Software Modeling and Design for Real-Time Embedded Systems

- COMET/RTE method
  - Software Design for RT embedded systems
  - From Use Case Models to Software Architecture
    - Uses UML, SysML and MARTE notations
  - Requirements and Analysis Modeling
    - Use case modeling
    - Static and Dynamic modeling
  - Design modeling
    - Concurrent, distributed, real-time embedded systems
Software Modeling and Design for Single Systems

- COMET: General software modeling and design method
- Requirements and Analysis Modeling
- Software design modeling
  - Develop software architecture using architectural design patterns
    - Object-Oriented Software Architectures
    - Client/Server Software Architectures
    - Service-Oriented Architectures
    - Component-Based Software Architectures
    - Concurrent and Real-Time Software Architectures
    - Software Product Line Architectures

Software Modeling and Design for Software Product Lines

- Software Product Line (SPL)
  - Family of products / systems (Parnas, Weiss, SEI)
- Software Modeling and Design for SPL
  - Model commonality and variability among members of SPL
    - PLUS method for SPL
      - Extends COMET, other methods for single systems
      - Integrate Feature Modeling with UML
      - Unifying View of Multiple-View Modeling Approach
      - Apply standard UML extension mechanisms
  - H. Gomaa, *Designing Software Product Lines with UML*, Addison Wesley, 2005
### Comparison of COMET and COMET/RTE

<table>
<thead>
<tr>
<th>COMET</th>
<th>COMET/RTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>General design method on software modeling and design</td>
<td>Focused design method on real-time software design</td>
</tr>
<tr>
<td>Has one chapter on each kind of software architecture. Real-time software architectures covered in one chapter.</td>
<td>Focus entirely on the design of real-time embedded systems, including real-time design patterns.</td>
</tr>
<tr>
<td>Main in-class case study is client/server system: Banking System.</td>
<td>Focus on real-time embedded system case studies.</td>
</tr>
<tr>
<td>Does not address issues specific to RT systems</td>
<td>Addresses issues specific to RT systems:</td>
</tr>
</tbody>
</table>

- Address systems engineering issues.
- Design of hardware/software interface
- More details on state machine modeling
- Component-based RT software design
- Real-Time scheduling and performance analysis
- Quality of Service
- Dynamic RT software adaptation

---

### Unified Modeling Language (UML)

- A standardized notation for object-oriented development
- A graphical language for describing the products of OO requirements, analysis, and design
- Approved as a standard by Object Management Group (OMG)
- Methodology independent
- Needs to be used with an analysis and design method
MARTE

- Modeling and Analysis of Real-Time Embedded systems
- UML profile
  - Extension of UML for a specific application domain
- MARTE
  - Profile supports concepts for real-time embedded systems

SysML

- Systems Modeling Language
  - Standardized notation for modeling system requirements
  - Approved as a standard by Object Management Group (OMG)
  - Methodology independent
- General-purpose graphical modeling language
  - specifying, analyzing, designing, and verifying complex systems
    - Hardware
    - software
    - information
    - personnel
    - Procedures
Model Driven Architecture

- Promoted by Object Management Group (OMG)
- Model Driven Architecture
  - Develop UML models of software architecture before implementation
- Platform Independent Model (PIM)
  - Precise model of software architecture before commitment to specific platform
- Platform Specific Model (PSM)
  - Map PIM UML model to a specific hardware/middleware platform
    - E.g., .NET, J2EE, Web Services, Real-Time platforms
- Real-time systems need to be mapped to PSM for performance analysis

Real-Time Systems

- Hard real-time systems
  - Time-critical deadlines
  - System failure could be catastrophic
  - Safety critical systems
- Soft real-time systems
  - Interactive systems
  - Missing deadlines is undesirable but not catastrophic
- Real-Time Embedded System
  - Component of larger hardware/software system
  - Has mechanical or electrical parts
    - E.g., aircraft, automobile, train
Real-Time Embedded Systems and Applications

- Real-Time Embedded System
  - Real-Time Embedded Application
    + Real-Time Operating System
    + Computer Hardware

![Layered architecture of real-time embedded system](image)

Figure 1.2: Layered architecture of real-time embedded system

Characteristics of Real-Time Embedded Systems

- Interaction with external environment
  - Input from sensors
  - Output to actuators
- Timing constraints
  - Must process input event within given time
- Real-time control
  - Make control decision based on input data
  - Without human intervention
- Reactive systems
  - System responds to external events
  - Response is often state dependent
- Concurrency
  - Many events happening in parallel
Measuring Time

• **Event**
  – Occurs at an instant of time

• **Duration**
  – Interval of time between
    • starting event
    • terminating event

• **Period**
  – Measurement of recurring intervals of same duration

• **Execution time**
  – CPU time taken to execute a given task

• **Elapsed time**
  – Total time to execute a task from start to finish
# Measuring Time

- **Elapsed time** = Execution time + Blocked time
- **Blocked time**
  - Waiting time when the task is not using the CPU
    - Waiting for I/O operations to complete
    - Waiting for messages or responses to arrive
    - Waiting to be assigned the CPU
    - Waiting for entry into critical sections
- **Physical time** (or real-world time)
  - Total time for a real-time command to be completed
  - E.g., to stop a train
  - Physical time = Elapsed time + time to physically stop train

---

# Real-Time Control

- Consider as a process control problem
- Speed control algorithm of automated train
  - Set point: target cruising speed
  - Controlled variable: current speed of train
- Speed control algorithm
  - Compares set point with controlled variable
  - Increasing or decreases the current speed of train
  - Goal: current speed = cruising speed +/- delta
- Adjustments to speed converted to voltage applied to electric motor
- Speed sensor measures current speed of train
Real-Time Control

- Distributed processing environment
  - Multiple computers communicating over network
- Typical applications
  - Distributed real-time data collection
  - Distributed real-time control
  - RT Client / server applications

Characteristics of Distributed RT Applications
Distributed RT Embedded Systems

- Distributed RT Embedded System
  - Distributed RT Embedded Application
    + Middleware
    + RT Operating System
    + Communication Software
    + Computer & Network Hardware
  - Communication Software = Network protocol software
- Middleware
  - Software layer provides uniform platform
    - Distributed OS often integrates middleware with OS
Advantages of Distributed RT Processing

- **Distributed Control**
  - Control distributed among multiple nodes
  - Hierarchical control
  - Peer-to-peer control
- **More localized control and management**
  - Design distributed subsystem to be autonomous
  - Subsystem on node
    - Relatively independent of other subsystems on other nodes
- **More flexible configuration**
  - A given application can be configured in different ways
Advantages of Distributed RT Processing

• **Improved availability**  
  – Operation is feasible in a reduced configuration  
  – There is no single point of failure  
• **Incremental system expansion**  
  – System can be expanded by adding more nodes  
• **Load balancing**  
  – Overall system load can be shared among several nodes

Internet of Things (IoT)

• Interconnect physical “things” to the Internet  
• Connect remote sensors and actuators to the Internet  
• Remote access to sensor data  
  – RFID technology  
  – Electronic RFID tag is attached to a physical product  
  – Product + RFID  
    • Smart object uniquely identified over Internet  
• IoT  
  – Integrate real-time embedded systems with the Internet
Cyber-Physical Systems

- Smart networked systems with embedded sensors, processors and actuators
- Designed to sense and interact with the physical world
- Support real-time, guaranteed performance in safety-critical applications
- Joint behavior of “cyber” and “physical” elements
- Embedded cyber system
  - Monitors and controls physical processes
- E.g., automated train
  - Cyber subsystem: Real-time automated control
  - Physical subsystems: electric motor, braking system, transmission, smart sensors and actuators

Software Design Concepts for Real-Time Systems

- Concurrent tasks
  - For structuring system into components that execute in parallel
  - Key concept for designing concurrent, real-time, and distributed systems
- Finite state machines
  - Key concept for defining control aspects of real-time systems
- Information hiding
  - For structuring system into modifiable components
  - Key concept for object-oriented design
Sequential & Concurrent Problems

Sequential problems
Activities happen in strict sequence
E.g., compiler, payroll
Sequential solution = program

Concurrent problems
Many activities happen in parallel
E.g., multi-user interactive system, air traffic control system
Sequential solution to concurrent problem increases design complexity

Concurrency

- Characteristics of concurrent task
  - A.k.a. (lightweight) process, thread
    - Active object, concurrent object
  - One sequential thread of execution
  - Represents execution of
    - Sequential program
    - Sequential part of concurrent program
  - Concurrent system
    - Many tasks execute in parallel
    - Tasks need to interact with each other
Active and Passive Objects

- Objects may be **active** or **passive**

  - **Active object**
    - Concurrent Task
    - Has thread of control

  - **Passive object**
    - a.k.a. Information Hiding Object
    - Has no thread of control
    - Operations of passive object are executed by task
    - Operations execute in task’s thread of control
      - Directly or indirectly

- Software Design terminology
  - **Task** refers to active object
  - **Object** refers to passive object

---

**UML notation for messages**

- Simple message
  - No decision yet made about message type

- Asynchronous (loosely coupled) message communication
  - UML 1.3

- Synchronous (tightly coupled) message communication
  - UML 1.4 and in UML 2.0

- Synchronous message with reply
  - Option 1: message name (argument list, out argument list)
  - Option 2: message name (argument list)
  - Reply
Examples of Concurrent Processing

Figure 3.7 Asynchronous message communication

Task Communication via Information Hiding Object

- Passive object
  - Encapsulates data
  - Hides contents of data structure
  - Data accessed indirectly via operations
- Passive object accessed by two or more tasks
  - Operations must synchronize access to data
  - Use semaphore or monitor object
Interaction Between Concurrent Tasks

- Mutual exclusion
  - Two or more tasks need to access shared data
  - Access must be mutually exclusive
- Binary semaphore
  - Boolean variable that is only accessed by means of two atomic (indivisible) operations
  - acquire (semaphore)
    - if the resource is available, then get the resource
    - if resource is unavailable, wait for resource to become available
  - release (semaphore)
    - signals that resource is now available
    - if another task is waiting for the resource, it will now acquire the resource

Run-Time Support for Concurrent Tasks

- Operating System Services
  - Multi-tasking Kernel
    - Task creation and deletion
    - Priority pre-emption task scheduling
    - Mutual exclusion using semaphores
    - Inter-task synchronization using events
    - Inter-task communication using messages
- Language Support for Concurrent Tasks, e.g., Java
  - Concurrent tasking constructs
  - Task creation and deletion
  - Support for inter-task communication and synchronization
Task Scheduling Algorithms

- Round-robin scheduling
  - Tasks have same priority
  - FIFO queuing and CPU allocation of tasks on Ready List
  - Task executes for time slice or blocks
  - NOT satisfactory for Real-Time System
- Priority pre-emption task scheduling
  - Each task is assigned a priority
  - Task(s) with highest priority assigned to CPU
  - Task executes until
    - it blocks or
    - is pre-empted by higher priority task

State Machine
Execution Cycle of Concurrent Task

- Ready State
  - Task on Ready List
- Executing State
  - Task is removed from Ready List and assigned CPU
- Blocking States – Task blocks and is
  - Waiting for I/O
  - Waiting for Event
  - Waiting for Message
  - Waiting to Enter Critical Section
Finite State Machines

- Many information and real-time systems are state dependent
  - Action depends not only on input event
  - Also depends on state of system
- Finite State Machine
  - Finite number of states
  - Only in one state at a time
- State
  - A recognizable situation
  - Exists over an interval of time
- Event
  - A discrete signal that happens at a point in time
  - Causes change of state
Information Hiding

Each object hides design decision
E.g., data structure
interface to I/O device
Information hiding object
Hides (encapsulates) information
Accessed by operations
Basis for Object-Oriented Design
Advantage
Objects are more self-contained
Results in more modifiable -> maintainable system

Example of Information Hiding

• Example of Analog Sensor Repository class
• Information hiding solution
  – Hide internal data structure and internal linkage
  – Specify operations on data structure
  – Access to class only via operations
    – readAnalogSensor
    – updateAnalogSensor
Example of Information Hiding

Goals of Real-Time Design Method

- Capability of structuring system into concurrent components
  - Concurrent task structuring
- Development of maintainable and reusable software
  - Information hiding
  - Inheritance
- Definition of system control and sequencing
  - Finite state machines
  - Event sequence scenarios
- Component-based software architecture
  - Concurrent OO components and connectors
- Capability to analyze performance of design
  - Real-Time scheduling
Requirements for Real-Time Software Design Method

- **Structural modeling**
  - Model problem domain, system (hardware and software) boundary, software system boundary

- **Dynamic (behavioral) modeling**
  - Model interaction sequences between system and software artifacts

- **State machines**
  - React to external events given current state of system

- **Concurrency**
  - Model activities that execute in parallel with each other

Requirements for Real-Time Software Design Method

- **Component-based software architecture**
  - Concurrent object-oriented components and connectors,
    - components deployed to different nodes in distributed environment

- **Performance analysis of real-time designs**
  - Analyze the performance of the real-time system before implementation
  - Determine whether the system will meet its performance goals.
Characteristics of RT Embedded Systems

Reactive systems
- Control decisions are often state dependent
  - Finite state machines
Concurrent inputs from many sources
- Concurrent Processing
Real-time requirements
- Need to analyze performance of design
  - Real-Time scheduling
Develop maintainable and reusable software
- Need to integrate RT technology with modern software engineering concepts and methods
  - RT Software Engineering

Figure 4.1 COMET/RTE life cycle model