Summer Institute on
Software Architecture

Embedded Systems Architecture 3: Vehicular Case Study

Instructor: Calton Pu
calton.pu@cc.gatech.edu

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

Georgia Tech

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.

Georgia Tech

Vehicular Control System

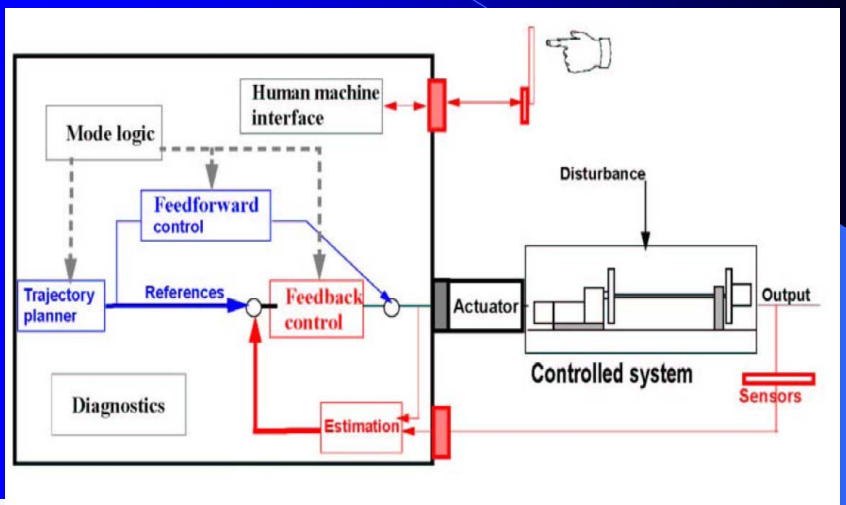


Figure 2.1 Basic elements of a control system

Functionality Evolution (99)

Driver Assistance
Adaptive cruise control
Stop & go control
Parking assistance, Parking brake by wire
Blind spot detection
Video-based traffic sign, lane and obstacle recognition

Infotainment and Telematics
Dynamic navigation RL'05
Speech recognition MDX'03
Multimedia entertainment
Internet access, E-mail
Telediagnosis
Roadside assistance OnStar'00

Lexus LS460 self-park feature (2007).

Electronic Control Unit (ECU)

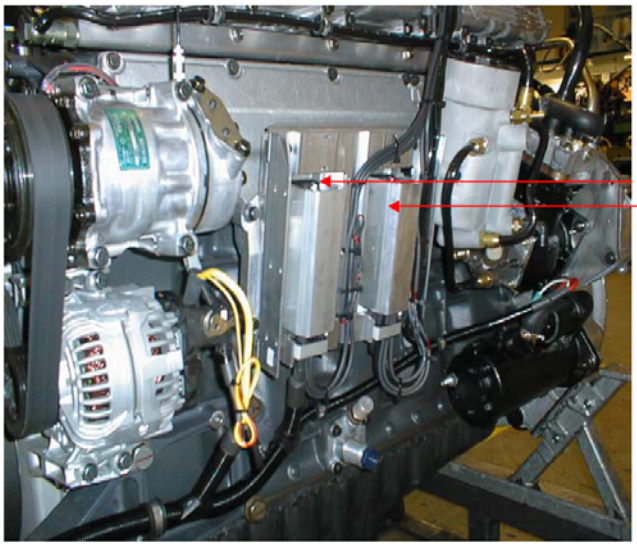


Figure 2.3 Mechanical embedding exemplified by the Scania diesel engine controller

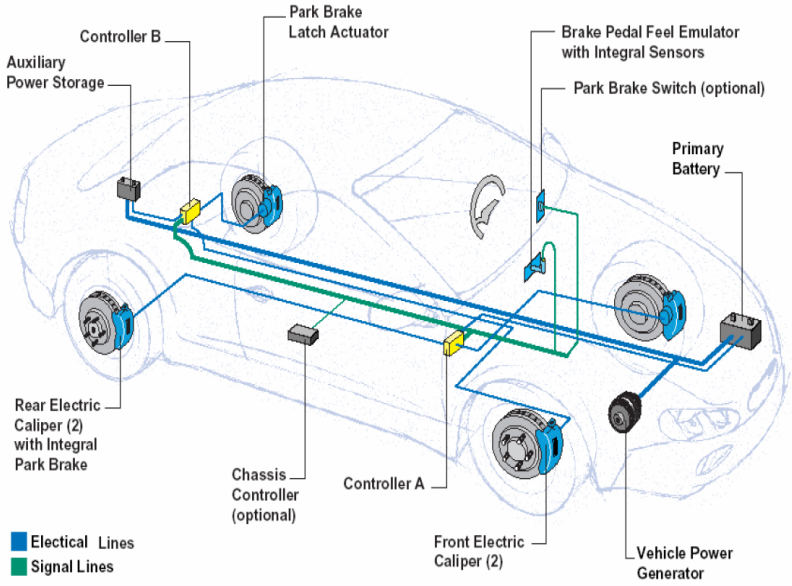


Figure 2.4 Mechanical embedding of a braking control system illustrating power supplies, power cabling, controllers, communication between controllers, and some sensor and actuator elements.

Distributed Computer System

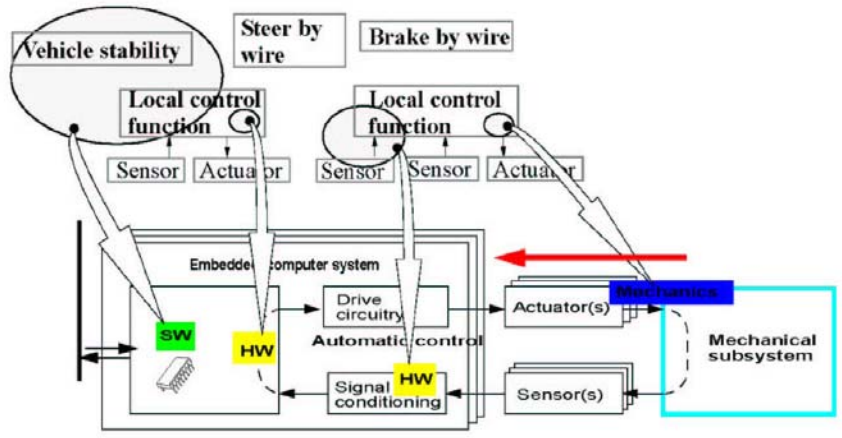


Figure 2.5 Mapping of functionality to a hardware architecture including mechanical and electronic components, where the electronic components form a distributed computer system

Typical ECU

- 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

ECU Structure

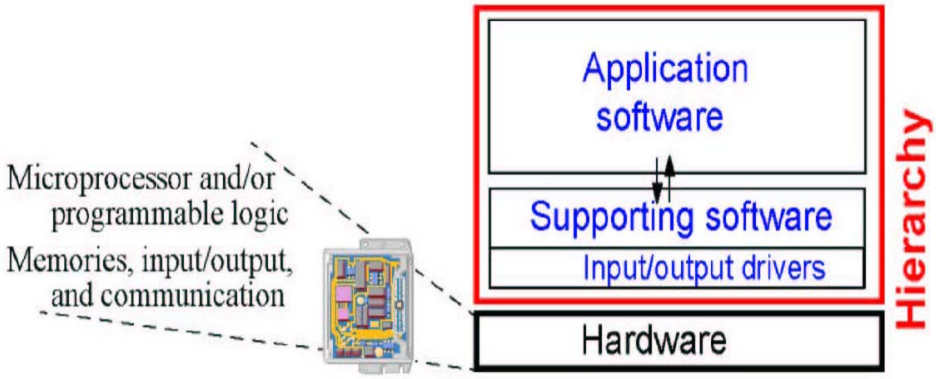


Figure 2.6 *Structure of a node, part of the embedded distributed computer system. The*

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

Georgia Tech

Complexity of Current Cars

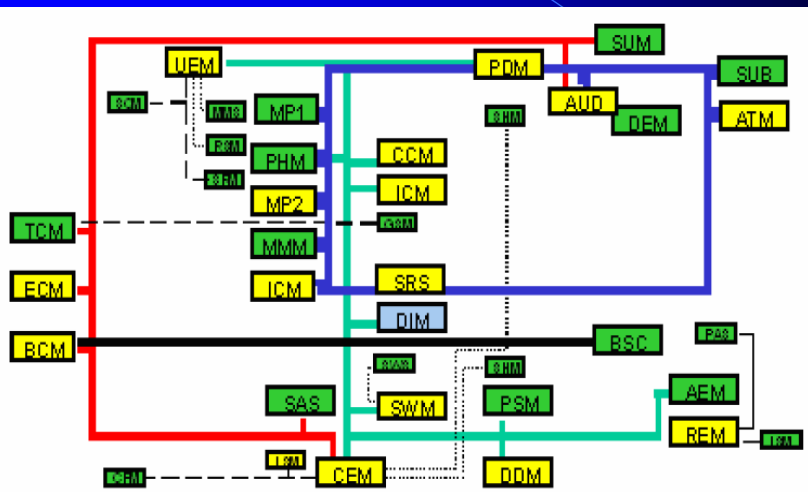


Figure 2.8. The electronic architecture of Volvo XC90.

Car Node Architecture

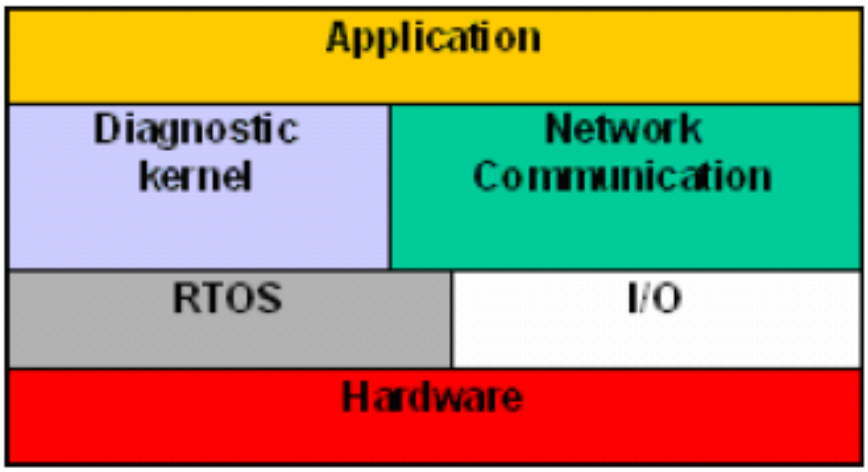


Figure 2.9 The node architecture.

Architecture Components

- 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

Georgia Tech

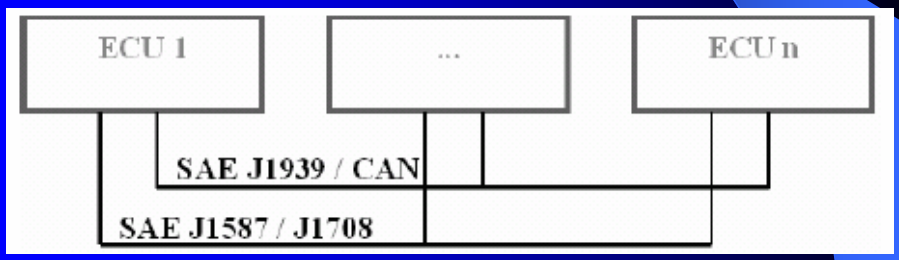
VCE Requirements

- 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

Georgia Tech

VCE Network Architecture

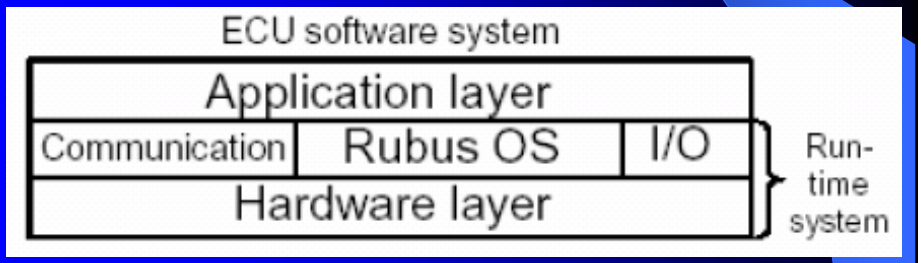
- Flexible allocation of control units
  - Map more of simple software components to a smaller number of control units



Georgia Tech

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

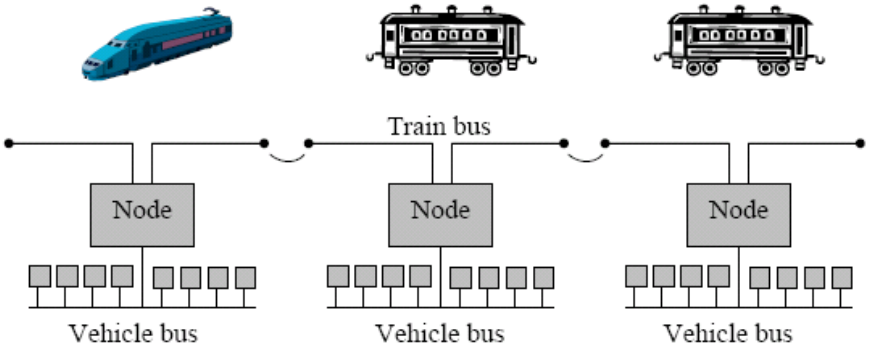
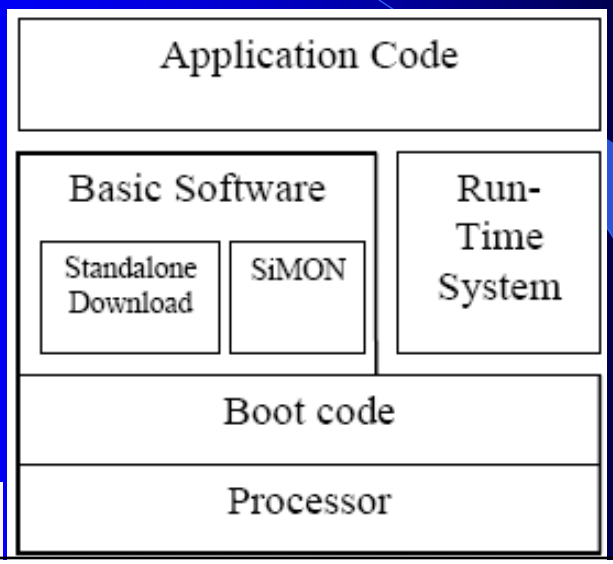


***Figure 2.13**, the train communication network*

Train Node Architecture



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

Georgia Tech

Industrial Robotics (ABB S4)

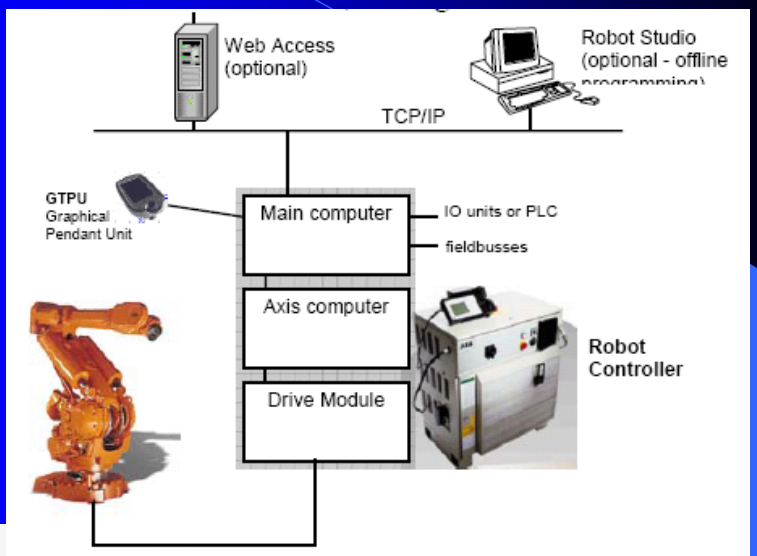
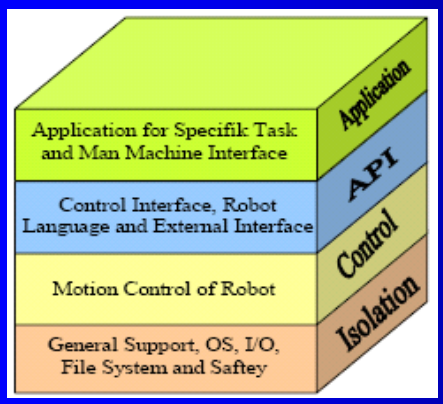


Figure 2.15 The system architecture of a robot controller.

Georgia Tech

Robot Software Architecture

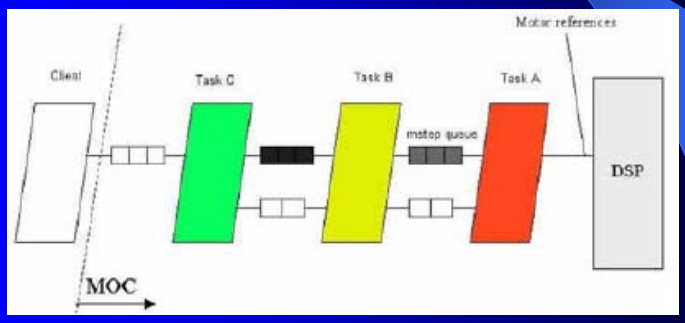


- MTBF=60,000 Hours
- 2.5 MLOC
  - 400 to 500 components
  - 15 subsystems
- Isolation: VxWorks
- System language: C
- Robot language: RAPID

Georgia Tech

Robot Execution Model

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



Georgia Tech

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

Georgia Tech

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

Georgia Tech

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?

Georgia Tech

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

Georgia Tech

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

Generic product lifecycle

Figure 3.1. Product lifecycle

SW/HW Comparison

Software Development

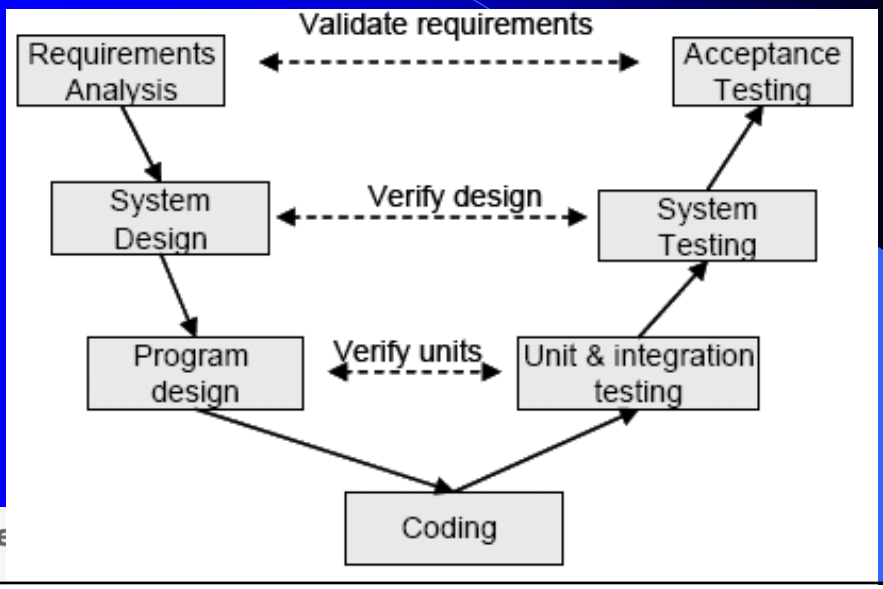
Hardware Development

Figure 3.2. Software and hardware development processes

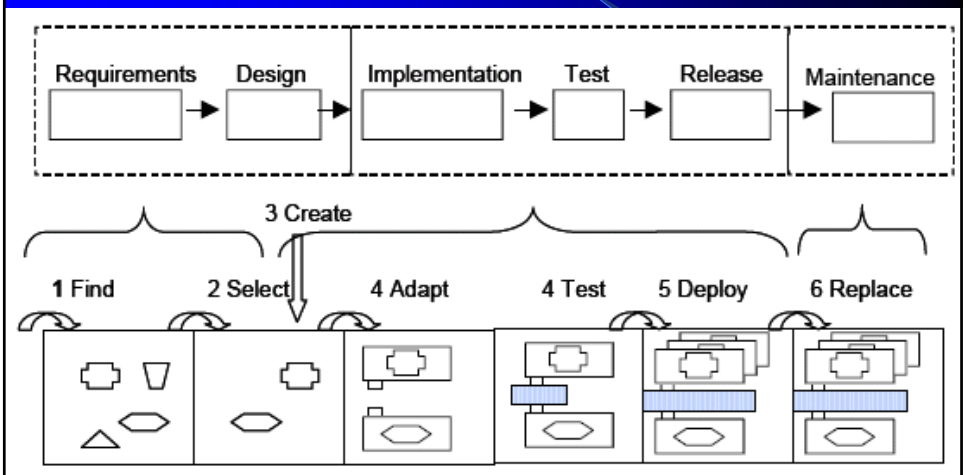
Waterfall Model

- 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

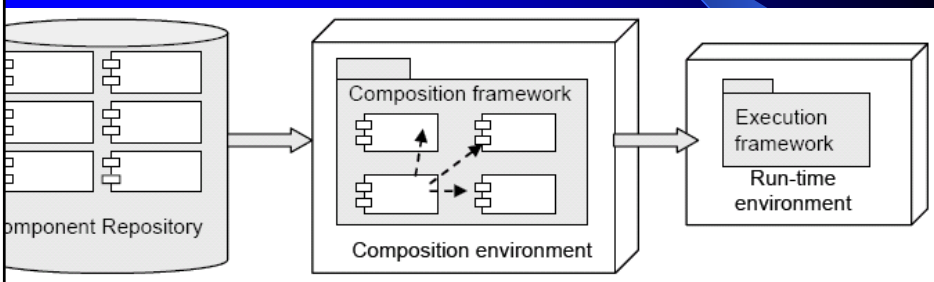


Figure 3.5 shows different environments in a component life cycle.

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

Georgia Tech

ABB Robotics

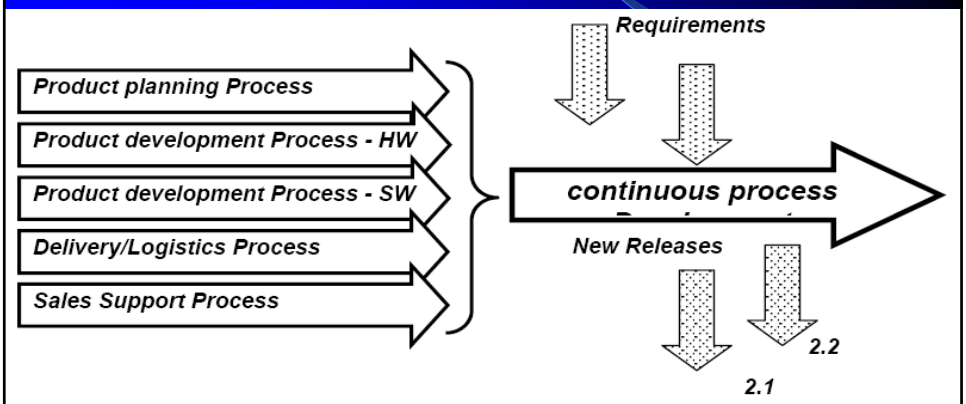


Figure 3.6, the main process and the sub-processes at ABB Robotics

Georgia Tech

ECU Development (VCE)

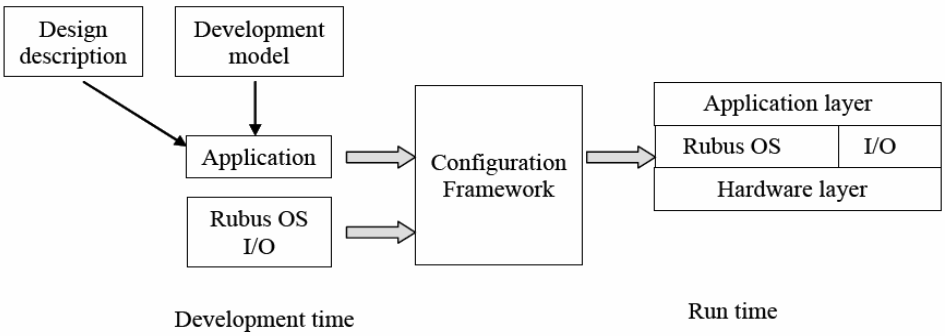
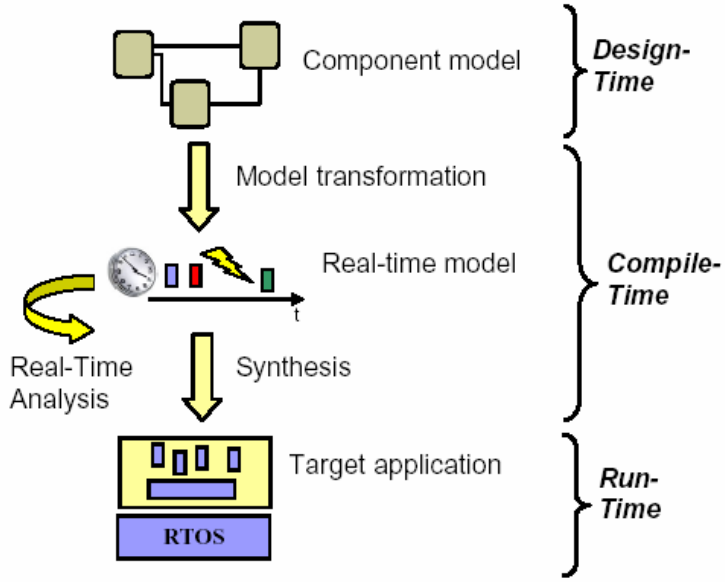
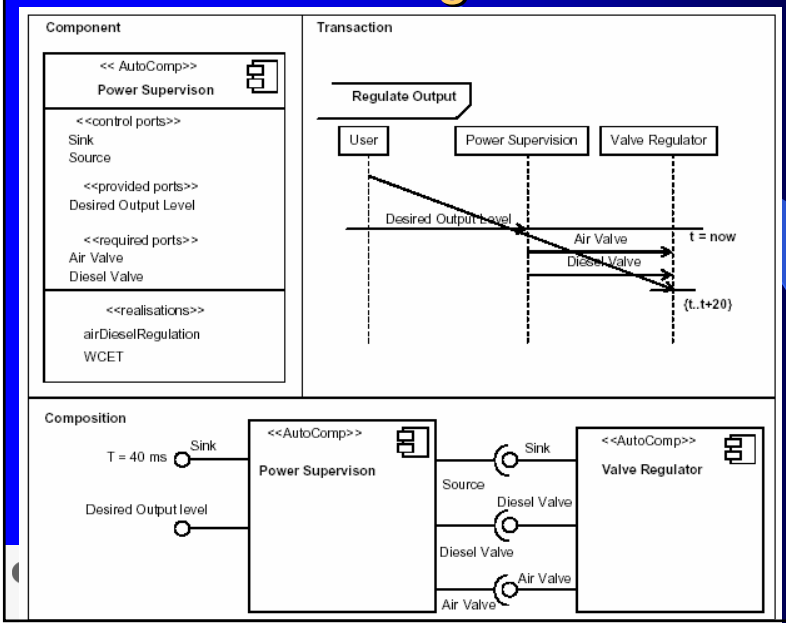


Figure 3.8 Development and run time frameworks for an ECU

Design Process



Design Tools

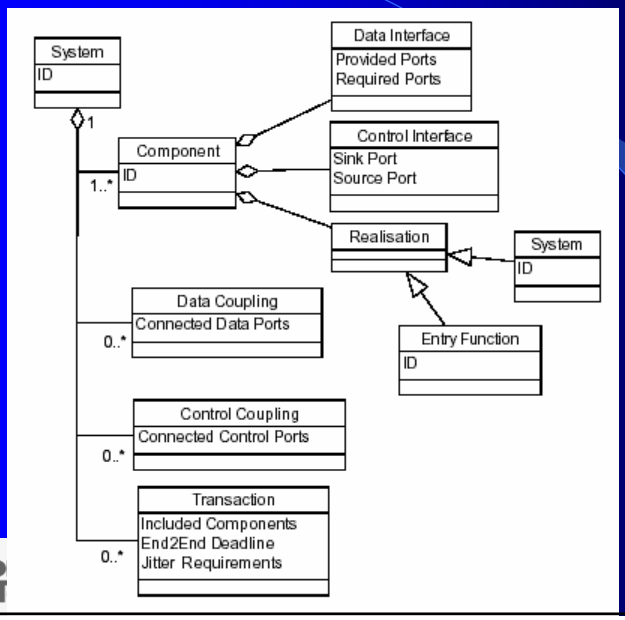


Design Specs

- 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)

Georgia Tech

Component Model



Model Translation

- 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

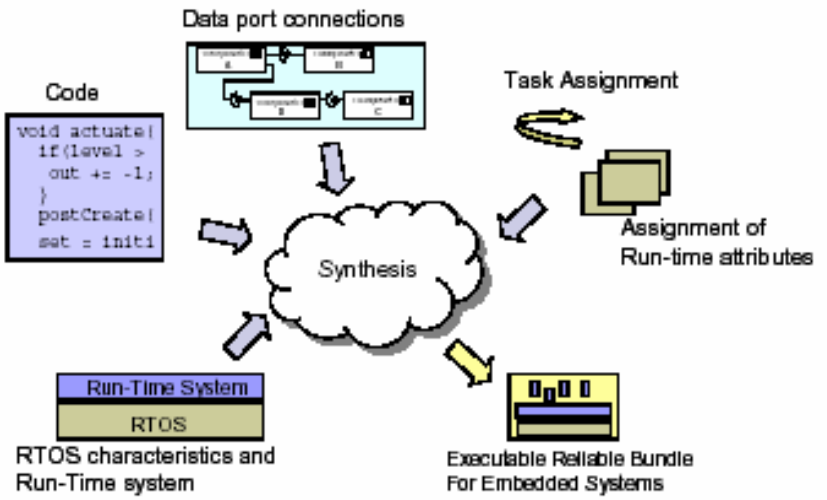
Georgia Tech

Schedulability Analysis

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

Georgia Tech

Code Synthesis

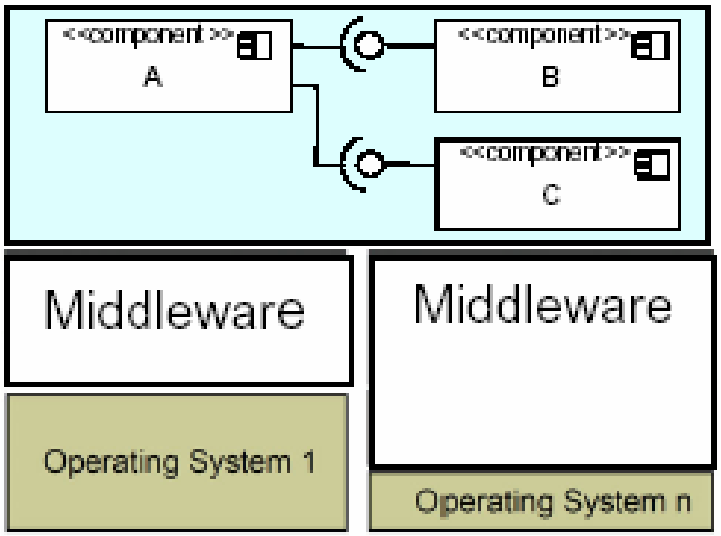


Task Communications

- 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

Georgia Tech

Portability from Middleware



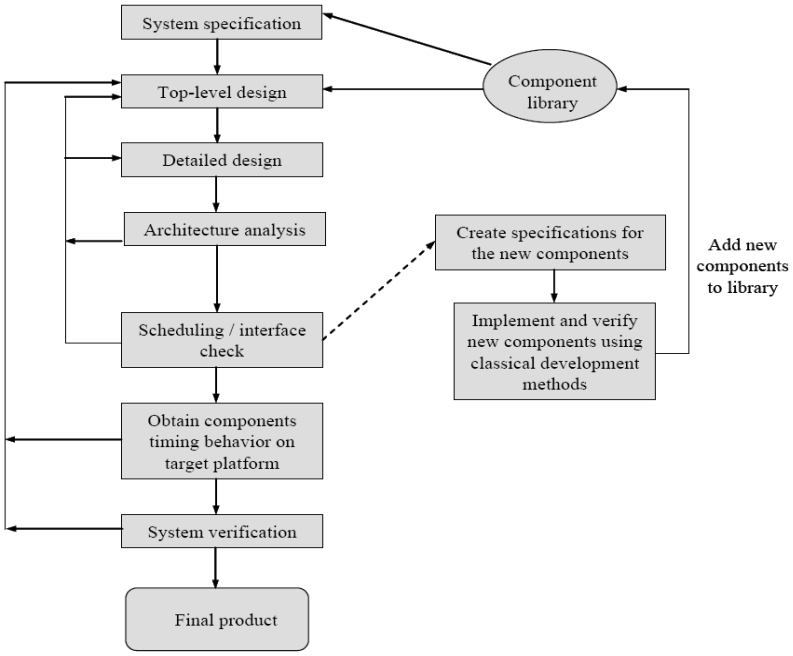


Figure 3.9 Development steps at VCE

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

Georgia Tech

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

Georgia Tech

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

Georgia Tech

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

Georgia Tech

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)

Georgia Tech

Findings (Technical)

7. Task model too restrictive
   - Control jitters for simple controlles
   - 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)

Georgia Tech

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

Georgia Tech

Discussion

- Component-based approaches

Georgia Tech

Introduction to CBSE

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

Georgia Tech

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

Georgia Tech

Basic Concepts

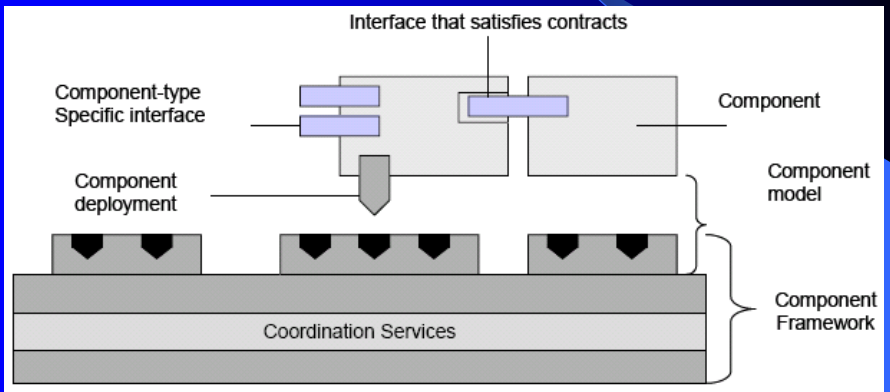


Figure 4.1. Relations between basic concepts of component technology

CBSE Lifecycle

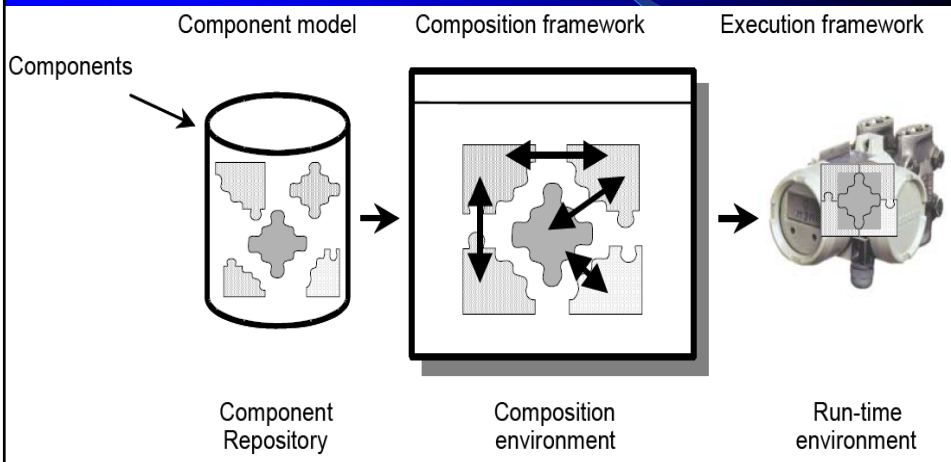
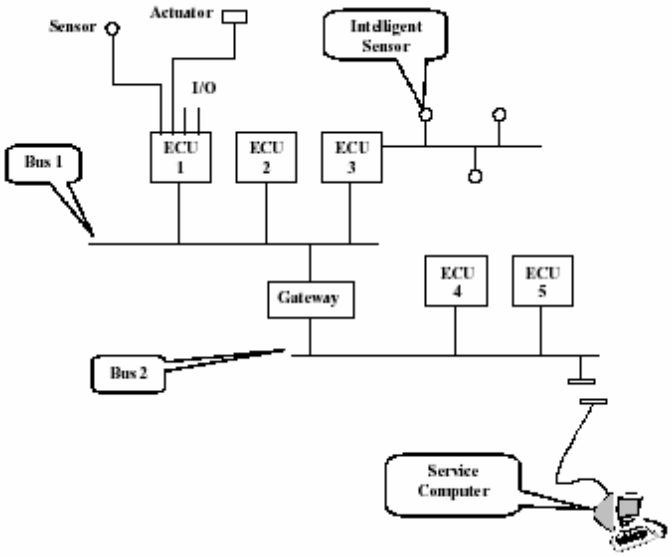


Figure 4.2 Component technology for embedded systems

Vehicle Network Architecture



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

Georgia Tech

Development Requirements

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

Georgia Tech

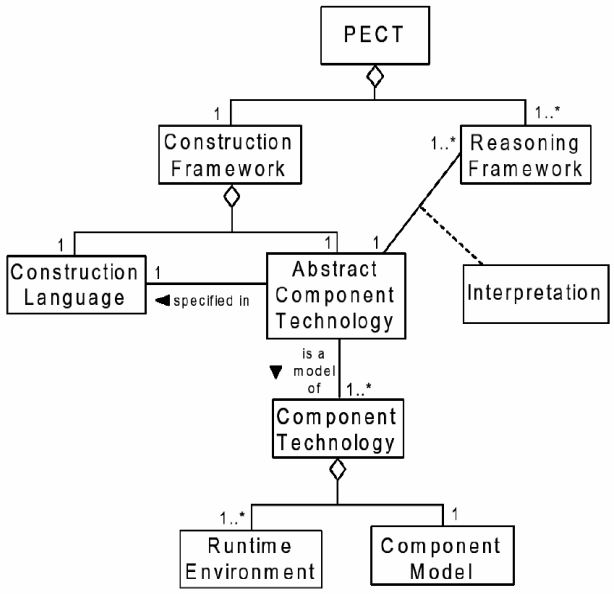


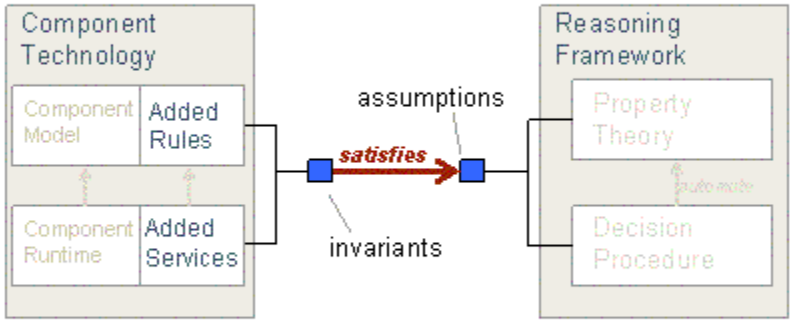
Figure 4.3, UML class diagram of PECT concepts

PECT

- Prediction-Enabled Component Technology

Carnegie Mellon
Software Engineering Institute

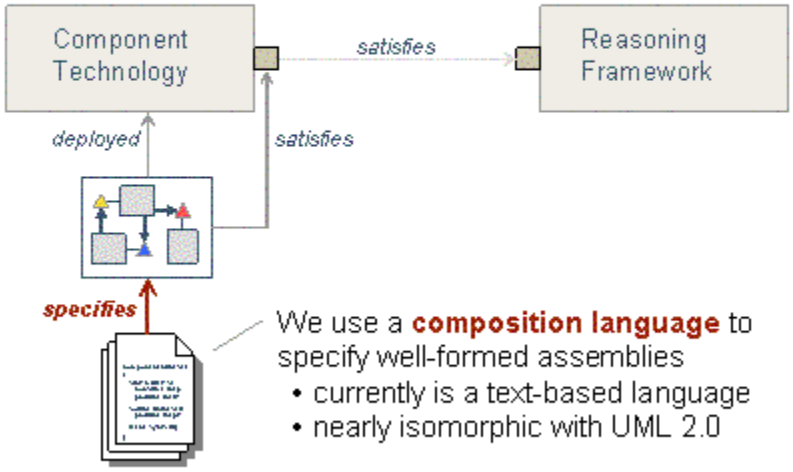
Prediction Enabled Component Technology 8



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

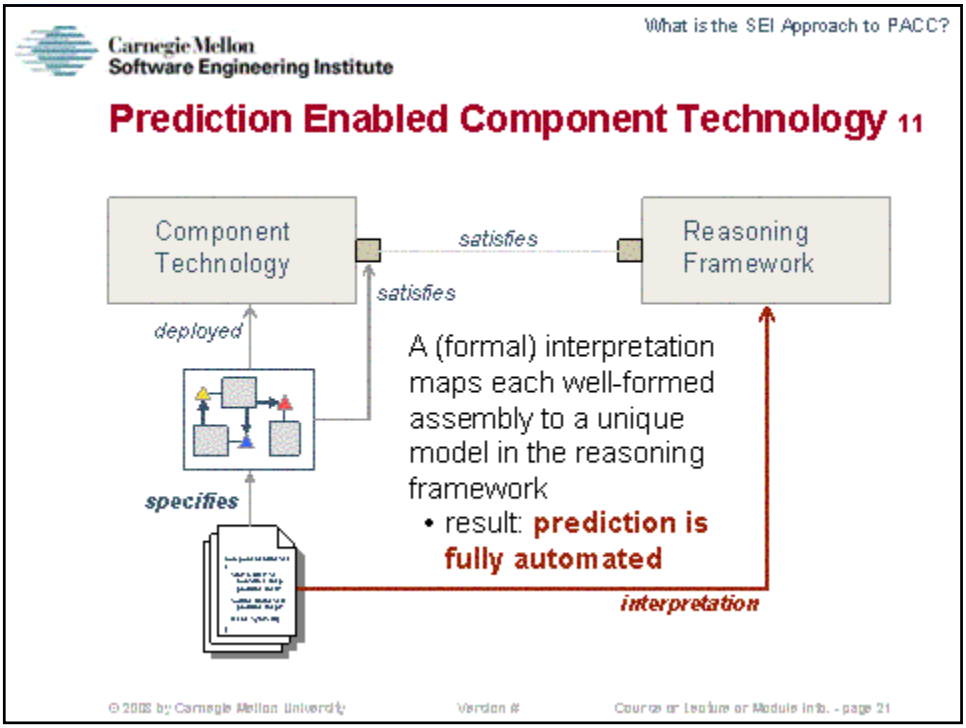
Carnegie Mellon
Software Engineering Institute

Prediction Enabled Component Technology 10



We use a **composition language** to specify well-formed assemblies

- currently is a text-based language
- nearly isomorphic with UML 2.0

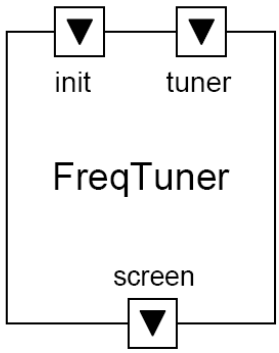


Koala

- 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)

Georgia Tech

Koala Component Example




```
component FreqTuner{
provides  IInit init;
          ITuner tuner;
requires  IScreen screen;
}
```

```
interface IInit{
void initialize(void);
}
```

```
interface ITuner{
int getFrequence(void);
void setFrequency(int
f);
}
```

```
interface IScreen{
void updatePicture
(void);
}
```

Figure 4.5, a Koala component

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

Georgia Tech

Rubus Meta-Model

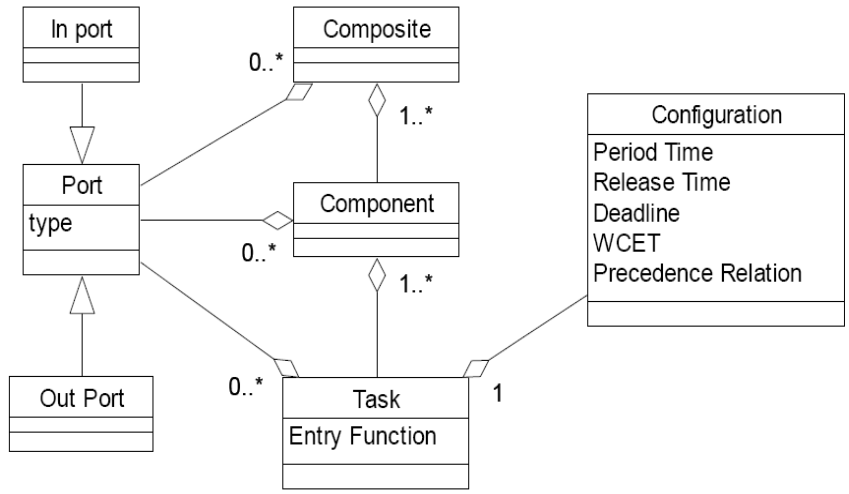
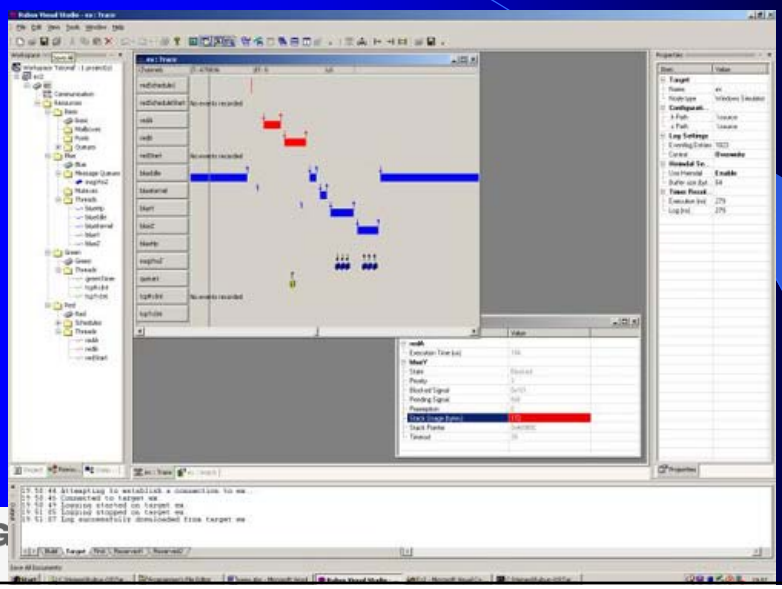
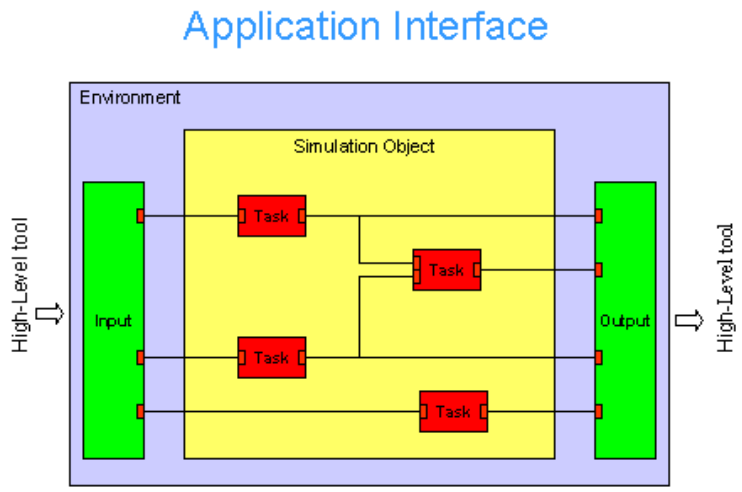


Figure 4.6, A UML meta-model of the relations between different items

Rubus Visual Studio



Rubus OS Simulator



Port Based Objects

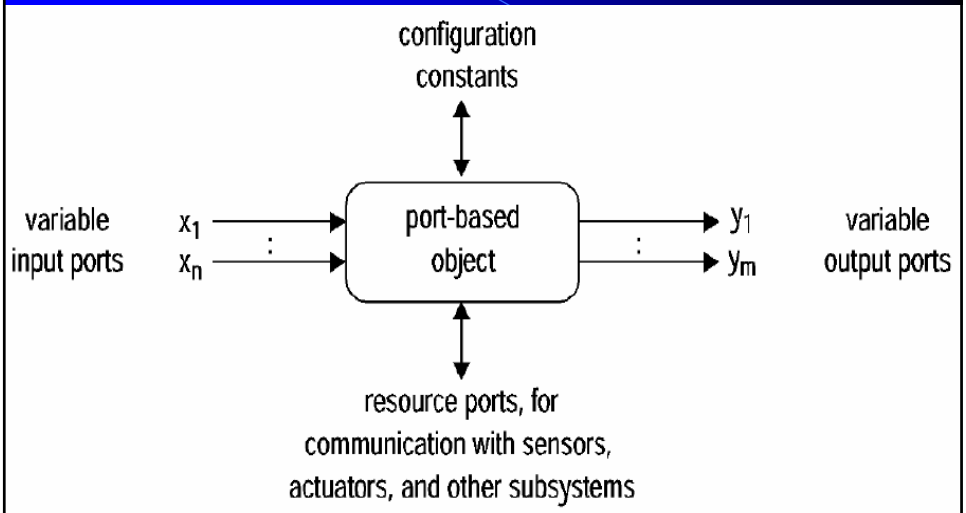


Figure 4.7, Architectural view of a port based object

PBO Example

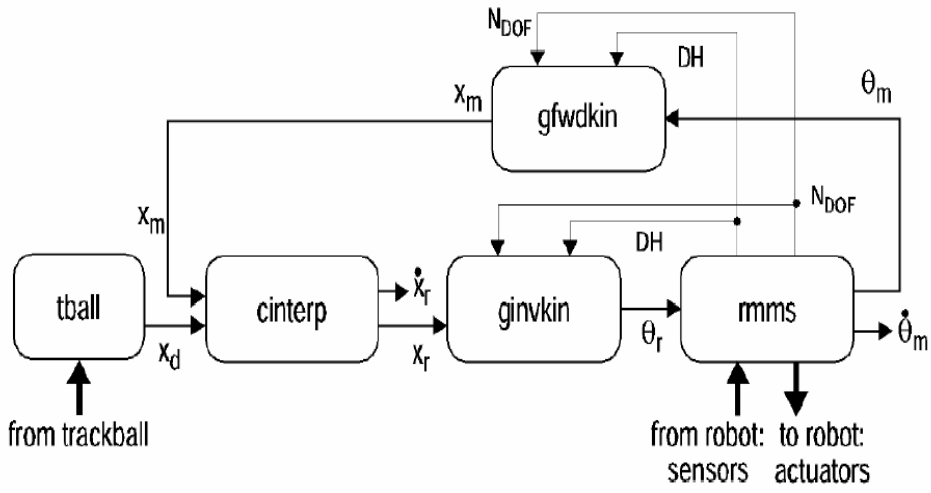


Figure 4.8, a control application based on PBO

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)

Georgia Tech

Focus of Project

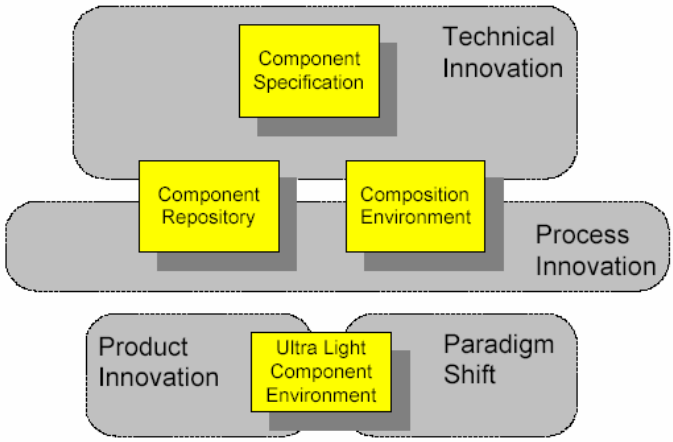


Figure 2 PECOS contribution to the different aspects of innovation

PECOS Compo. Structure

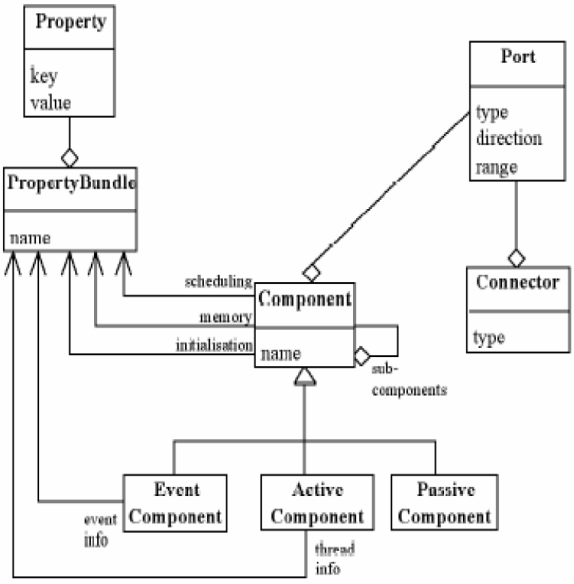


Figure 4.9, UML diagram showing the structure of a component

PECOS Example

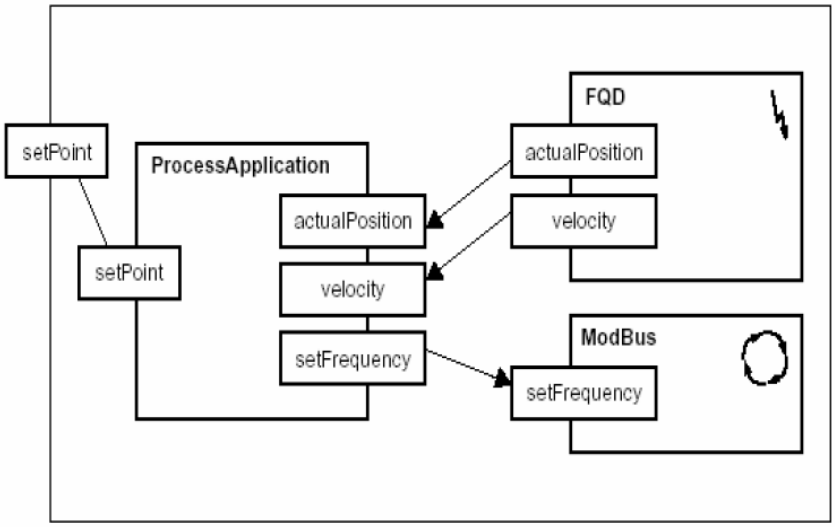
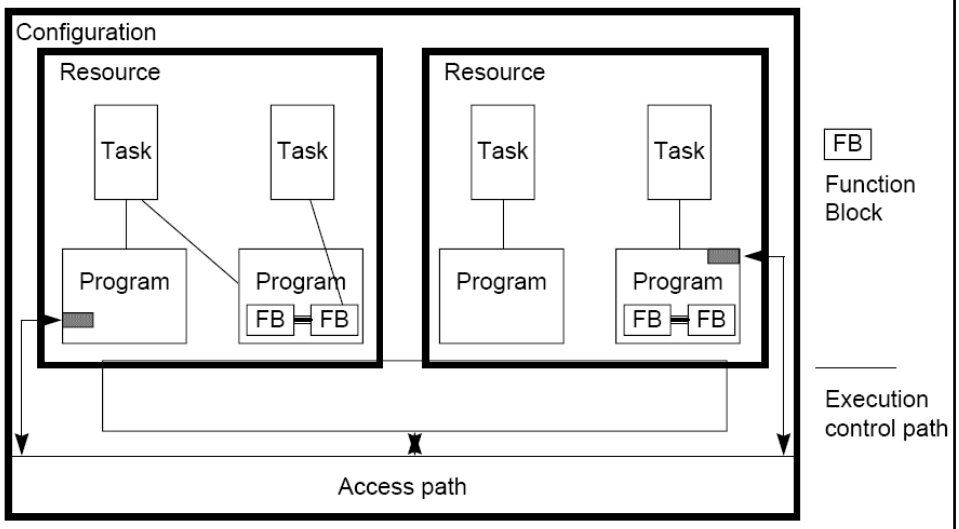


Figure 4.10, A composite component

IEC 61131-3 Standard

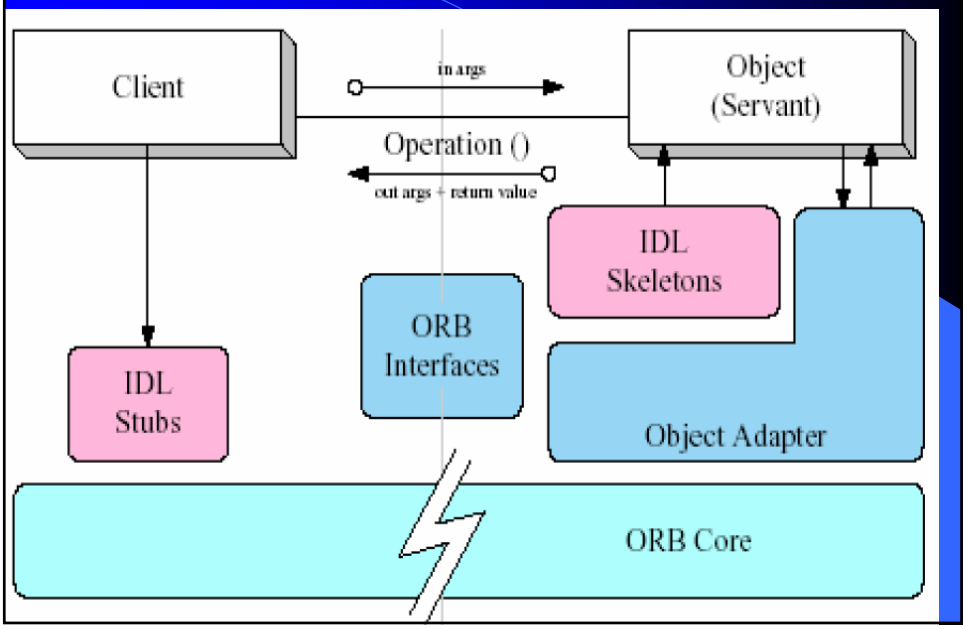


4.11, a graphical view of the elements covered in the IEC 61131

IEC 61131 Example

```
FUNCTION_BLOCK
(** External Interface **)

                 EXAMPLE
BOOL -----  IN                OUT  -----BOOL
TIME -----  DB_TIME        ET_OFF  -----TIME

(** Function Block Body **)

                A                B
IN -----  X        Y  -----  V        Z  ----- OUT
                             T

                A
DB_TIME -----  X        Y
               O  ---------------------- ET_OFF

END_FUNCTION_BLOCK
```


CORBA Architecture



Comparison & Evaluation

	Analysable	Testable and debugable	Portable	Resource Constrained	Component Modelling	Computational Model	Introducible	Reusable	Maintainable	Understandable	Source Code Components	Static Configuration	Average	Number of 2's	Number of 0's
PECT	2	NA	2	NA	0	NA	2	NA	NA	0	NA	NA	1.2	3	2
Koala	0	1	1	2	0	2	0	2	2	2	2	2	1.3	7	3
Rubus Component Model	1	1	0	2	0	2	1	1	1	2	2	2	1.3	5	2
PBO	2	1	0	0	0	1	1	1	1	2	2	0	0.9	3	4
PECOS	2	1	2	2	0	2	1	2	1	2	0	2	1.4	7	2
CORBA Based Technologies	0	1	1	0	0	0	2	0	0	1	0	0	0.4	1	8
Average	1.2	1.0	1.0	1.2	0.0	1.4	1.4	1.2	1.0	1.5	1.2	1.2	1.1	4.3	3.5

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

Georgia Tech

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

Georgia Tech

Discussion

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

Georgia Tech