# Modern Software Design Methods for Concurrent and Real-Time Systems

Hassan Gomaa

200611506
이종훈

# Concurrent Processing

- Concurrent Tasks
- Mutual Exclusion
- Synchronization of Tasks
- Message Communication

# Concurrent Tasks

- 연속적인 Concurrent 프로그램의 컴포넌트 또는 연속적인 프로그램의 수행을 의미
- 소프트웨어 시스템의 **동시성**은 비동기의 다중 프로세스가 서로 다른 속도를 가질 때 얻는다
- 대부분의 Concurrent 시스템에서 Task는 동격의 처리를 요구한다.
- 사용되는 예

  OS, Real-Time System, Interactive System, Distributed System, Parrallel System, Simulation Application

# Concurrent Tasks

- Concurrent 프로그램을 설계 할 때는 아래의 사항들을 고려 해야 한다.
  - Mutual Exclusion
  - Synchronization of Tasks

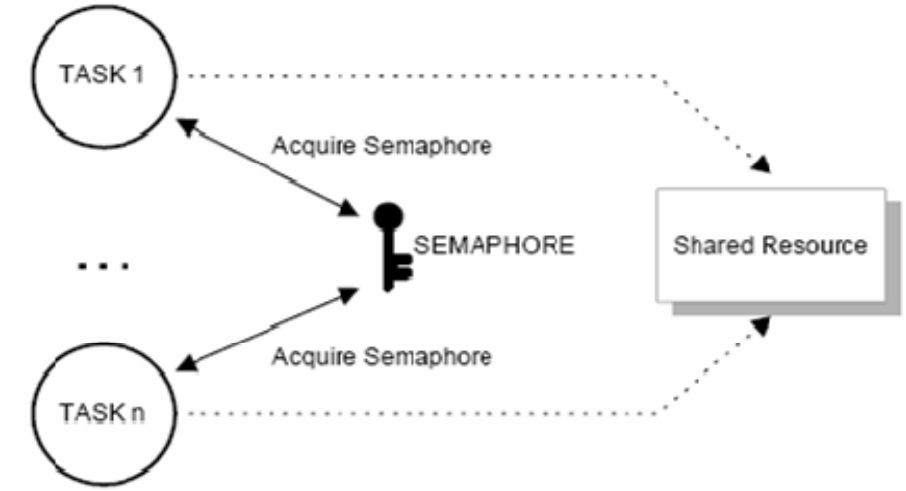
# Mutual Exclusion

- 하나의 task가 주어진 시간내에 리소스에 독점적인 접근을 할 필요가 있을 때 발생

- 각각의 서로 다른 공용자원의 사용을 막는다는 의미에서 상호배제(Mutual Exclusion)라고 부른다.

# Mutual Exclusion

- Semaphore
  - Mutual Exclusion을 해결하기 위한 방법
  - Binary Semaphores
- 공유자원을 접근하기 위해서는 semaphore를 얻어야 하고 접근이 끝나면 다른 task가 사용할 수 있도록 semaphore를 반환하는 기술



# Mutual Exclusion

Perform operations outside collision zone

P(Collision_Zone_Semaphore)
Perform mutually exclusive operations
V(Collision_Zone_Semaphore)

Perform more operations outside collision zone

# Synchronization of Tasks

- 두 개의 작업이 서로간에 통신할 데이터 없이 협업할 필요가 있을 때 사용
- 서로 신호를 보내 작업을 동기화 시킨다.
- ex)
  - 생산자가 데이터를 완성하면 소비자에게 신호를 보낸다.
  - 만약 소비자가 신호를 받지 못했다면 도착할때까지 신호를 기다린다. 신호를 받게 되면 데이터를 받는다.

# Synchronization of Tasks

Robot A
While Work_Available DO

Pick up part
Move part to workplace
Release part
Move to safe position
Signal(part_ready)
Wait(part_completed)
Pick up part
Remove fromworkplace

END

robot B
While Work_Available DO

Wait(part_ready)
Move to workplace
Drill four holes
Move to safe pos ition
Signal(part_completed)

# Message Communication

- 동기화 문제를 해결하기 위한 하나의 방법

- 약하게 결합된 메시지 커뮤니케이션 (Loosely coupled message communication)
  - Asynchronous message communication
  - 생산자는 소비자에게 메시지를 보내고, 그 반응을 기다리지 않고 계속해서 작업을 수행한다.

# Message Communication

- 강하게 결합된 메시지 커뮤니케이션
  (Tightly coupled message communication)

  - Synchronous message communication
    - 생산자는 소비자에게 메시지를 보내고 반응을 기다린다.
      - With reply : 생산자는 반응이 올 때까지 기다림
      - Without reply : 소비자가 정보를 받았는지만 확인

# Message Communication

Vision System:

- Wait(car_arrived)
- Tak picture of car body
- Identify car body
- Determine location and orientation of car body
- Send message to robot (car model,ofset)

Robot Task:

- Wait for message
- Read message(mdoel,ofset)
- Select welding program for car mdoel
- Excute welding program using ofset for car position Signal

# Run-Time Support for Concurrent Tasks

- 아래의 사항들에 의해 제공되어 진다.
  - Kernel of an OS
    - Concurrent Processing을 위한 서비스를 제공
  - Run-time support system
    - Concurrent 언어를 위해서
  - Thread Package
    - 무거운 프로세스를 다루는 서비스를 제공

# Language Support

- C/C++/Pascal/Fortran 같은 언어는 concurrent task를 위한 기능을 지원하지 않는다. 대신 커널 또는 쓰레드를 이용하여 개발이 가능하다.

- Ada/Java 는 concurrent programming이 가능한 언어

  (Task communication과 Synchronization 구조체를 지원)

# Real-Time Operating Systems

- Real-Time OS에 반드시 필요한것
  - Multi-tasking
  - Priority preemption scheduling
  - Task synchronization 과 Communication 메커니즘
  - Memory-lociking
    - 자동으로 메모리가 지워지지 않게 방지하는 기술
  - Priority inheritance 메커니즘
    - 하위 우선순위 프로그램에 의해 상위 우선순위 프로그램이 기다려서는 안됨
  - Predictable behavior

    Ex) Task context switching, Task synchronization, Interrupt handing
    System load를 하면서 예상되는 반응 시간을 알 수 있을 것이다.

# Design Methods

- Concurrent, Real-time System 디자인 방법
  - MASCOT
  - RTSAD (Real-Time Structured Analysis and Design
  - DARTS (Design Approach for Real-Time Systems)
  - JSD (Jackson System Development)
  - CODARTS (Concurrent Design Approach for Real-Time Systems)
  - Octopus
  - ROOM (Real-Time Object-Oriented Modeling)

# The COMET Method

- COMET (Concurrent Object Modeling and Architectural Design Method)
- concurrent applications을 개발할때 사용
- UML 기법을 사용
- Life-cycle은 굉장히 반복적이다.

# The COMET Method

- Highly iterative software life-cycle
  - Requirements modeling
  - Analysis modeling
  - Design modeling

# Requirements Modeling

- UML 유스케이스(use case)가 만들어짐
  - Use case : 시스템을 기능 위주로 파악하는데 유용한 모델
- 시스템을 입력과 출력만 있는 블랙박스로 보면서 시스템의 기능적인 관점에서 파악
- Requirements Modeling을 하는데 있어 Actor는 아래의 사항들이 될수가 있다.

(Actor는 입력을 받거나 출력을 받는 요소)

  - 사용자
  - 외부 시스템
  - 입출력 장치
  - 타이머

# Analysis Modeling with UML

- 여기서 시스템의 정적, 동적모듈이 만들어진다

- 정적 모듈링(Static Modeling)
  - System Context : 시스템과 외부환경 사이의 인터페이스
  - UML은 System context 다이어그램을 지원하지 않는다
  - 그러나 정적모델과 협력모델에 의해 그려질수 있다.
  - System context 클래스 다이어그램은 유스케이스 다이그램 보다 좀더 시스템의 경계의 세부적인 관점을 제공해준다.

# Object Structuring

- 비슷한 객체들을 그룹화 하기 위해 객체를 분류한다.
- UML streotypes은 다른 종류의 어플리케이션 클래스들을 구분하기위해 사용
- Streotype : 존재하는 모델링 요소의 서브 클래스
  - 아래와 같은것들이 될수 있다.
    - 엔트리 클래스
    - 인터페이스 클래스
    - 컨트롤 클래스
    - 어플리케이션 로직 클래스

# Dynamic Modeling

- Concurrent,realtime 시스템에서는 매우 중요한 과정이다.
- State-dependent dynamic analysis
  - ) state-dependent use case와 관계된 객체들 사이에서 인터렉션을 보냄
- State chart diagram
  - 하나의 객체를 대상으로 생존기간 동안 가질 수 있는 객체 상태의 변화를 분석한 다이어그램
- Collaboration diagram
  - 객체들 사이의 관계를 다룸(정적인측면

# Design Modeling

- 공학적인 디자인 모델이 만들어진다.
- Transition From Analysis to Design
- Software Architecture Design
- Concurrent Collaboration Diagrams
- Architectural Design of real-time Systems
- Task Structuring
- Detailed Software Design

# Transition From Analysis to Design

- Collaboration diagram
  - 각각의 유스 케이스에 의해 만들어진다
  - Consolidate Collaboration diagram
    - 유스 케이스를 지원하게 만들어진 모든 collaboration diagram의 종합이다.
    - robustness analysis와 유사함.
    - 규모가 큰 시스템에서는 매우 큰것을 얻을 수 있다.
    -

# Software Architecture Design

- Software Architecture Design을 하면서 시스템은 Subsystem과 그것이 정의된것들간의 인터페이스가 분류시킨다.
- 이것의 목적은 다른 시스템에서는 약하게 결합하고 같은 시스템에서는 강하게 결합하는 객체를 얻기 위함이다.

(Subsystem : a Composite object)

# Concurrent Collaboration Diagrams

- active object는 그자신의 컨트롤 쓰레드와 다른 객체간에 동시성을 가진다.

- passive object는 다른 객체에 의해 실행된다.

# Architectural Design of Distributed Real-Time Systems

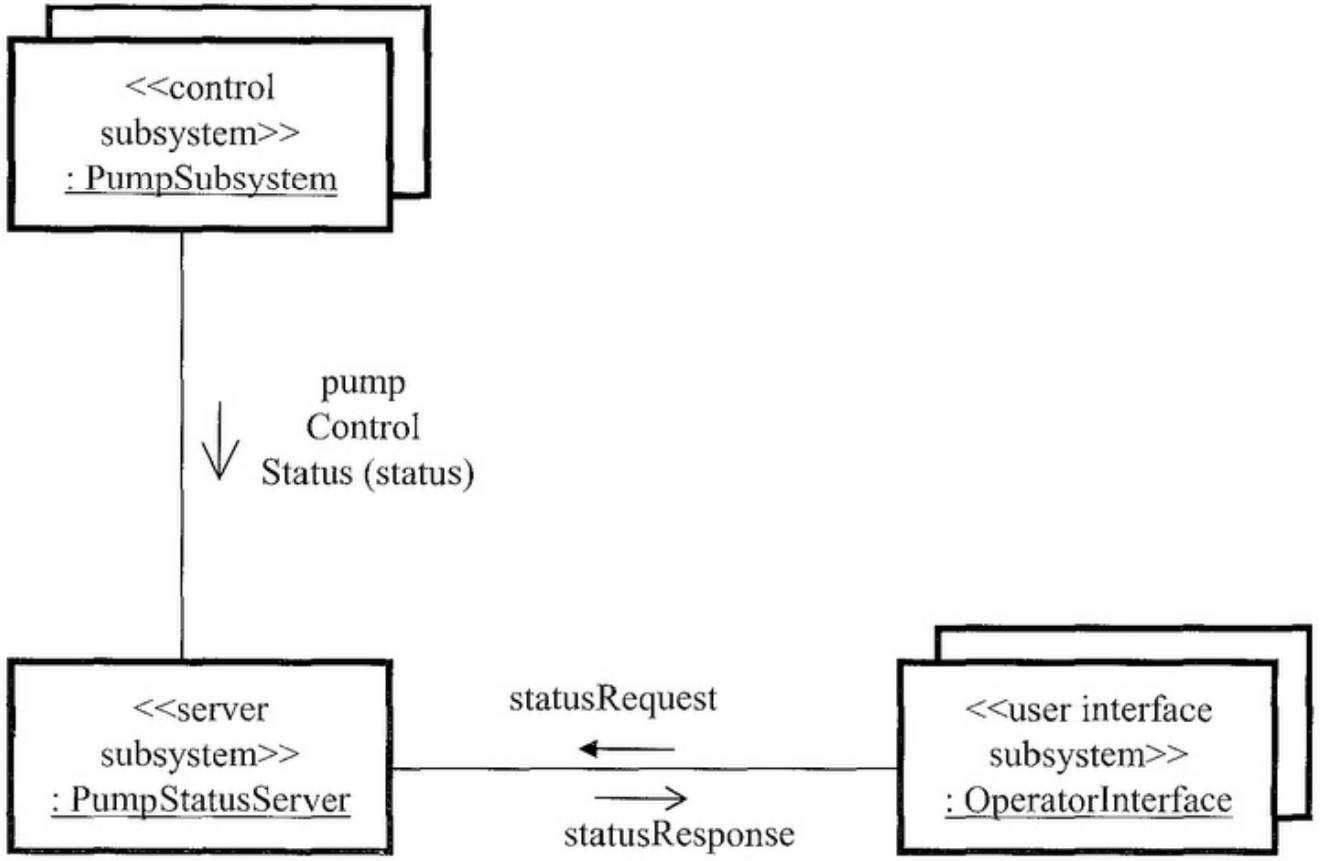



**Figure 5.** Distributed software architecture.

# Task Structuring

- 각 Subsystem은 Concurrent task로 구축된다
- Task Structuring Criteria
  - 객체들로부터 Task들을 유도하는 기준
  - Streotypes
    - I/O device interface
      인터럽트를 생성하는 디바이스를 추상화
    - Periodic I/O device interface
      폴링되는 입력 디바이스를 추상화
    - Passive I/O device interface
      수동적인 출력 디바이스를 추상화
    - resource monitor
      여러 소스로부터 출력 요청을 받아 순차적으로 처리하는 디바이스를 추상화 ex) 라인 프린터

# Detailed Software Design

- Composite task의 본질을 디자인한다
- 세부적인 Task 동기화에 초점을 맞춘다
- 내부 통신의 세부적인것들이 캡슐화 되어 Connector class가 디자인된다.
- 각 태스크 내부 event sequencing logic 을 정의한다.

# Performance Analysis

- 성능 분석은 Real-Time System에서 매우 중요한 부분이다.
- 양적인 분석은 잠재적인 문제점을 조기에 발견할 수 있게 해준다.
- COMET에서 성능 분석하는 방법
  - Real-Time scheduling thoery
  - Event sequence analysis

# Conclusions

- Concurrent 와 real-time system 디자인의 개념과 메소드에 대해 알 수 있다.
- 디자인을 할 때는 객체지향 개념이 필수적인 요소임을 알수 있다.
- UML을 이용하여 COMET 방법론을 이용해 디자인하는 법을 소개했다.
- 앞으로는 Real-Time 시스템이 급격하게 발달할것이고 미래에는 그 중요성이 증가할것이다.