>NA</td>
<td>1.3</td>
<td>5</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PBO</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>NA</td>
<td>0.9</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PECOS</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1.4</td>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CORBA Based Technologies</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.4</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>1.2</strong></td>
<td><strong>1.0</strong></td>
<td><strong>1.0</strong></td>
<td><strong>1.2</strong></td>
<td><strong>0.0</strong></td>
<td><strong>1.4</strong></td>
<td><strong>1.4</strong></td>
<td><strong>1.2</strong></td>
<td><strong>1.0</strong></td>
<td><strong>1.5</strong></td>
<td><strong>1.2</strong></td>
<td><strong>1.1</strong></td>
<td><strong>4.3</strong></td>
<td><strong>3.5</strong></td>
<td></td>
</tr>
</tbody>
</table>

## Development Challenges

- Advanced features vs. limited resources (light-weight implementation)
- Standard specification of component properties and interfaces
- Obtaining and predicting extra-functional component properties: timing, performance, dependability, etc
- Platform and vendor independence (standards), quality certification, isolation
- CBSE Tools
Research Priorities

- Predicting system properties
  - Given component properties
  - Predict integrated system properties
  - Guaranteeing system properties

- Component models for real-time systems
  - Widely accepted model and specification
  - Generating run-time infrastructure, contract monitors, and other shared functionality

Discussion

- CBSE as an advanced software engineering discipline
- New systems building techniques
  - Web services and services computing (SOA)
  - Code generation and AOP