임베디드 소프트웨어 개발 프로세스 소개

김 영 철

홍익대학교 컴퓨터정보통신공학과

bob@hongik.ac.kr

http://selab.hongik.ac.kr

2009. 5. 13/15

Table of the Content

3일차 수요일

Lecture 1
개발 프로세스

개발 프로세스의 정의

프로세스는 왜 필요한가?

- ❖ 프로젝트 실패에 대한 두려움
- ❖ 체계적이지 못한 개발에 대한 불만
- ❖ 좀더 효율적인 개발에 대한 필요성
- ❖ 미래에 대한 예측

Development Process

❖ 프로세스는 새로운 요구사항 또는 요구사항의 변경에 대해
- ***누가, 언제, 무엇을, 어떻게*** 일해야 하는지를 정의함으로써,
- 새로운 소프트웨어를 구축하고 기존 소프트웨어를 더욱 강화

❖ 프로세스를 구성하는 주요 요소
- 사람(People), 활동(Activity), 산출물(Artifact)

❖ 프로세스의 특징
- 모든 곳에 적용되어 사용될 수 있는 프로세스란 없음
- 프로세스는 경험에 의존

Lecture 1
개발 프로세스

기존의 소프트웨어 개발 프로세스

Product Line vs. MDA

Product Line	MDA
1) 한번의 개발 라이프사이클 후에, 핵심 자산 재사용하는 기법 2) 고품질의 임베디드 시스템을 적시에 경제적으로 개발할 수 있는 *단일 제품군 기반 개발 방법* (Product-Line based Development) 3) Feature Driven 4) 커스터마이징 용이 5) UML 2.0 적용	1) 한번의 개발 라이프사이클 중에, 하나의 메타 모델을 재사용하는 기법 2) 고품질의 임베디드 시스템을 적시에 경제적으로 개발할 수 있는 *이종 제품군 기반 개발 방법* (Heterogeneous Product based Development) 3) xUML + UML 2.0 적용 4) *자동화 도구 필수*

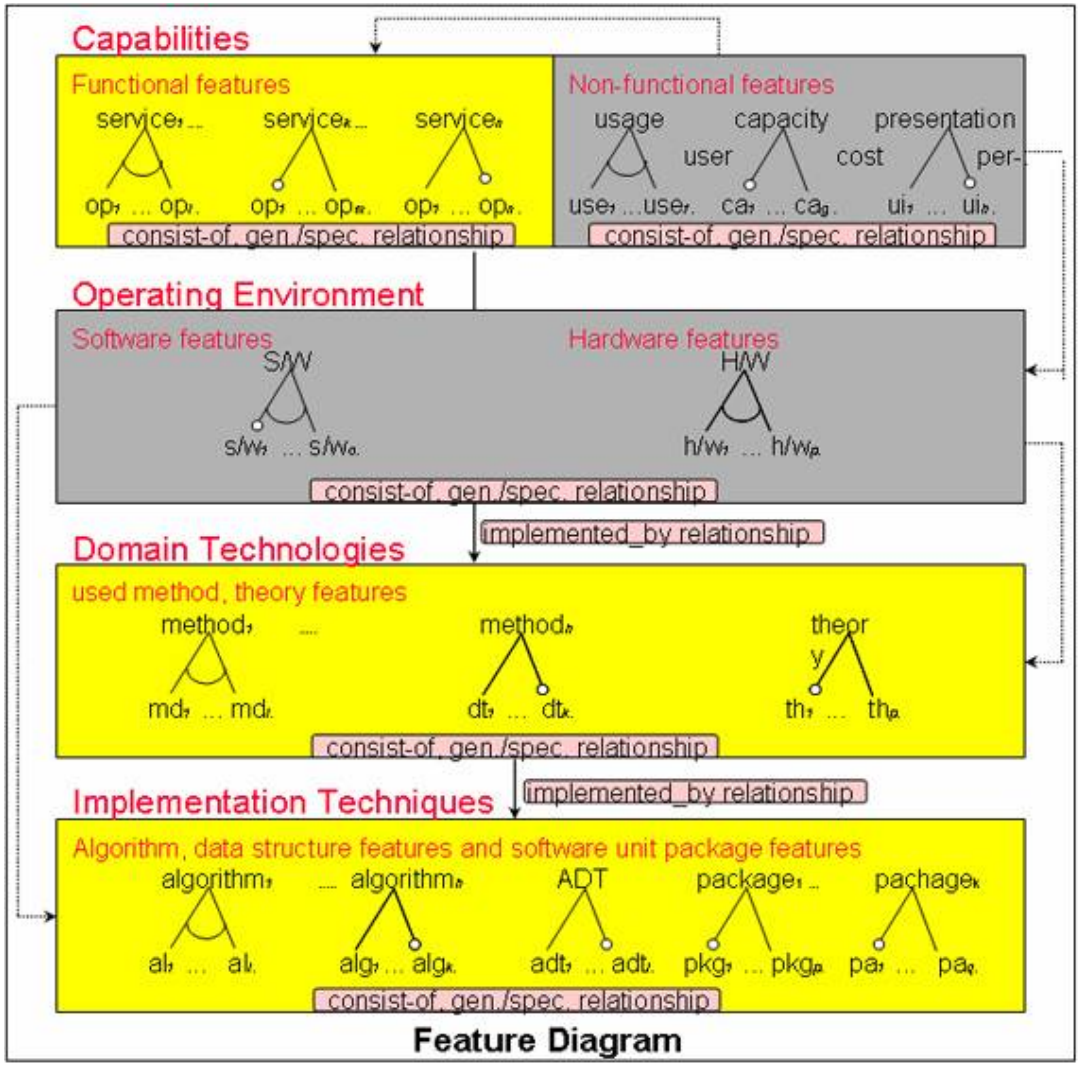
1. Product Line (Feature Model)

Capability Layer

Operating Environment Layer

Domain Technology Layer

Implementation Technique Layer



Composition rules

featuren **require** featurem.
featurei **mutex** featurej.
....

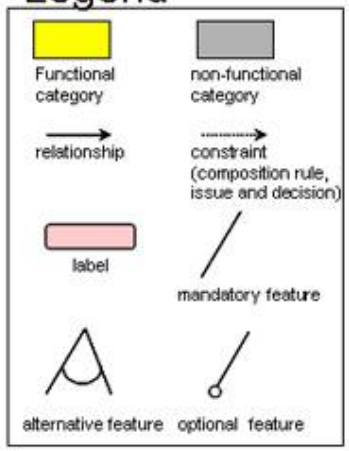
Issues and Decisions

featurek is more secure.
featurep is more expensive.
....

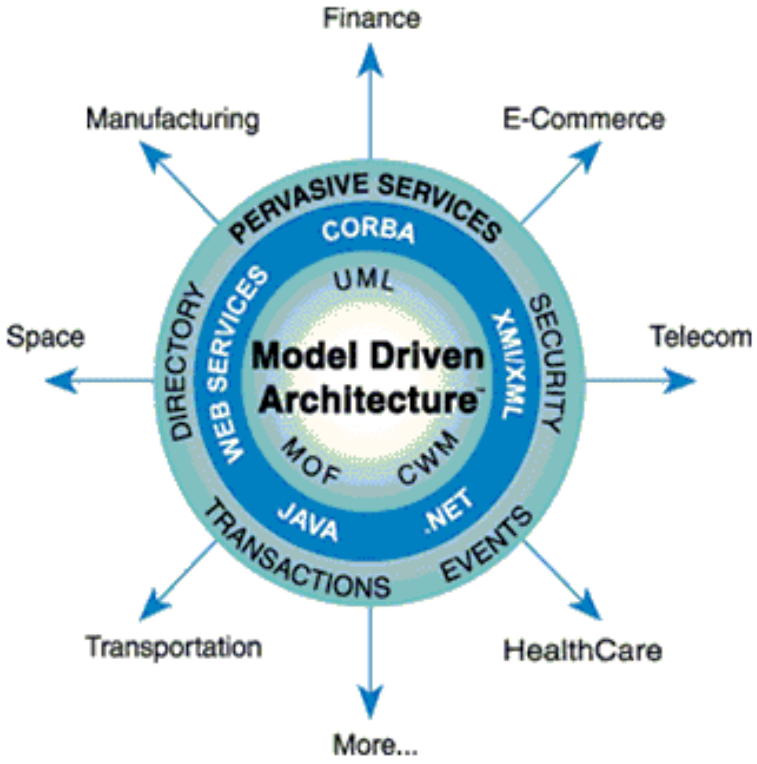
Domain Dictionary

feature1 :means ....
....
Featuren :means.....

Legend



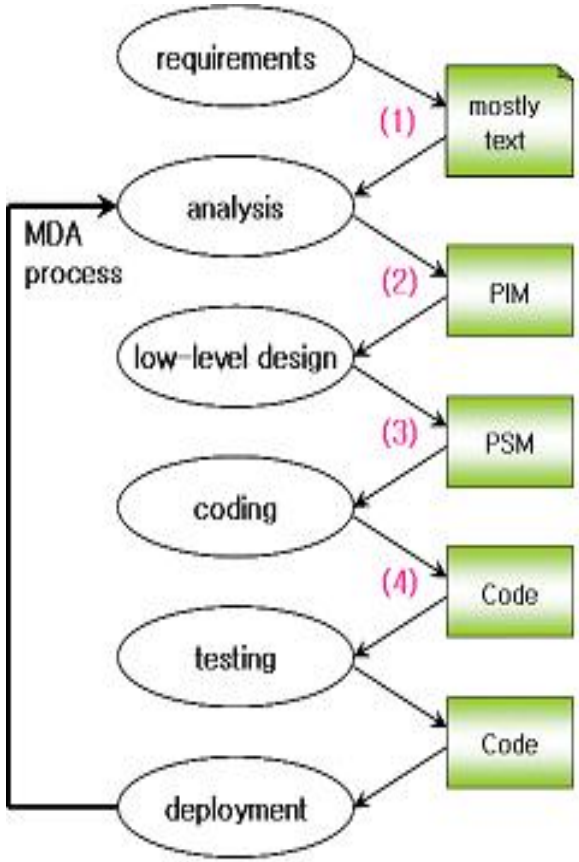
2. What is MDA



OMG is in the ideal position to provide the model-based standards that are necessary to extend integration beyond the middleware approach…Now is the time to put this plan into effect. **Now is the time for the Model Driven Architecture.**

Richard Soley and the OMG staff,
MDA Whitepaper Draft 3.2
November 27, 2000

MDA 개발 프로세스



[MDA 소프트웨어 개발 프로세스]

PIM(Platform Independent Model)

MDA 개발 프로젝트의 시작점은 PIM을 구축하면서 부터이다. PIM은 구현 플랫폼 환경을 고려하지 않은 순수한 플랫폼 독립적인 설계 모델이며 UML 표준 모델링 도구를 이용해 시스템을 모델링한다.

PSM(Platform Specific Model)

특정 플랫폼에 종속적인 모델로서 모든 플랫폼에 대한 정보를 표현

Code

모든 비즈니스 정보를 갖는 PIM에서 각 플랫폼을 적용한 PSM을 통해 Rational Rose와 같은 CASE Tool을 이용해서 특정 환경 종속적인 소스코드를 만든다.

Lecture 1
개발 프로세스

임베디드 소프트웨어 개발 프로세스

임베디드 SW 개발 프로세스 비교

EMMA (Product Line) [Yang, Kang]	Harmony Process (MDD) [Douglass]	HiMEM (MDA/MDD Paradigm) [Kim]
1) 한번의 개발 라이프사이클 후에, 핵심 자산 재사용하는 기법 2) 단종의 임베디드 시스템을 적시에 경제적으로 개발 3) Feature Driven 4) 커스터마이징 용이 5) UML 2.0 적용	1) 한번의 개발 라이프사이클 중에, 정제한 프로토타입 재사용하는 기법 2) 단종의 임베디드 시스템을 적시에 경제적으로 개발 3) 시스템/소프트웨어 책임 구분 4) UML 2.0 + SysML 적용 5) *코드 자동 생성기 필수*	1) 한번의 개발 라이프사이클 중에, 하나의 메타 모델을 재사용하는 기법 2) 이종의 임베디드 시스템을 적시에 경제적으로 개발 3) xUML + UML 2.0 적용 4) 커스터마이징 용이 5) *코드 자동 생성기 필수*

1. EMMA (Product Line) [Yang, Kang]

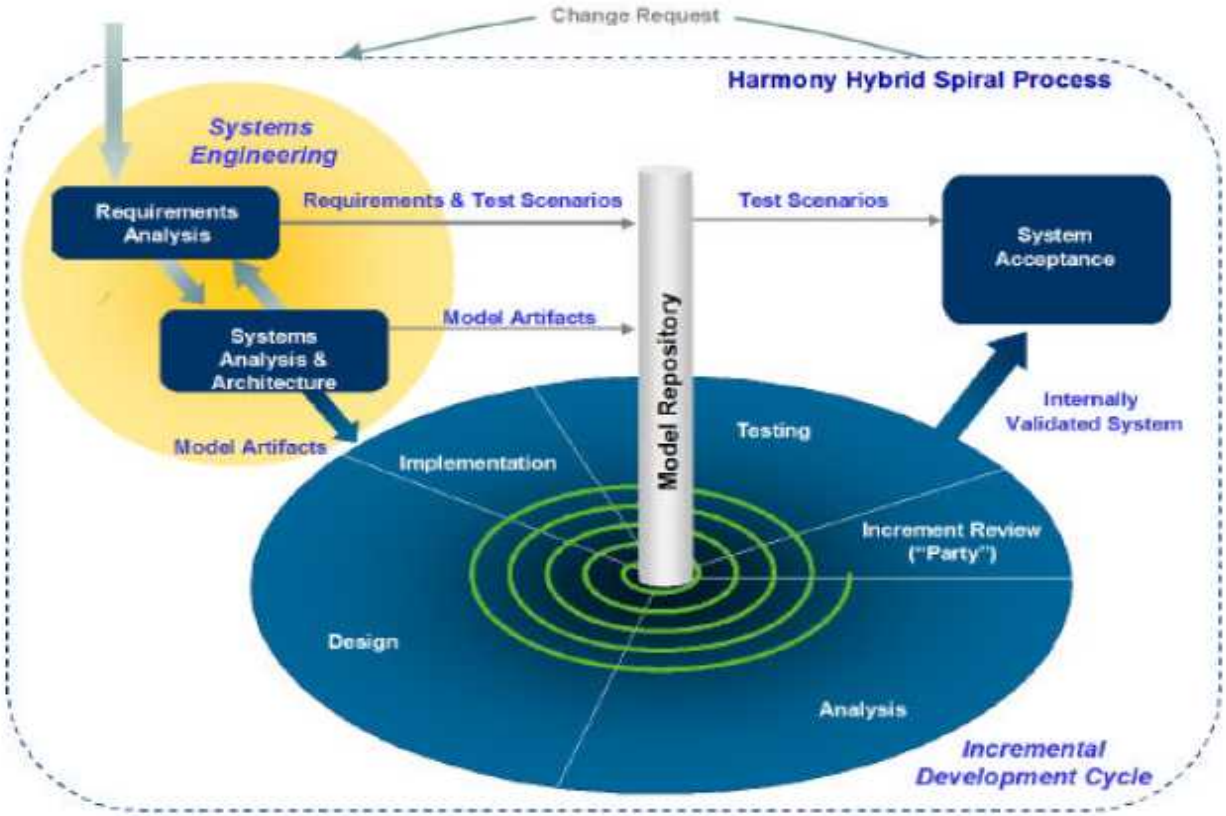
- **Overview**
  - Product Line을 임베디드 개발에 적용
  - 동일한 제품군의 재사용 자산을 정의하고 재사용 체계를 구축하는 방법
  - 제품군에 대한 핵심 자산을 구축
  - 프로젝트 수행 중에 발생한 문서, 자산을 저장하고 이것을 재사용
  - 핵심자산을 잘 정의하면 효율적으로 재사용 가능

1. EMMA (Product Line) [Yang, Kang](cont.)

- **Problem**
  - 단일 제품군에만 재사용 가능
  - 한번의 시스템 개발 후에 핵심자산을 구축하기 어려움 (여러 번의 개발이 필요)
  - 새로운 시스템 개발에 매우 불리함

2. Harmony Process(MDD) [B. P. Douglass]

- **Overview**
  - 모델기반방법을 임베디드 개발에 적용
  - 점진적이고 반복적인 방법으로 개발



2. Harmony Process(MDD) [B. P. Douglass]

- **Problem**
  - 재사용성에 대한 고려가 미흡
  - 제품의 완료에 대한 기준이 없음
    (언제까지 반복해야 되는지 기준이 없음)
  - 여러 번의 프로토타입 생성으로 개발시간이 오래 걸림
  - 이종 시스템을 위한 개발 방법이 아님

3. HiMEM(MDA/MDD Paradigm) [Kim]

- **Overview**
  - 한번의 개발 라이프사이클 중에, 하나의 메타 모델을 재사용하는 기법
  - 하나의 메타모델을 모델변환으로 여러 개의 모델로 변환
  - 모델 변환을 통해 중복 설계의 문제를 해결가능
  - MDA/MDD 파라다임을 이용하여 임베디드에 적용

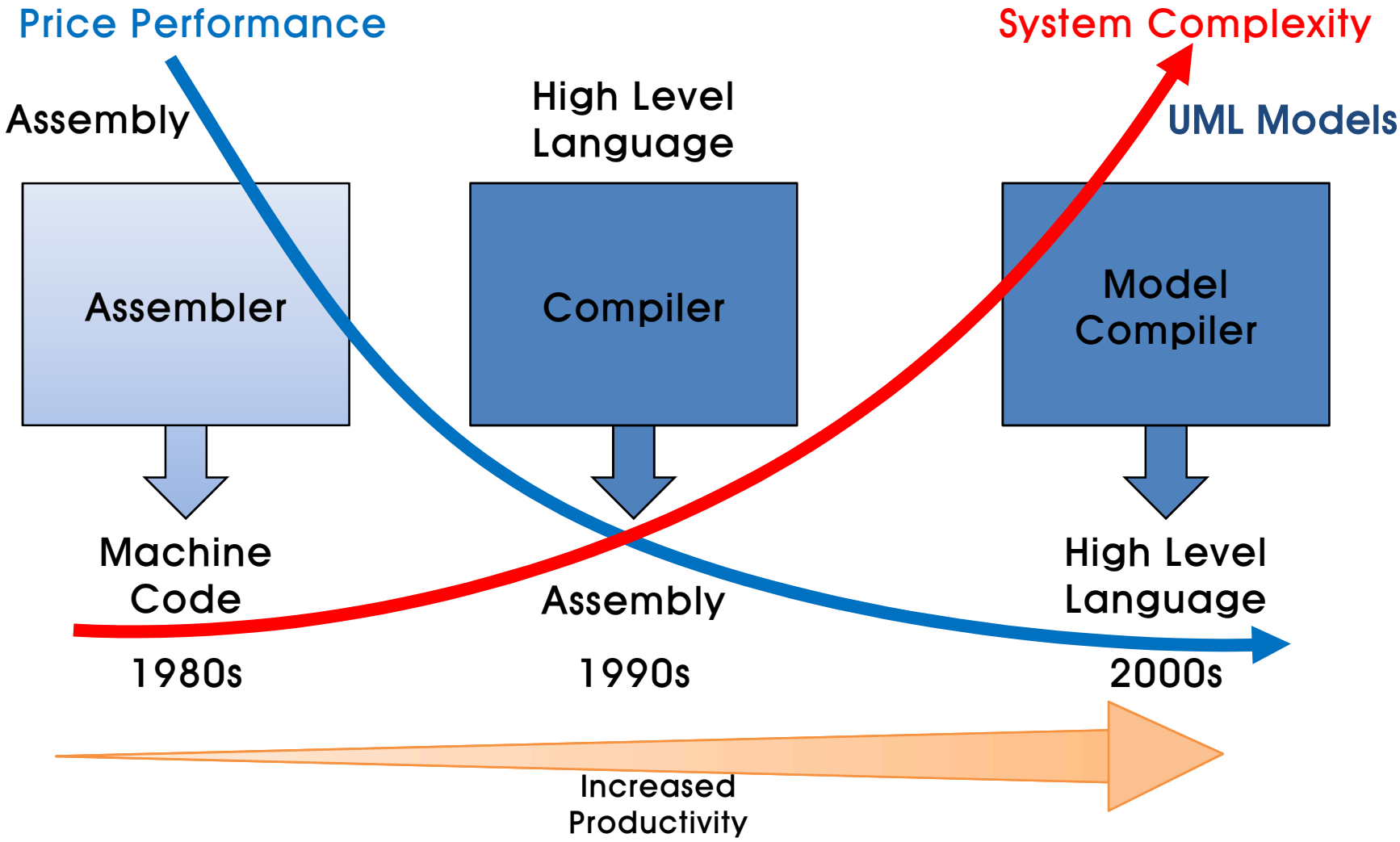
3. HiMEM(MDA/MDD Paradigm) [Kim](cont.)

- **Problem**
  - Target Independent Model 설계가 어려움
  - UML, UML Profile, MOF, XMI, CWM, OCL 등 많은 기술이 포함되어 쉽게 적용하기 어려움
  - 자동화 도구 필수

Lecture 1
개발 프로세스

기존 UML 비교

The Evolution of Software Development



기존 UML 비교

Capability	UML 1.x	UML 2.0	Embedded UML	xUML
Primary Market	Commercial/business applications	Telecom, Real-time	SoC	Embedded
분석 능력	X	O	O	O
이벤트를 모델링 할 수 있는가	X	O	–	O
구현 전에 설계 변화를 분석할 수 있는가	X	O	–	O
설계 능력	O	O	–	O
Reverse Fork/Join 기능을 가지는가	X	X	X	X
동적 모델링에서 Concurrency 모델링이 가능한가	X	O	X	O
Nondeterministic 관점을 고려하는가	X	X	X	X
설계 변경이 쉬운가	X	X	–	O
여러 타겟에서의 설계 재사용이 가능한가	X	O	–	O
설계 단계에서 문제를 바로잡을 수 있는가	X	X	–	O
구현 능력	X	X	X	O
코드 생성	스켈레톤 코드	VM 기반	VM 기반	동적 요소를 포함한 완벽한 코드
모델과 코드가 연관되어져 있는가	X	O	X	O
실시간/임베디드 시스템의 구현능력을 가지고 있는가	X	O	–	O
테스트/디버그 능력	X	X	X	O
실행 가능한 모델인가	X	X	X	O
디자인 레벨에서 디버깅이 가능한가	X	X	X	O

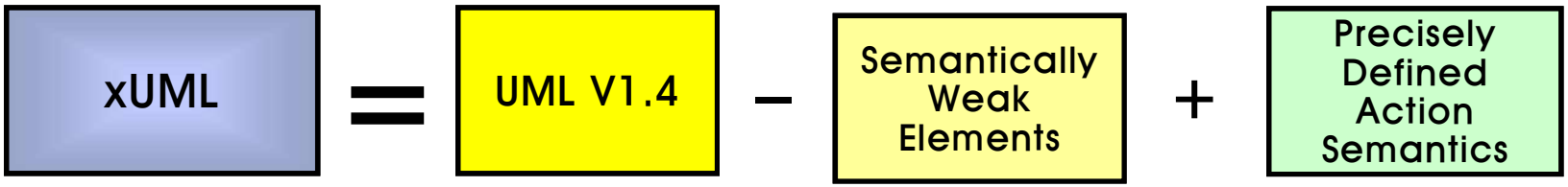
UML vs xUML

UML x.x	xUML
Gigantic; hard to understand	Tractable subset; easy to learn
Defects found late	Defects found early
Model, then code	Model *is software*
Models capture implementation	Models capture rules and policies of problem
Models specific to target	Retargettable/Reusable models

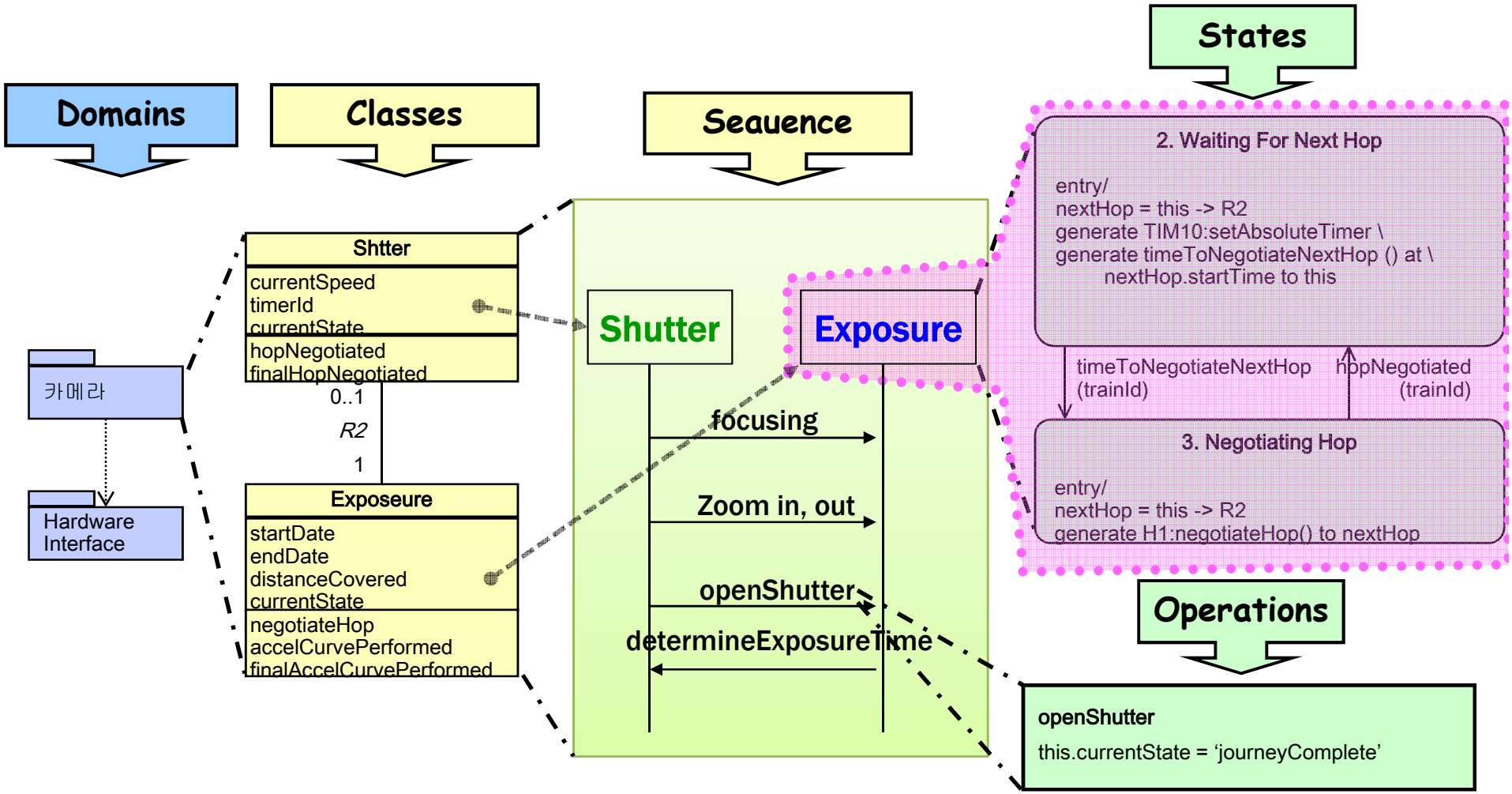
xUML(Executable UML) 소개

❖ **xUML** [Mellor&Balcer, 2002][Raistrick, etc.,2004] **is**

- an **executable version** of the UML, with…
  - clearly defined simple model structure
  - a precise semantics for actions, which will be incorporated into the UML standard
  - a compliant action specification language
  - an accompanying process
- a **proven development process**, oriented towards…
  - executable modelling
  - large-scale reuse
  - pattern based design



Model Layers



Lecture 1
개발 프로세스

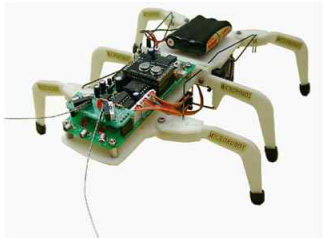
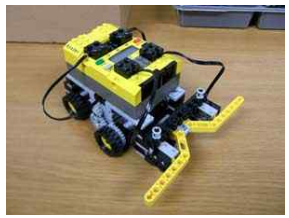
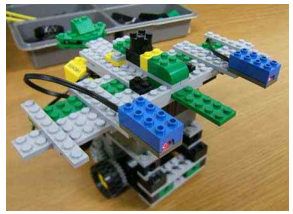
Embedded MDA Approach

기존 접근 방법

❖ 임베디드 소프트웨어 개발 시,

- 서로 다른 도메인 타겟상에서 각각의 코드를 개발
- 각 시스템마다 개발 소요기간 및 투입 인원이 중복 소요됨으로 비용이 증대

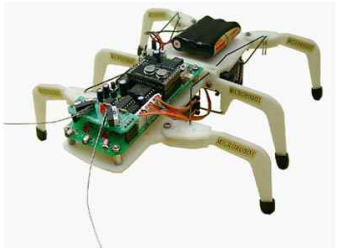


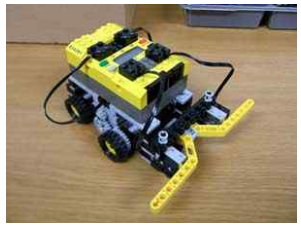





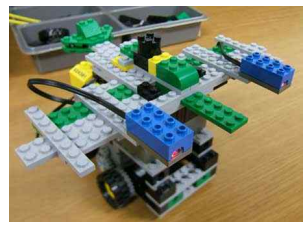
채택한 MDA 방법

❖ 하나의 메타 모델링(Target Independent Model)을 통하여 각각의 다른 도메인에 맞는 타겟 종속적 모델(Target Specific Model)들을 만들고, 그에 따른 소스 코드(Target Dependent Code)를 개발하는 방법

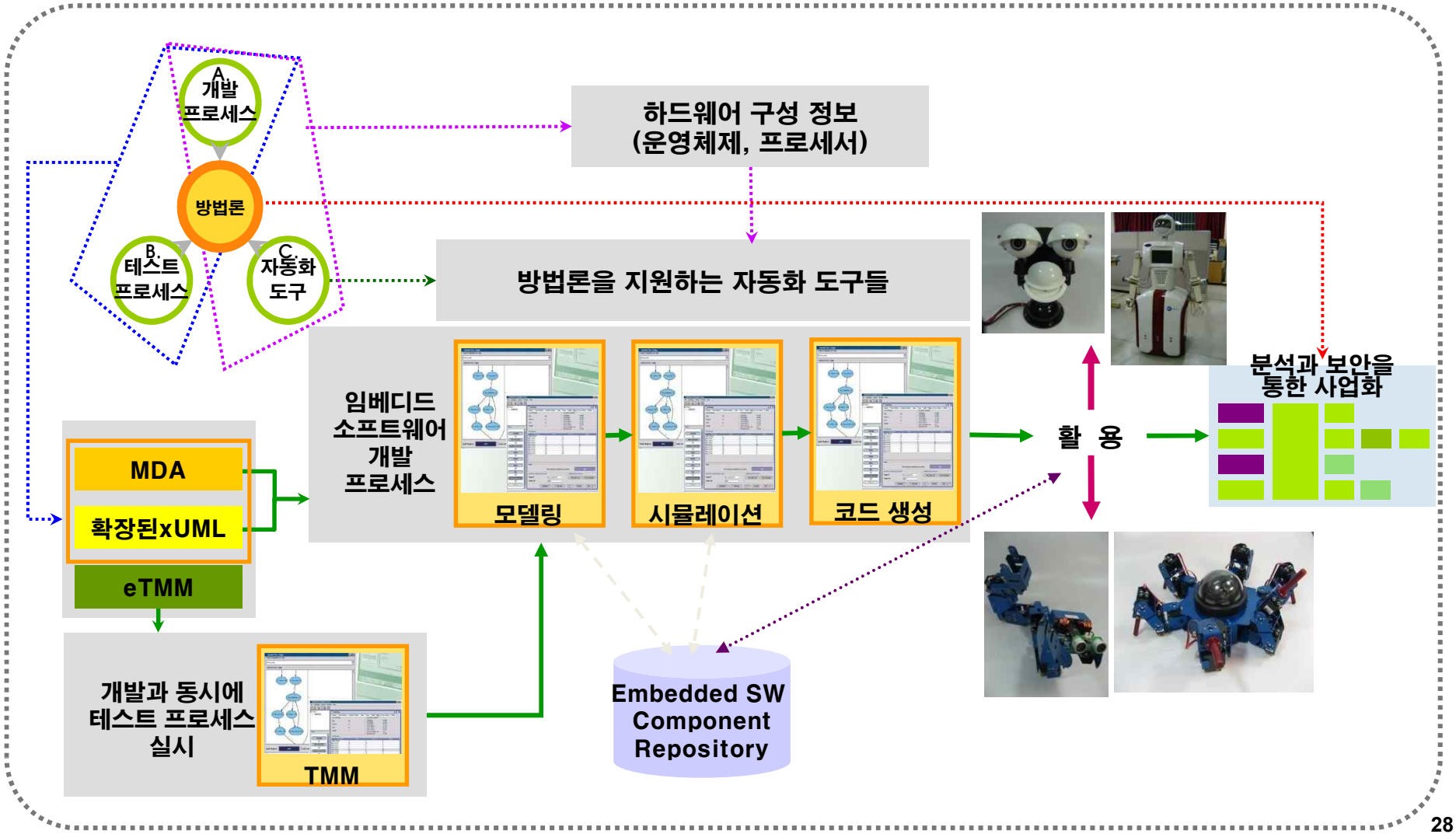
MDA 기반의 임베디드 시스템 개발, xUML 사용한 모델링



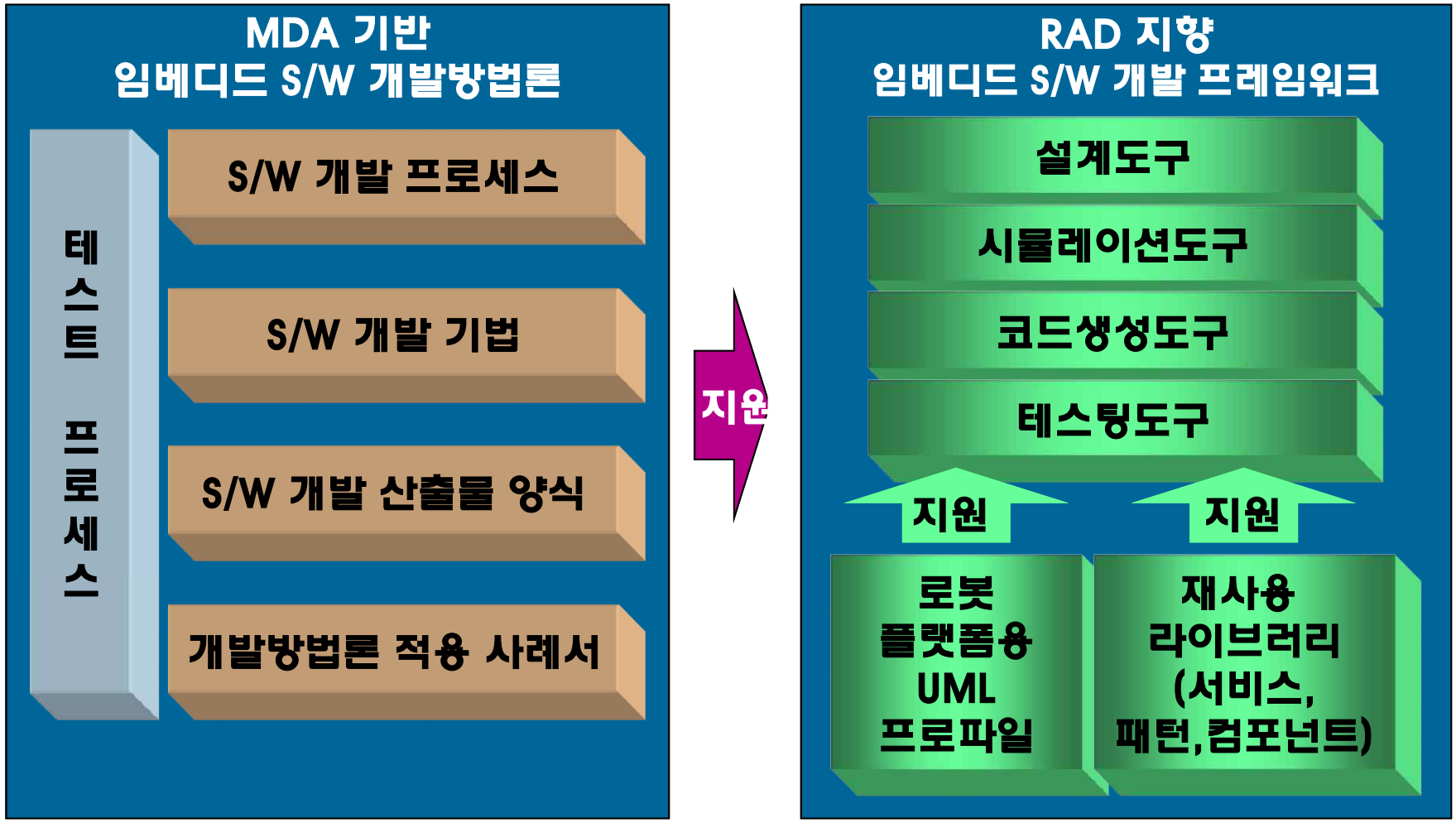




임베디드 S/W 개발 방법론 구조(HiMEM V1.0)



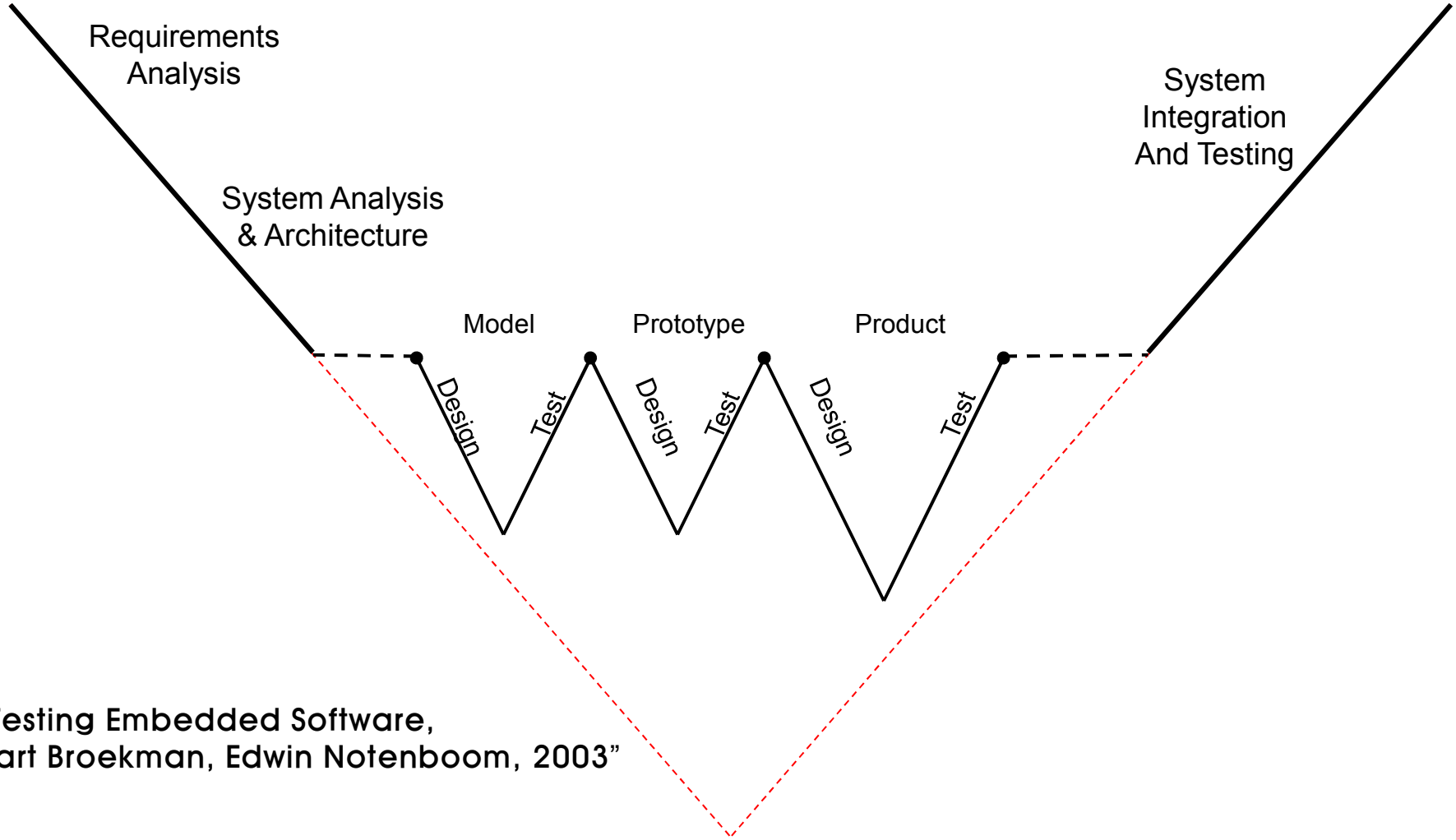
임베디드 S/W 개발 방법



MDA(Model Driven Architecture)

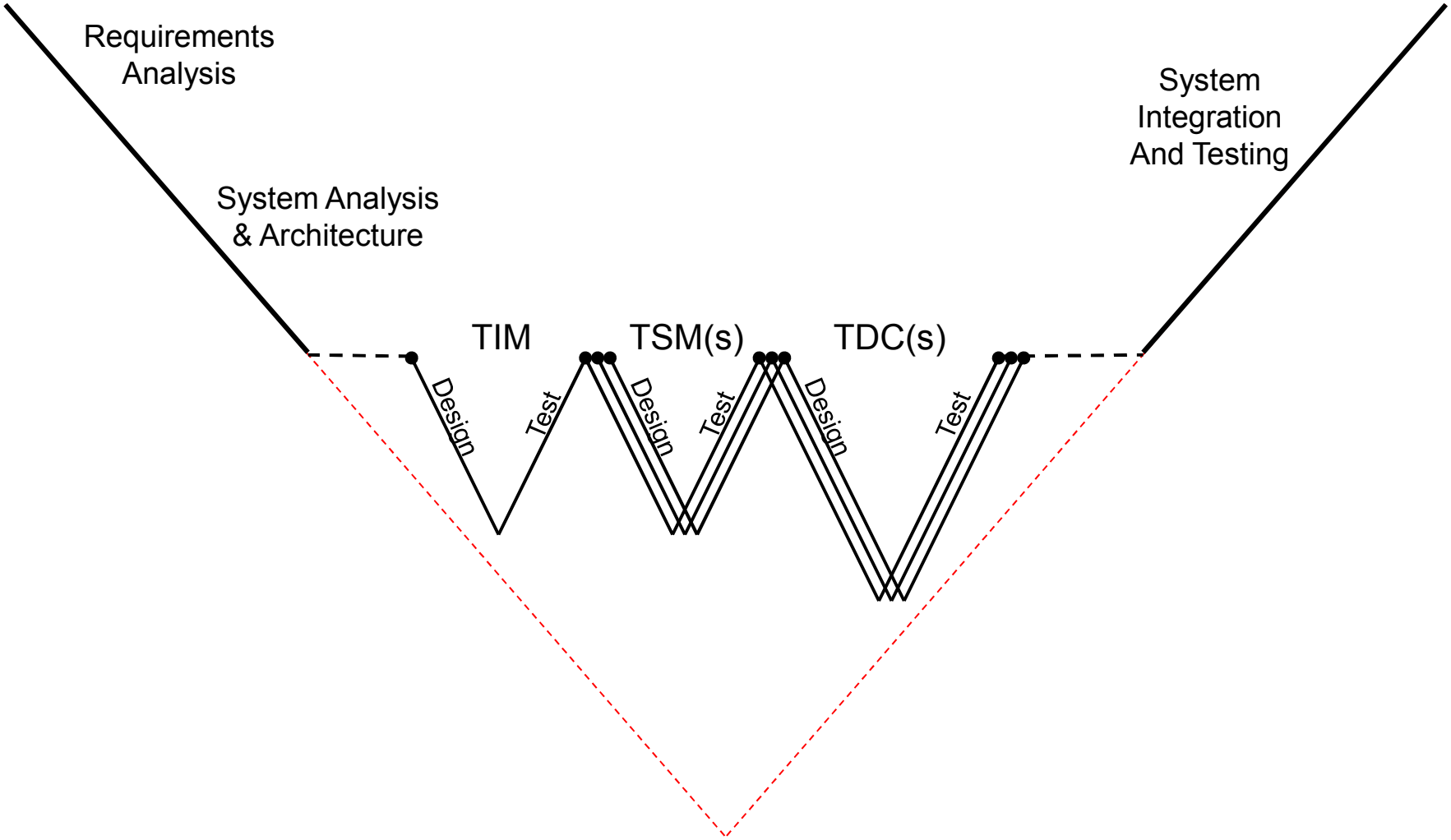
RAD(Rapid Application Development)

기존 임베디드 S/W 개발 프로세스 구조



"Testing Embedded Software,
Bart Broekman, Edwin Notenboom, 2003"

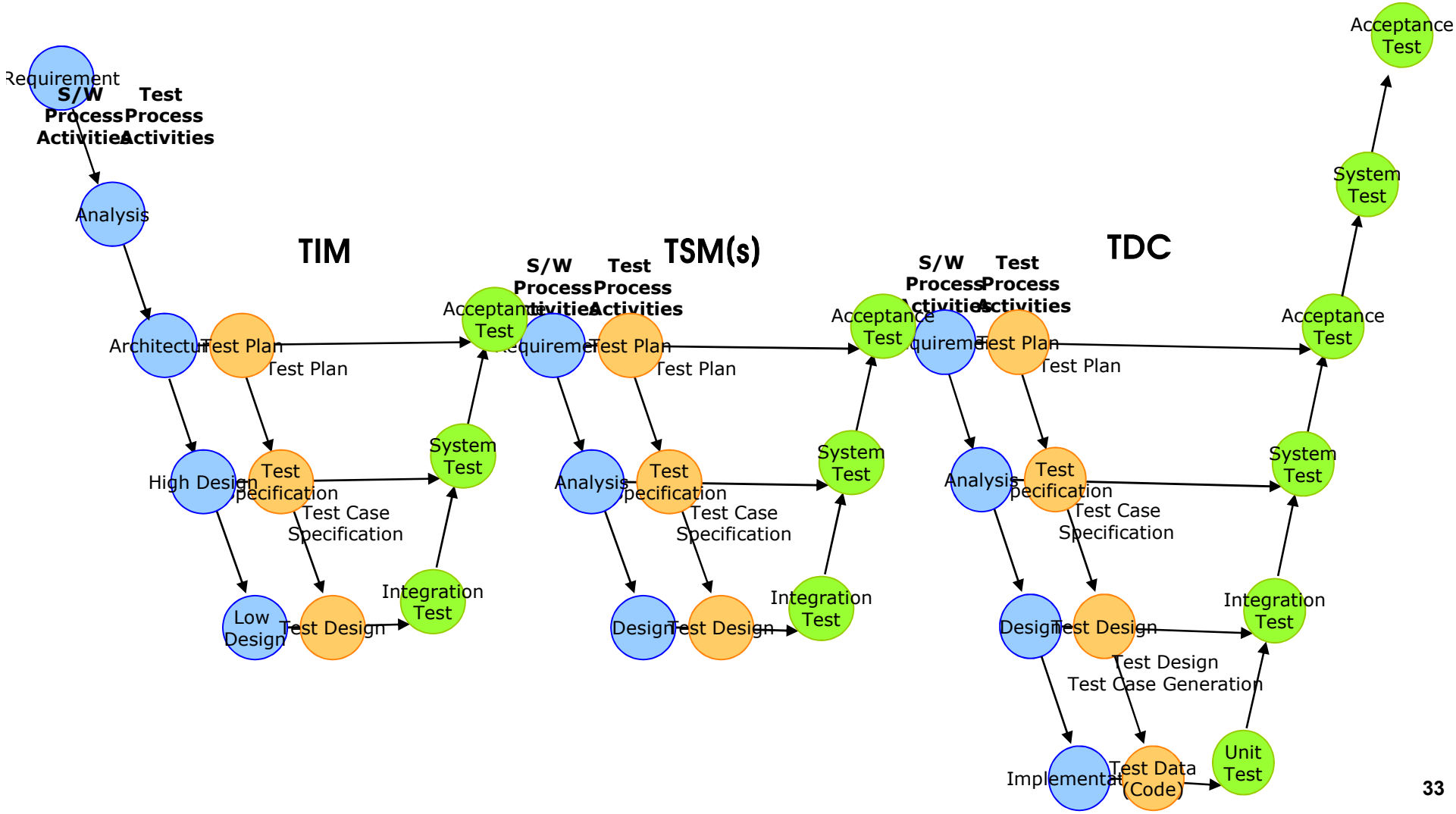
제안한 임베디드 S/W 개발 프로세스 구조



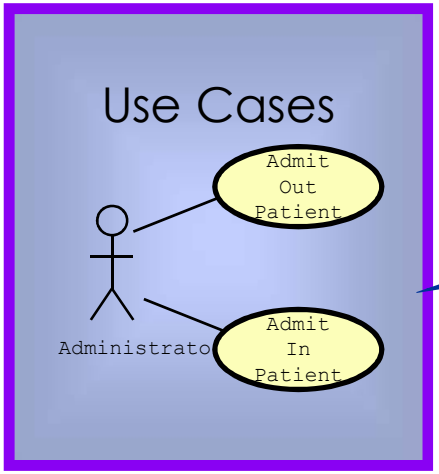
Lecture 1
개발 프로세스

Multiple V-모델

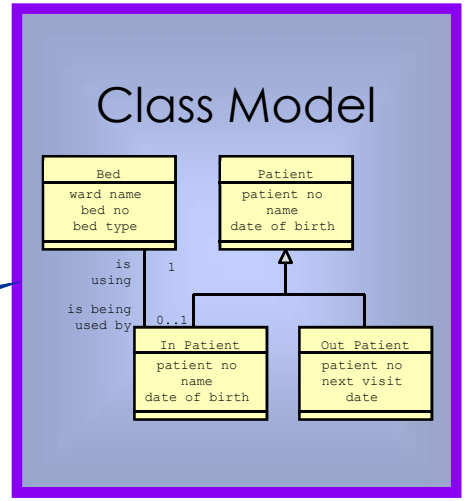
MDA와 Multiple V 모델의 접목



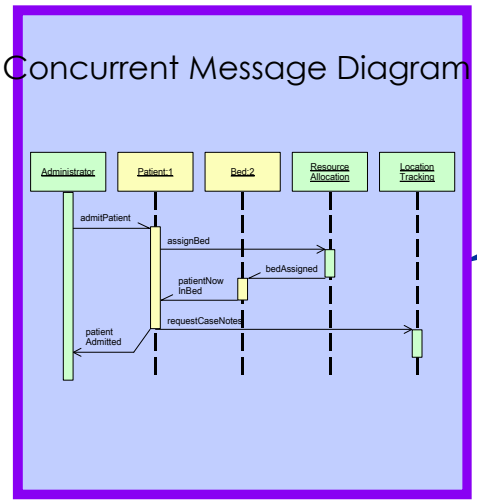
모델링 절차



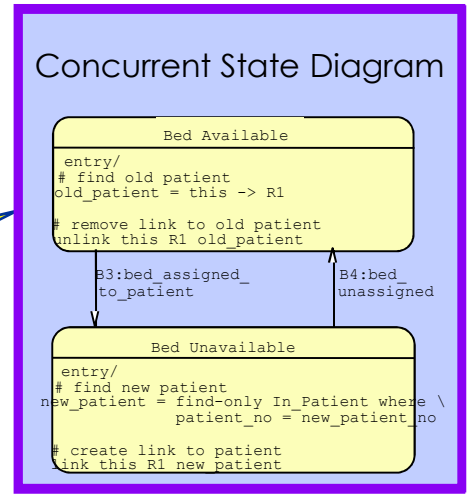
Capture requirements with use cases



Produce first-cut class diagram for each domain to be modelled

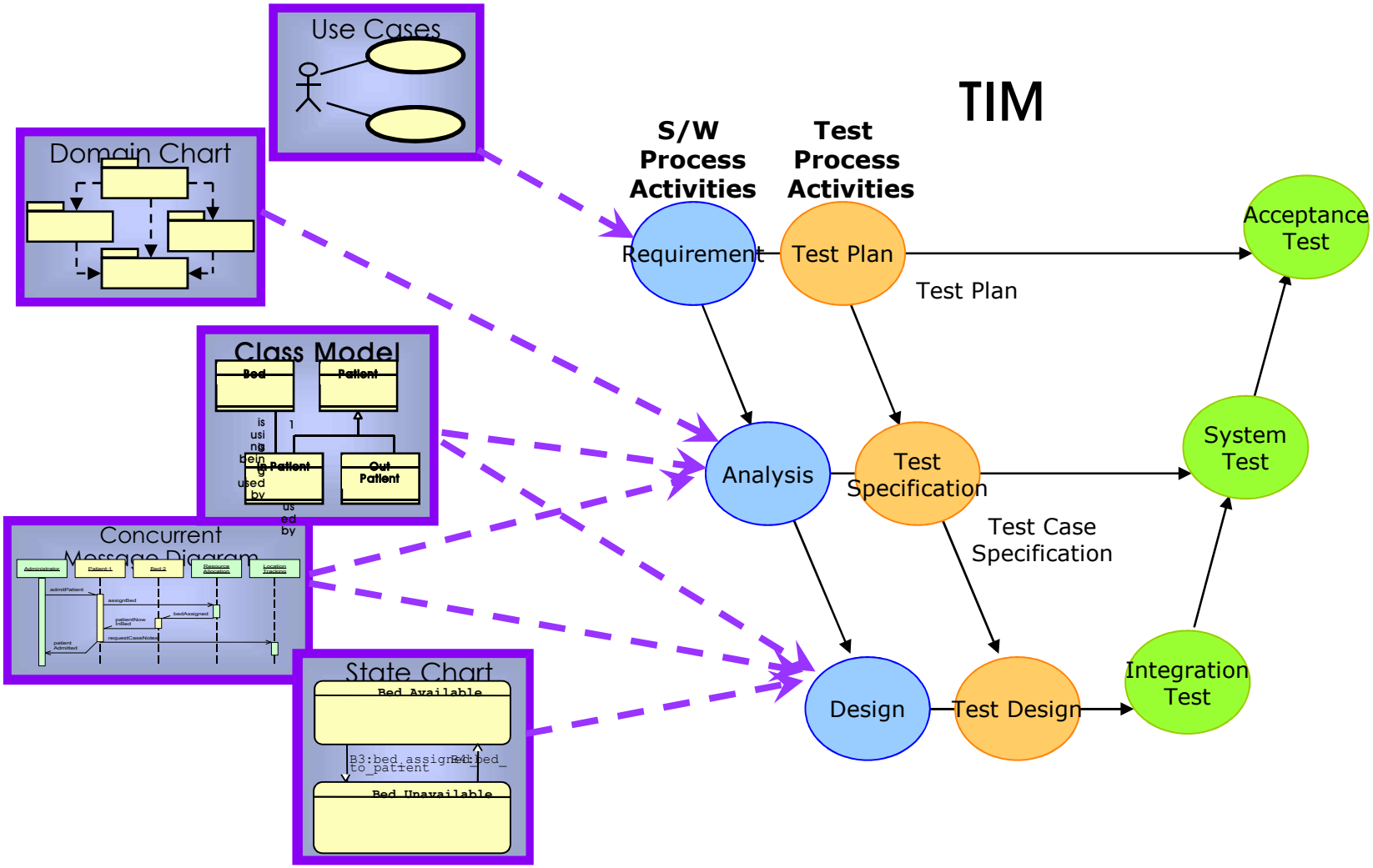


Produce the interaction diagram

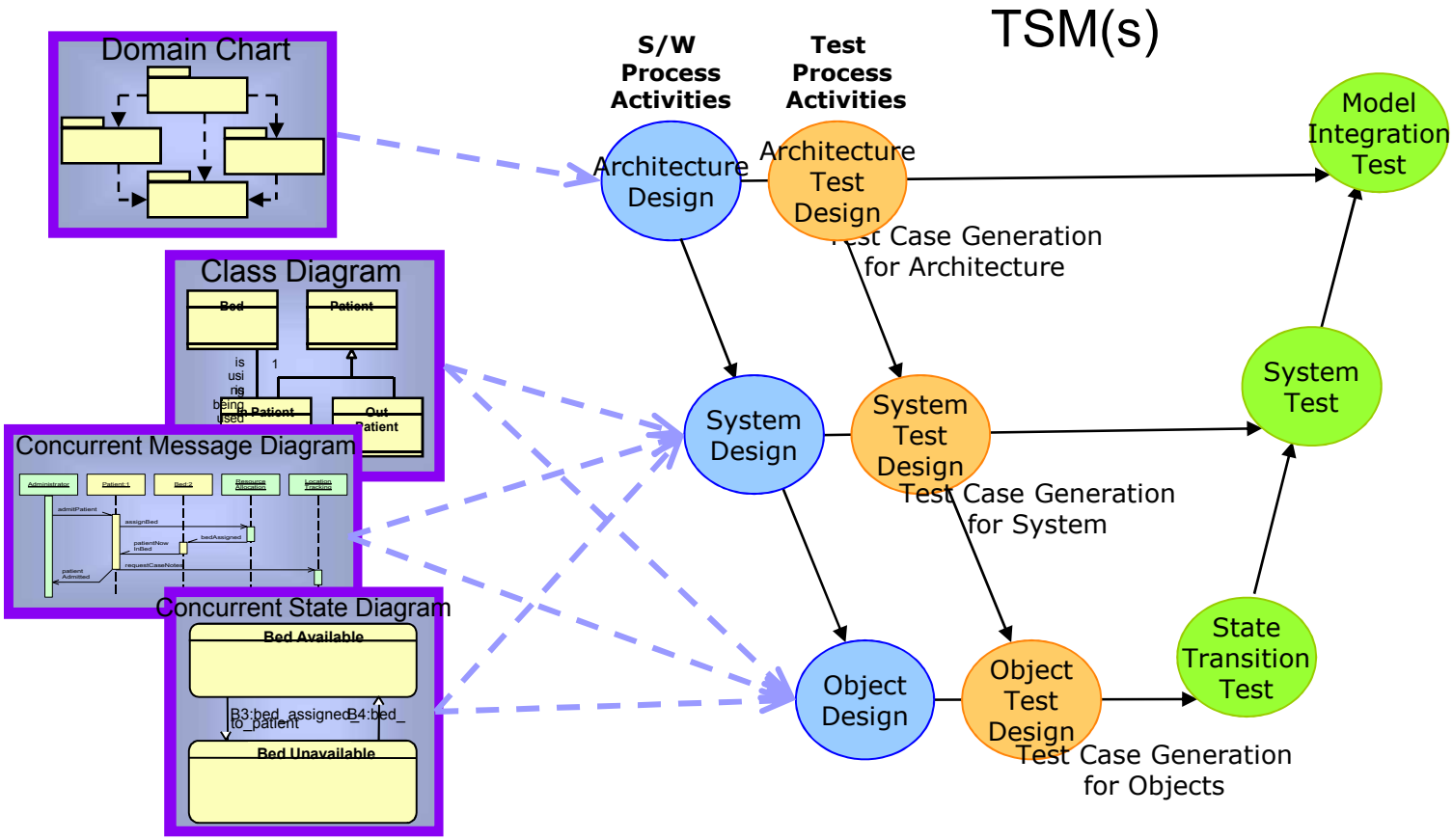


Develop state diagram for classes and specify actions using an OCL

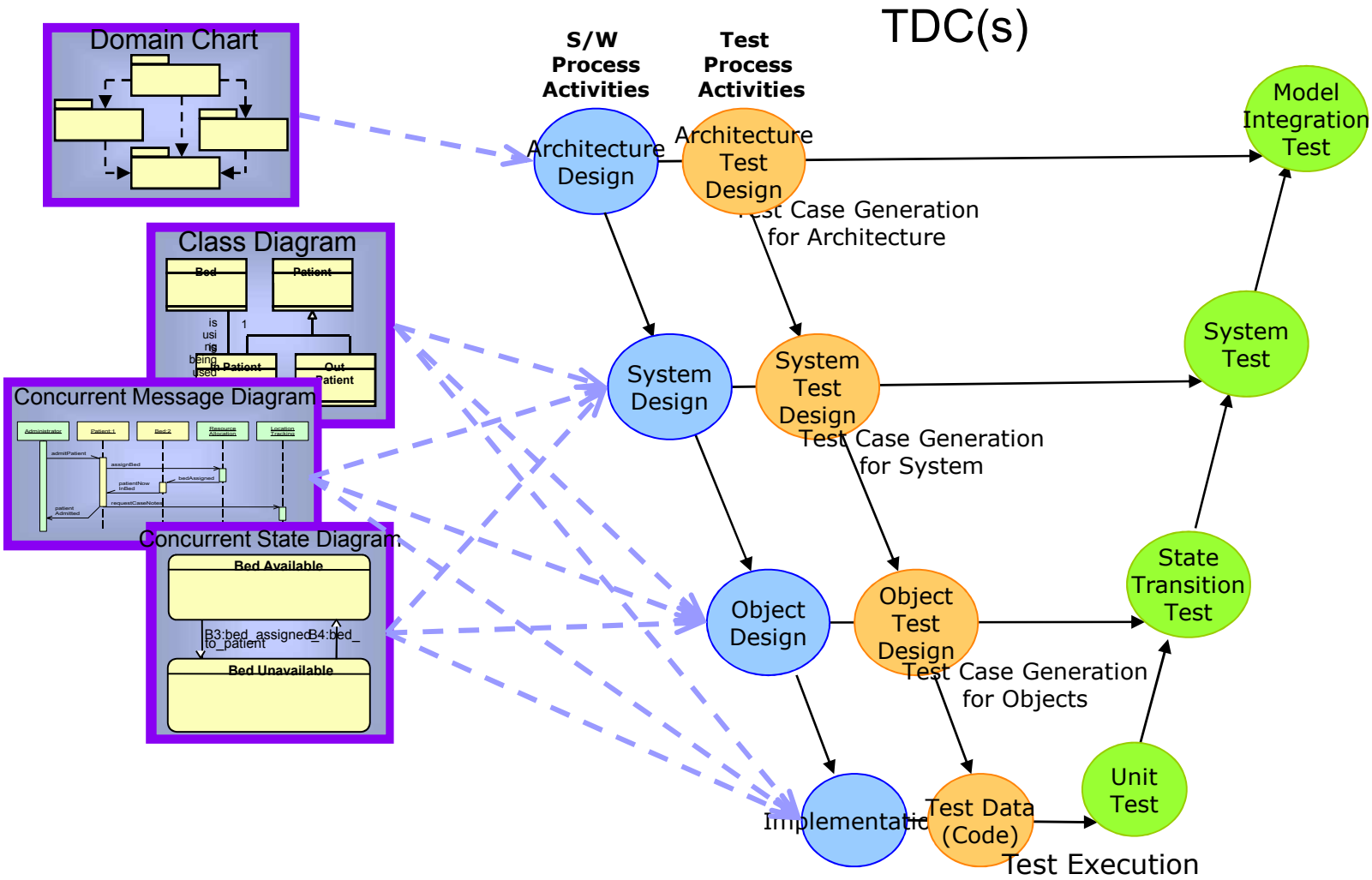
1.타겟 독립 모델(Target Independent Model)



2.타켓 종속 모델(Target Specific Model)



3.타켓 의존 코드(Target Dependent Code)



MDA기반의 임베디드 S/W 컴포넌트 방법론[1/3]

공정	단계	활동	작업	산출물
Target Independent Model	도메인 정의	요구사항 정의	상위 요구사항 정의	상위 요구사항 정의서
			도메인 모델링	도메인 명세서
				용어집
			현행 시스템 분석	현행 시스템 분석서
			요구사항 명세	요구사항 명세서
	도메인 분석	요구사항 분석	유스케이스 모델링	유스케이스 명세서
			사용자 인터페이스 모델링	사용자 인터페이스 정의서
			추상 클래스 모델링	추상 클래스 명세서
			테스트 케이스 정의	테스트케이스 정의서
	도메인 구조	아키텍처 정의	소프트웨어 아키텍처 정의	소프트웨어 아키텍처 정의서
			시스템 아키텍처 정의	시스템 아키텍처 정의서
			표준 지침 수립	표준 지침서
		프로세스 모델링	프로세스 모델링	프로세스 명세서
	도메인 설계	컴포넌트 추출	컴포넌트 추출	컴포넌트 추출 결과서
		컴포넌트 개략 설계	컴포넌트 식별	컴포넌트 목록
				컴포넌트 아키텍처 정의서
			컴포넌트 획득방법 식별	컴포넌트 획득 방법 식별서
			인터페이스 상호작용 정의	인터페이스 상호작용 명세서
			컴포넌트 명세	인터페이스 명세서
				컴포넌트 명세서
			사용자 인터페이스 설계	사용자 인터페이스 설계서
			개념적 데이터 모델링	개념적 데이터 설계서
	도메인 테스트	도메인 테스트	도메인 테스트 설계	도메인 테스트 설계서
			도메인 테스트 수행	도메인 테스트 결과서
총 1 공정	**총 5 단계**	**총 7 작업**	**총 21 활동**	**총 24 산출물**

MDA기반의 임베디드 S/W 컴포넌트 방법론[2/3]

공정	단계	활동	작업	산출물
Target Specific Model	도메인 종속 정의	요구사항 정제	요구사항 정의	요구사항 정의서
			도메인 모델링	도메인 명세서
				용어집
			요구사항 명세	요구사항 명세서
	도메인 종속 분석	요구사항 분석	유스케이스 모델링	유스케이스 명세서
			사용자 인터페이스 모델링	사용자 인터페이스 정의서
			클래스 모델링	클래스 명세서
			테스트 케이스 정의	테스트케이스 정의서
	도메인 종속 구조	아키텍처 정제	소프트웨어 아키텍처 정제	소프트웨어 아키텍처 정의서
			시스템 아키텍처 정제	시스템 아키텍처 정의서
			표준 지침 확정	표준 지침서
	도메인 종속 설계	종속 컴포넌트 추출	컴포넌트 추출	컴포넌트 추출 결과서
		컴포넌트 상세 설계	컴포넌트 식별	컴포넌트 목록
				컴포넌트 아키텍처 정의서
			컴포넌트 획득방법 식별	컴포넌트 획득 방법 식별서
			인터페이스 상호작용 정의	인터페이스 상호작용 명세서
			컴포넌트 내부 설계	인터페이스 설계서
				컴포넌트 설계서
			사용자 인터페이스 내부 설계	사용자 인터페이스 설계서
			논리적 데이터 모델링	논리적 데이터 설계서
	단위 테스트	단위 테스트	단위 테스트 설계	단위 테스트 설계서
			단위 테스트 수행	단위 테스트 결과서
총 1 공정	**총 5 단계**	**총 6 작업**	**총 19 활동**	**총 22 산출물**

MDA기반의 임베디드 S/W 컴포넌트 방법론[3/3]

공정	단계	활동	작업	산출물
Target Dependent Code	대상 정의	요구사항 정제	요구사항 정의	요구사항 정의서
			도메인 모델링	도메인 명세서
				용어집
			요구사항 명세	요구사항 명세서
	대상 분석	요구사항 분석	클래스 모델링	클래스 명세서
			테스트 케이스 정의	테스트케이스 정의서
	대상 설계	컴포넌트 구현 설계	컴포넌트 구현 설계	컴포넌트 구현 설계서
			사용자 인터페이스 구현 설계	사용자 인터페이스 구현 설계서
			물리적 데이터 모델링	물리적 데이터 설계서
	대상구현	컴포넌트 구현	데이터베이스 구축	물리적 데이터베이스
			컴포넌트 구현	컴포넌트 코드
			사용자 인터페이스 구현	사용자 인터페이스 코드
	대상 테스트	대상 테스트	대상 테스트 설계	대상 테스트 설계서
			대상 테스트 수행	대상 테스트 결과서
		시스템 테스트	시스템 테스트 설계	시스템 테스트 설계서
			시스템 테스트 수행	시스템 테스트 결과서
총 1 공정	**총 5 단계**	**총 6 작업**	**총 15 활동**	**총 16 산출물**

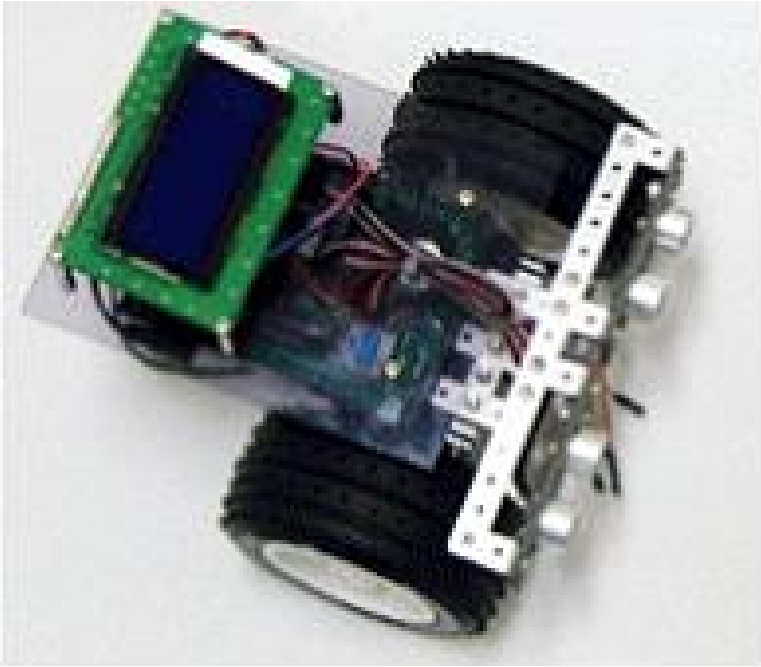
Lecture 1
개발 프로세스

임베디드 프로세스를 이용한 적용
(Small Unmanned Ground Vehicle)

적용사례

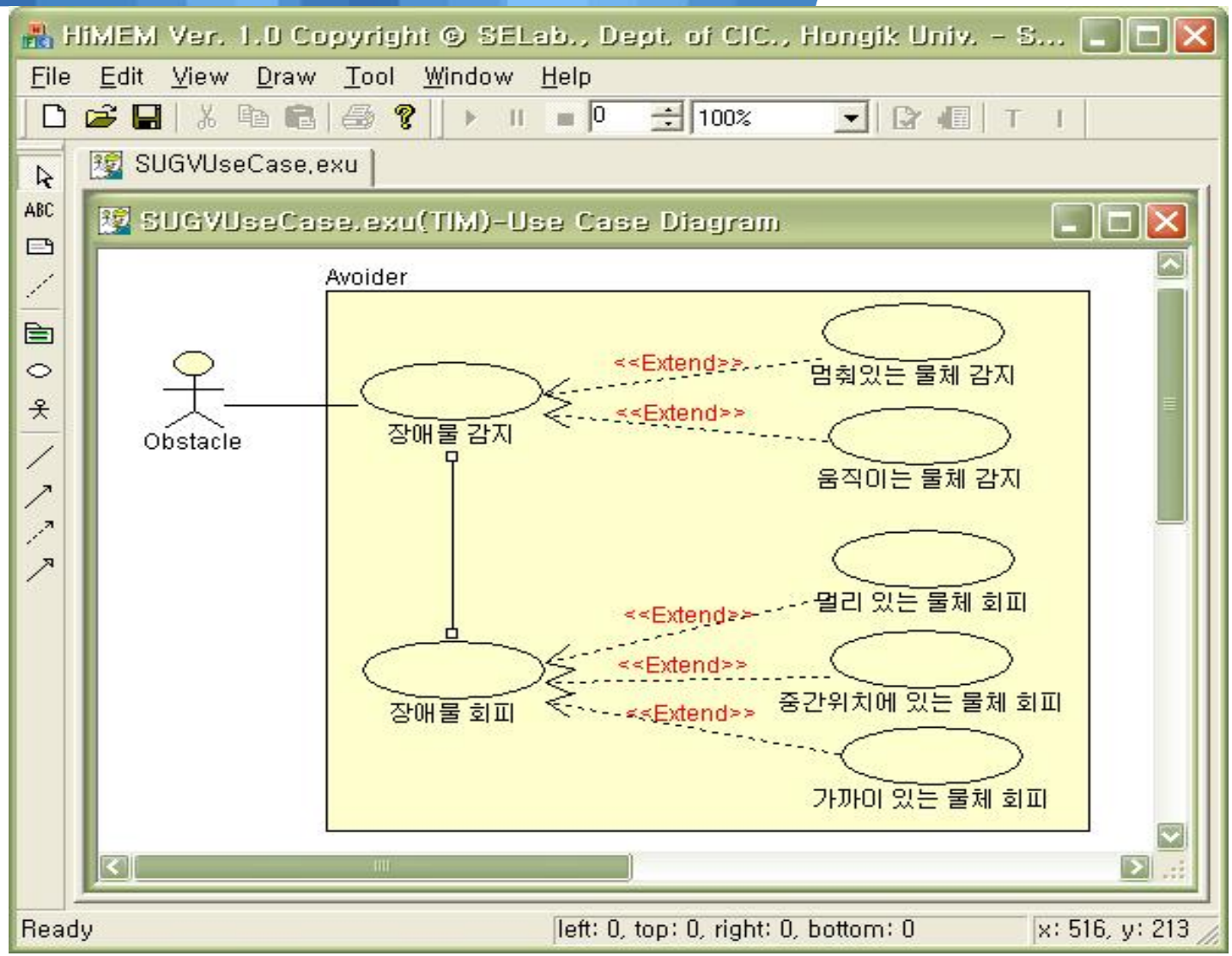
❖ SUGV (Small Unmanned Ground Vehicle) System

- HiMEM을 사용하여 개발
- 자동화 도구를 통해서 자동 코드생성 및 실행



Microcontroller	Ubicom SX48AC 20MHz
RAM	32 KByte
EEPROM	32 KByte
센서	초음파센서 2개
디스플레이	Text LCD
서보모터	2개
JVM	하드웨어
개발 언어	Java

유스 케이스 다이어그램


HiMEM Ver. 1.0 Copyright © SELab., Dept. of CIC., Hongik Univ. - S...
File Edit View Draw Tool Window Help
SUGVUseCase.exu
SUGVUseCase.exu(TIM)-Use Case Diagram
Avoider
Obstacle
장애물 감지
장애물 회피
<<Extend>>
멈춰있는 물체 감지
움직이는 물체 감지
멀리 있는 물체 회피
중간위치에 있는 물체 회피
가까이 있는 물체 회피
Ready
left: 0, top: 0, right: 0, bottom: 0
x: 516, y: 213

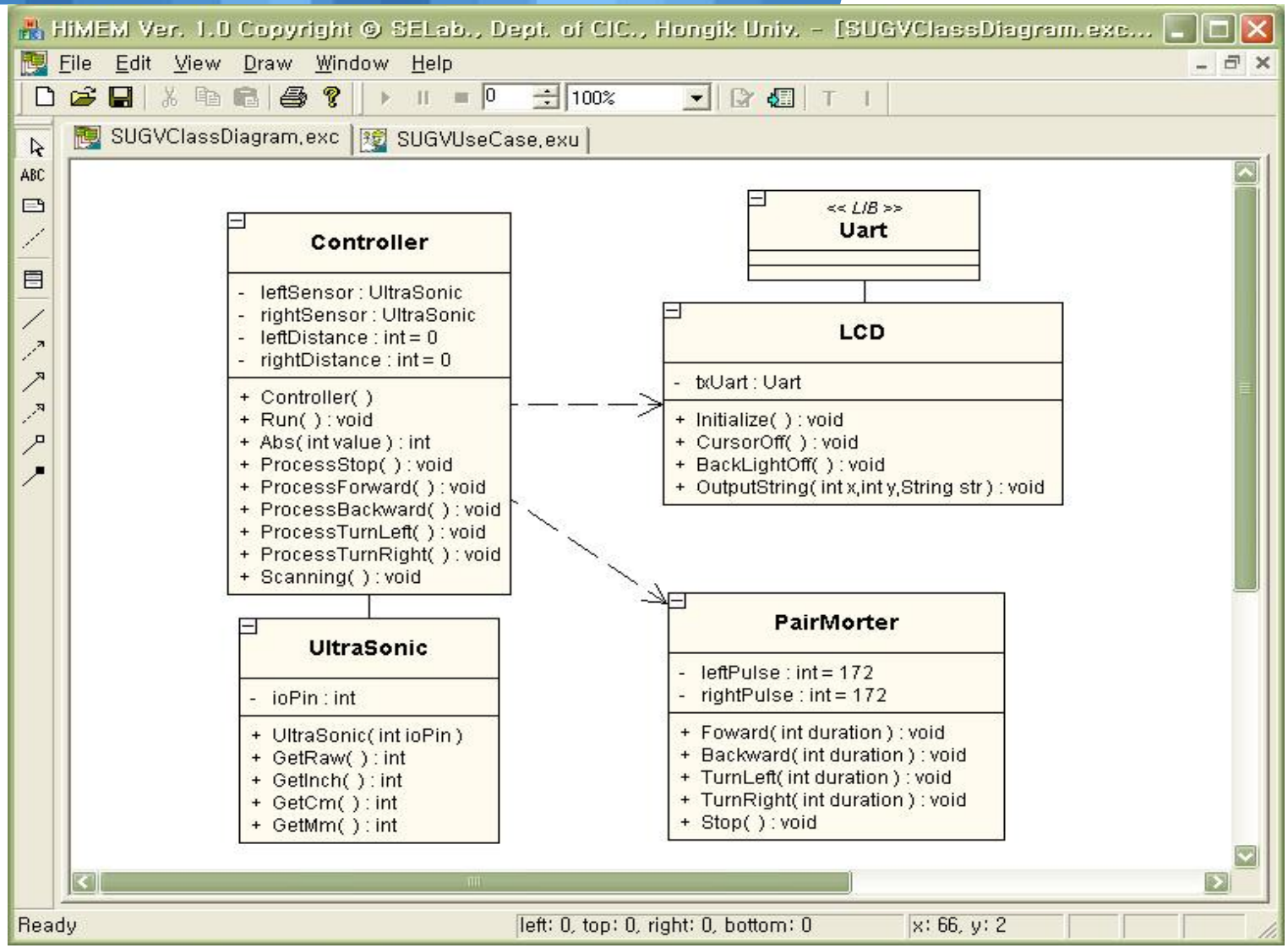
Use case 1

Use Case	장애물 감지
Purpose	주변 상황을 인지/파악
Primary Scenario	**1.** 전진하면서 센서의 상태를 감시한다. **2. 50cm** 이내에 장애물이 감지된다. **3.** 초음파 센서로부터 거리를 얻어온다. **4.** 거리를 판단하고 장애물 회피한다.
정확성	장애물과 탐색장비 사이의 거리를 정확히 측정한다.
안정성	장애물과 충돌하지 않도록 반응이 정확해야 한다.

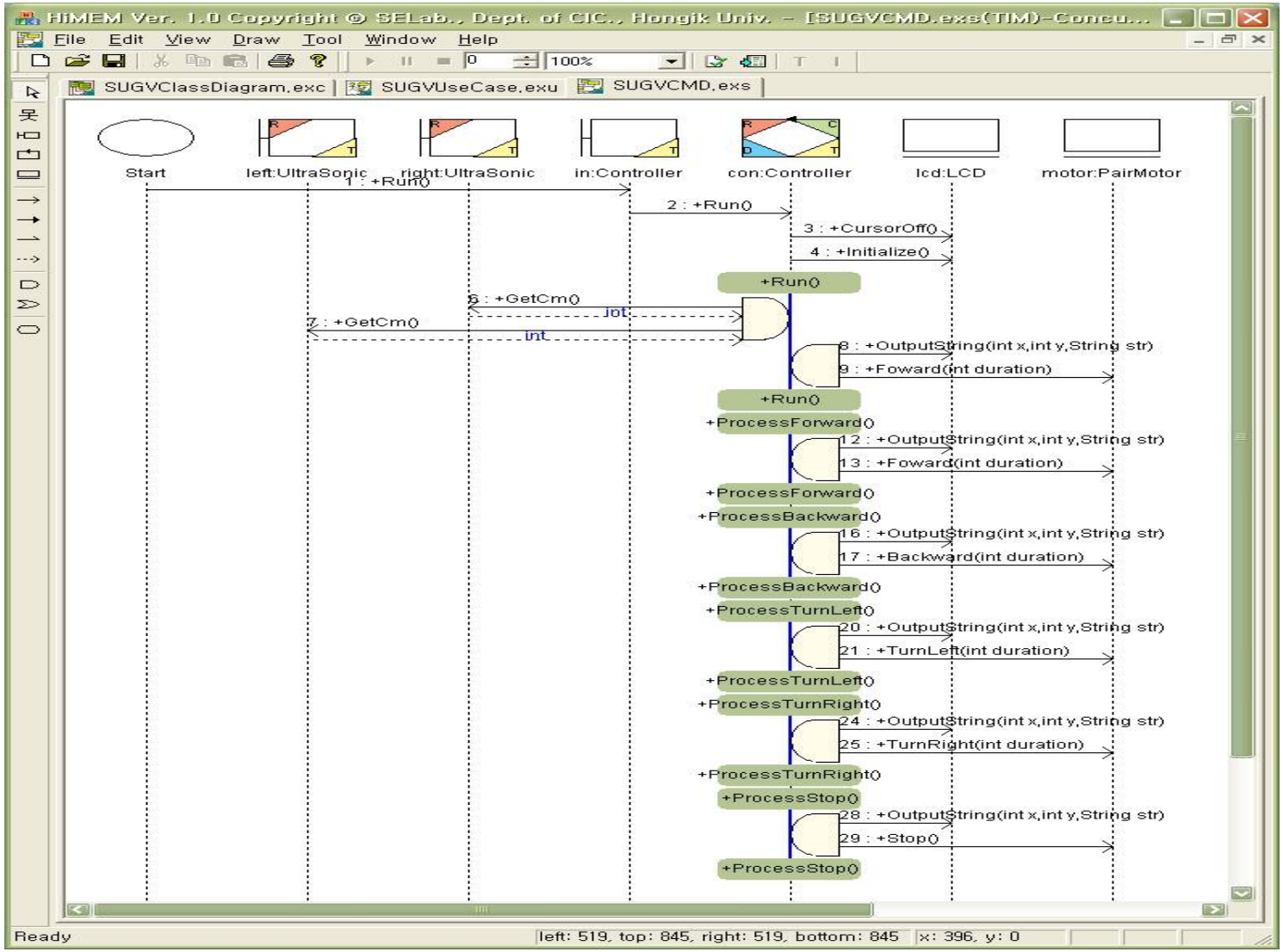
Use case 2

Use Case	장애물 회피
Purpose	물체를 발견하고 안전하게 물체를 회피 한다
Sub Scenario	**1.** 후진을 명령한다. **2.** 후진한다. **3.** 우측으로 **30**도 회전을 명령한다. **4.** 좌/우측 **Wheel**의 속도를 제어한다. **5.** 전진 상태로 전환한다. **6.** 전진한다.
정확성	정해진 각도만큼 정확히 회전한다.
안정성	장애물을 완전히 회피할 때 까지 우회를 반복한다.

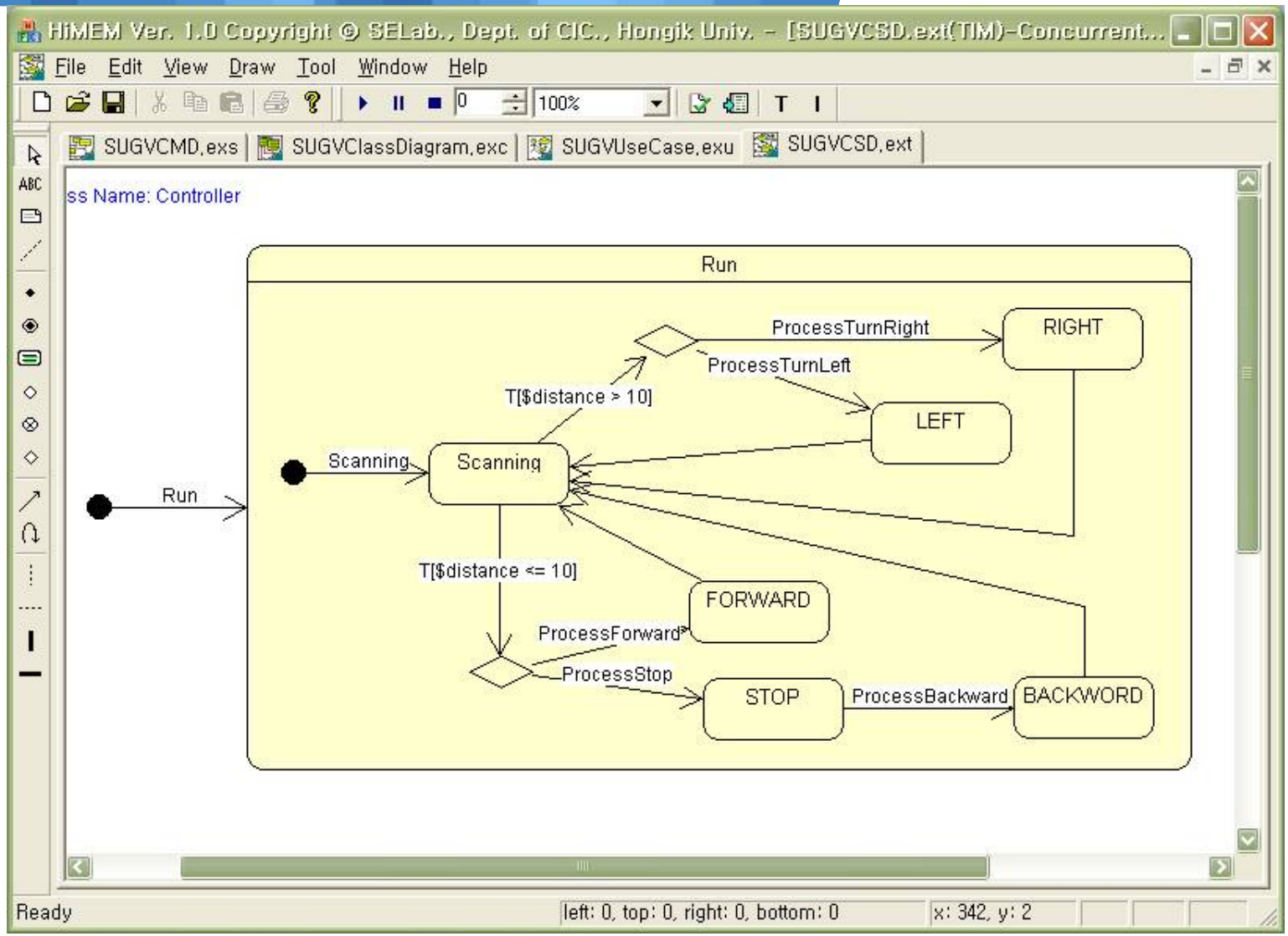
클래스 다이어그램


HiMEM Ver. 1.0 Copyright © SELab., Dept. of CIC., Hongik Univ. - [SUGVClassDiagram.exc...
File Edit View Draw Window Help
SUGVClassDiagram.exc
SUGVUseCase.exu
Controller
- leftSensor : UltraSonic
- rightSensor : UltraSonic
- leftDistance : int = 0
- rightDistance : int = 0
+ Controller( )
+ Run( ) : void
+ Abs(int value) : int
+ ProcessStop( ) : void
+ ProcessForward( ) : void
+ ProcessBackward( ) : void
+ ProcessTurnLeft( ) : void
+ ProcessTurnRight( ) : void
+ Scanning( ) : void
<< LIB >>
Uart
LCD
- txUart : Uart
+ Initialize( ) : void
+ CursorOff( ) : void
+ BackLightOff( ) : void
+ OutputString(int x,int y,String str) : void
UltraSonic
- ioPin : int
+ UltraSonic(int ioPin)
+ GetRaw( ) : int
+ GetInch( ) : int
+ GetCm( ) : int
+ GetMm( ) : int
PairMorter
- leftPulse : int = 172
- rightPulse : int = 172
+ Foward(int duration) : void
+ Backward(int duration) : void
+ TurnLeft(int duration) : void
+ TurnRight(int duration) : void
+ Stop( ) : void
Ready
left: 0, top: 0, right: 0, bottom: 0
x: 66, y: 2

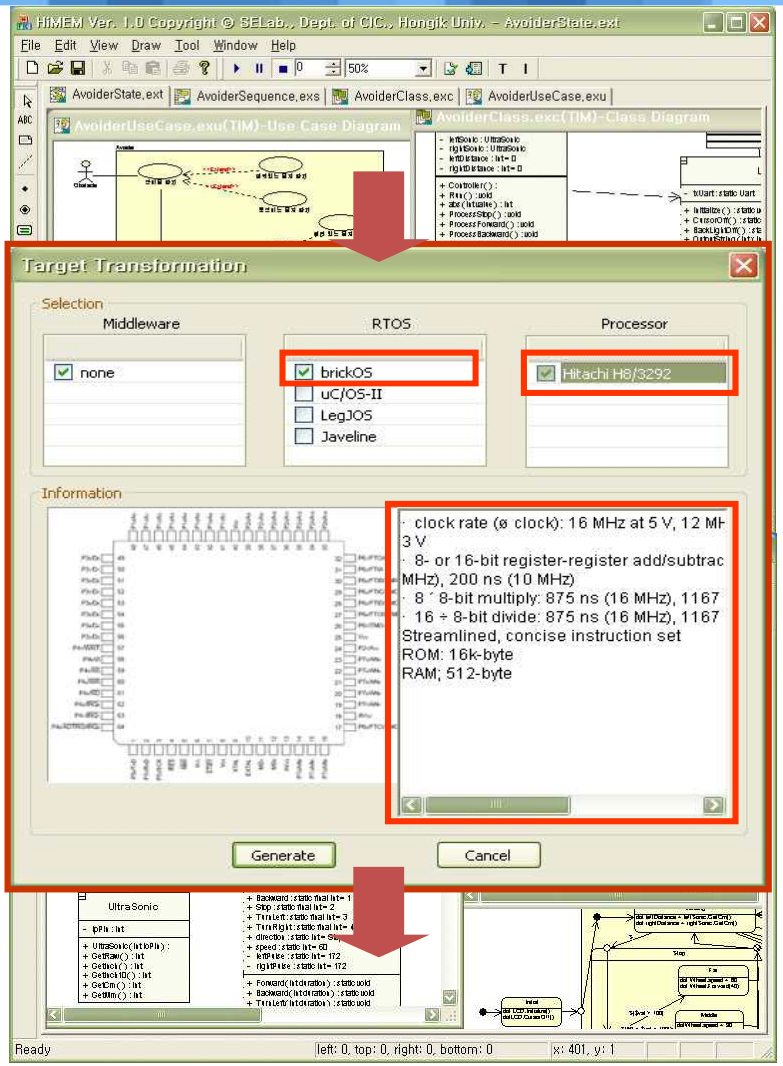
병렬 메시지 다이어그램


HiMEM Ver. 1.0 Copyright © SELab., Dept. of CIC., Hongik Univ. - [SUGVCMD.exs(TIM)-Concu...
File Edit View Draw Tool Window Help
SUGVClassDiagram.exc
SUGVUseCase.exu
SUGVCMD.exs
Start
left:UltraSonic
right:UltraSonic
in:Controller
con:Controller
lcd:LCD
motor:PairMotor
1 : +Run()
2 : +Run()
3 : +CursorOff()
4 : +Initialize()
+Run()
6 : +GetCm()
int
7 : +GetCm()
int
8 : +OutputString(int x,int y,String str)
9 : +Foward(int duration)
+Run()
+ProcessForward()
12 : +OutputString(int x,int y,String str)
13 : +Foward(int duration)
+ProcessForward()
+ProcessBackward()
16 : +OutputString(int x,int y,String str)
17 : +Backward(int duration)
+ProcessBackward()
+ProcessTurnLeft()
20 : +OutputString(int x,int y,String str)
21 : +TurnLeft(int duration)
+ProcessTurnLeft()
+ProcessTurnRight()
24 : +OutputString(int x,int y,String str)
25 : +TurnRight(int duration)
+ProcessTurnRight()
+ProcessStop()
28 : +OutputString(int x,int y,String str)
29 : +Stop()
+ProcessStop()
Ready
left: 519, top: 845, right: 519, bottom: 845
x: 396, y: 0

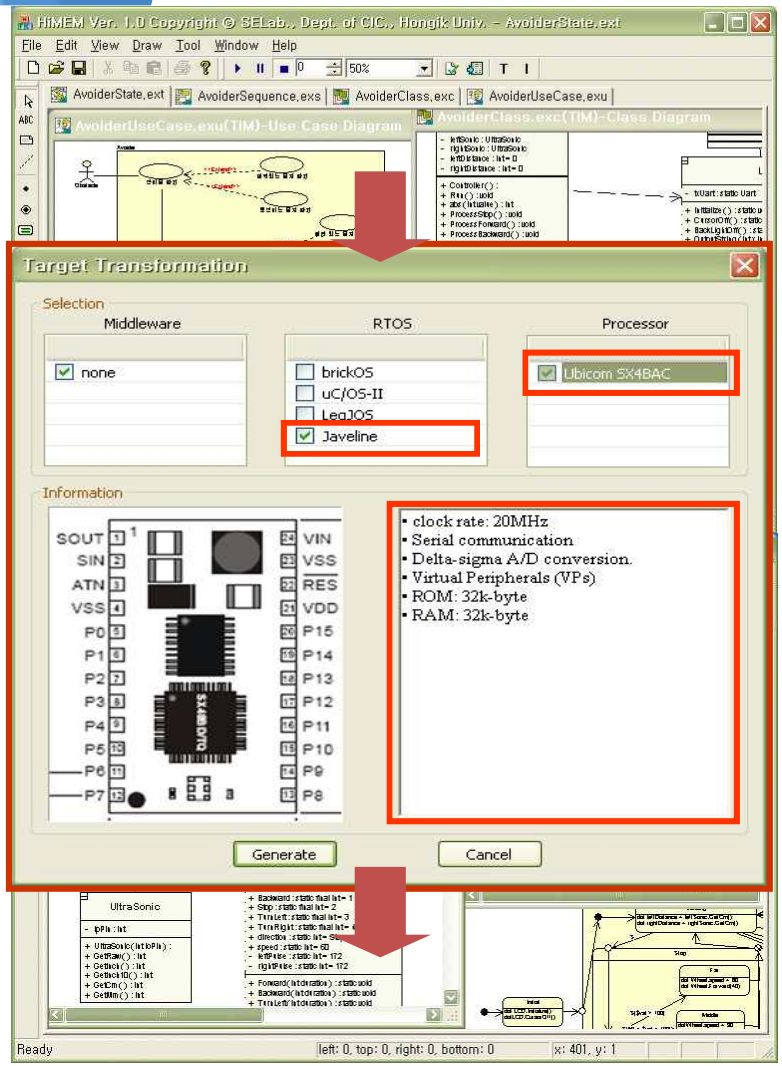
병렬 상태 다이어그램


HiMEM Ver. 1.0 Copyright © SELab., Dept. of CIC., Hongik Univ. - [SUGVCSD.ext(TIM)-Concurrent...
File Edit View Draw Tool Window Help
SUGVCMD.exs
SUGVClassDiagram.exc
SUGVUseCase.exu
SUGVCSD.ext
ss Name: Controller
Run
ProcessTurnRight
RIGHT
ProcessTurnLeft
T[$distance > 10]
LEFT
Scanning
Scanning
Run
T[$distance <= 10]
FORWARD
ProcessForward
ProcessStop
STOP
ProcessBackward
BACKWORD
Ready
left: 0, top: 0, right: 0, bottom: 0
x: 342, y: 2

TIM → TSM 변환 – T1

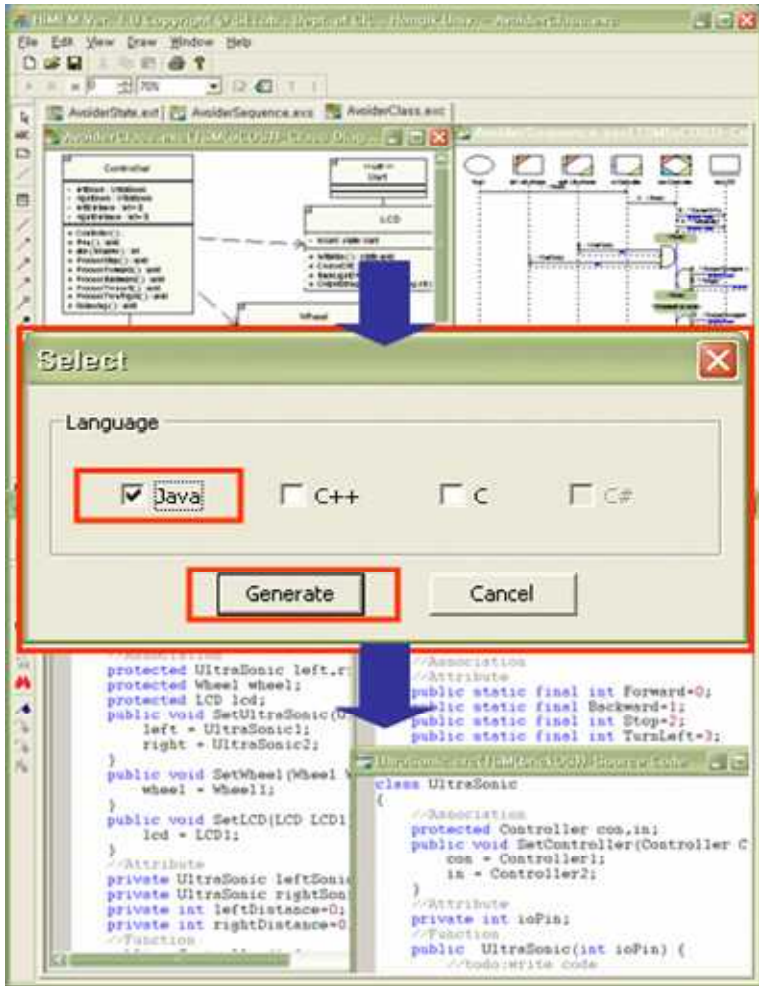


(a) SUGV 1

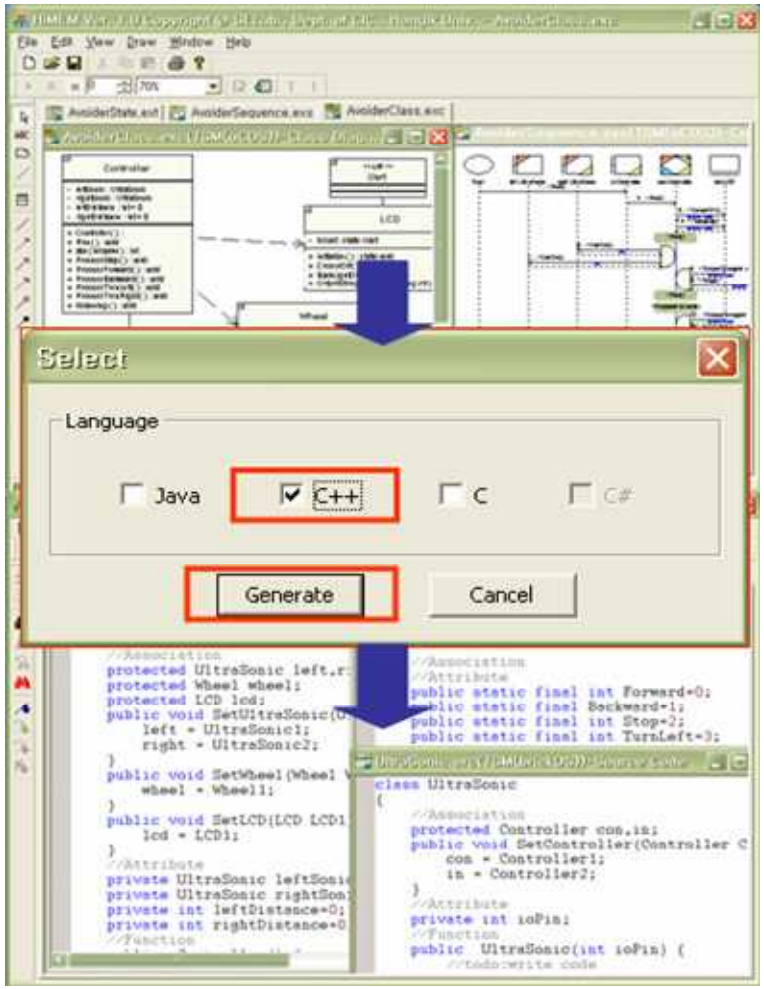


(b) SUGV2

TSM → TDC 변환 – T2

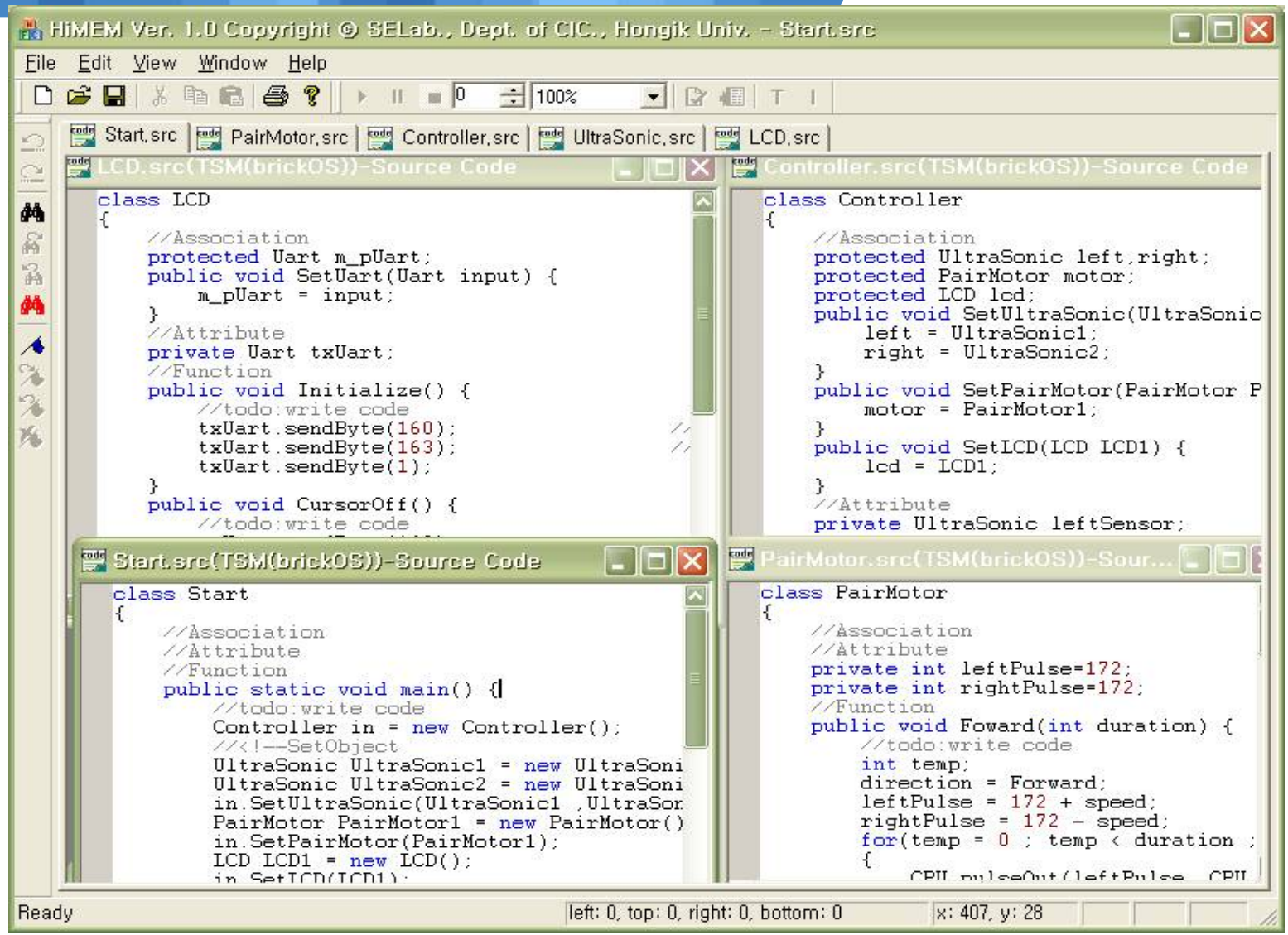


(a) SUGV 1

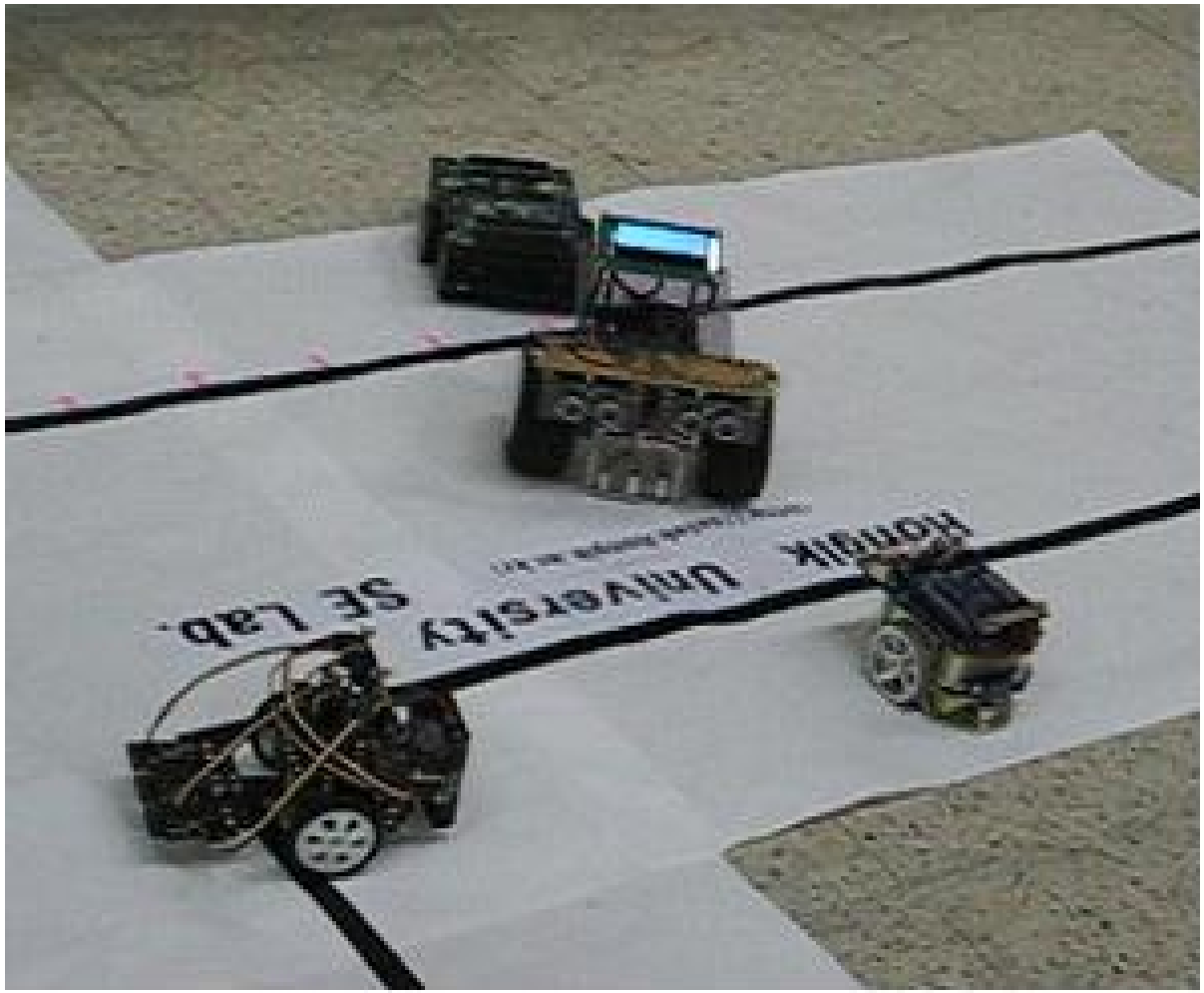


(b) SUGV2

코드생성


HiMEM Ver. 1.0 Copyright © SELab., Dept. of CIC., Hongik Univ. - Start.src
File Edit View Window Help
Start.src PairMotor.src Controller.src UltraSonic.src LCD.src
LCD.src(TSM(brickOS))-Source Code
class LCD
{
//Association
protected Uart m_pUart;
public void SetUart(Uart input) {
m_pUart = input;
}
//Attribute
private Uart txUart;
//Function
public void Initialize() {
//todo:write code
txUart.sendByte(160);
txUart.sendByte(163);
txUart.sendByte(1);
}
public void CursorOff() {
//todo:write code
Controller.src(TSM(brickOS))-Source Code
class Controller
{
//Association
protected UltraSonic left,right;
protected PairMotor motor;
protected LCD lcd;
public void SetUltraSonic(UltraSonic
left = UltraSonic1;
right = UltraSonic2;
}
public void SetPairMotor(PairMotor P
motor = PairMotor1;
}
public void SetLCD(LCD LCD1) {
lcd = LCD1;
}
//Attribute
private UltraSonic leftSensor;
Start.src(TSM(brickOS))-Source Code
class Start
{
//Association
//Attribute
//Function
public static void main() {
//todo:write code
Controller in = new Controller();
//<!--SetObject
UltraSonic UltraSonic1 = new UltraSoni
UltraSonic UltraSonic2 = new UltraSoni
in.SetUltraSonic(UltraSonic1 ,UltraSor
PairMotor PairMotor1 = new PairMotor()
in.SetPairMotor(PairMotor1);
LCD LCD1 = new LCD();
PairMotor.src(TSM(brickOS))-Sour...
class PairMotor
{
//Association
//Attribute
private int leftPulse=172;
private int rightPulse=172;
//Function
public void Foward(int duration) {
//todo:write code
int temp;
direction = Forward;
leftPulse = 172 + speed;
rightPulse = 172 - speed;
for(temp = 0 ; temp < duration ;
{
Ready
left: 0, top: 0, right: 0, bottom: 0
x: 407, y: 28

수행 결과



Lecture 2
UML 기반 모델링

확장 임베디드 UML 소개

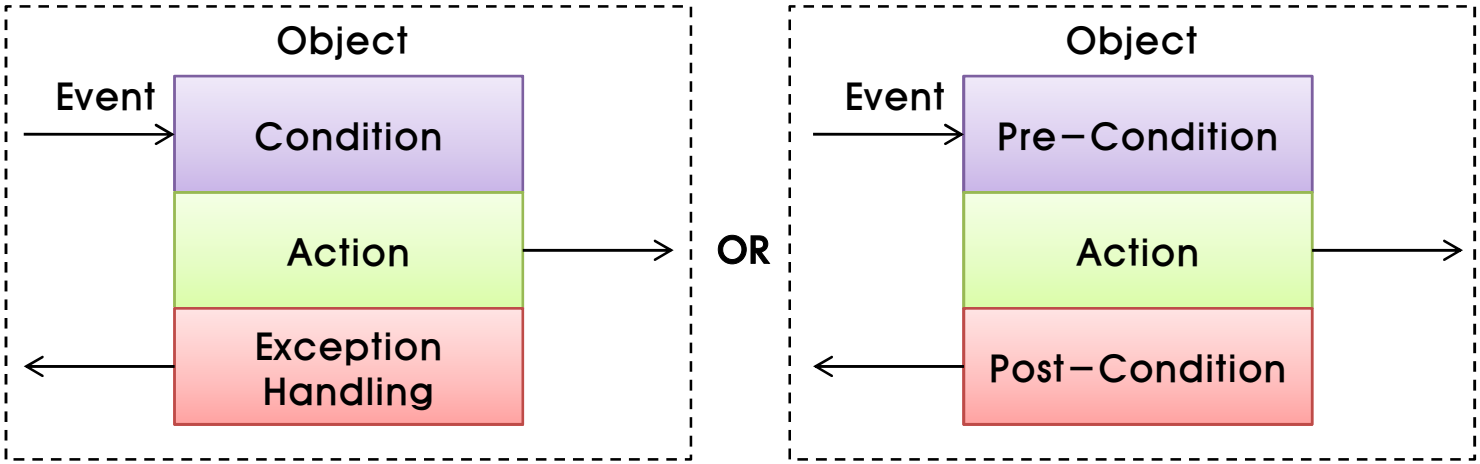
Extended xUML

❖ 배경

- xUML 조차도 표현하지 못하는 부분이 있기에 xUML을 확장

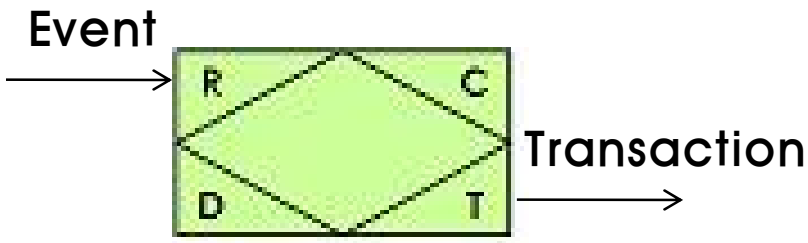
❖ 확장된 xUML에서 클래스 다이어그램

- 역할(Role)과 규칙(Rule)을 포함하며 시스템의 정적 구조를 묘사
- 기존의 객체(Object) 메커니즘 확장
  ECA(Event / Condition / Action)



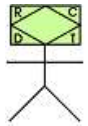
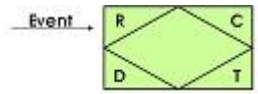
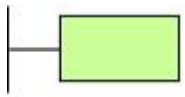
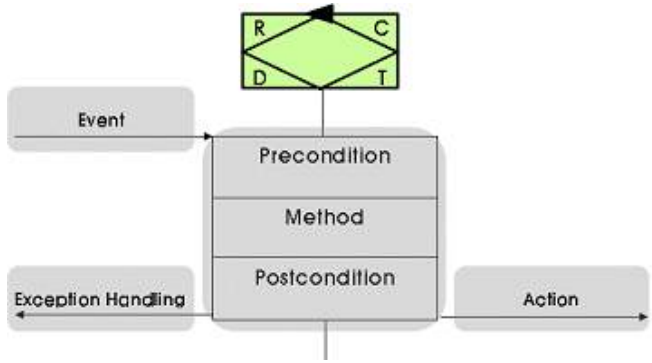
Extended xUML (cont.)

- 객체는 ERCDT(Event/Recognition/Communication/Decision/ Transaction)의 구조를 가짐
  - 인터페이스 객체는 이벤트(Event)가 들어오면 인지(Recognition) 해서 통신(Communication)하는 ERC의 구조
  - 제어 객체는 인터페이스의 기능뿐만 아니라 결정(Decision)을 내리고 수행(Transaction)하는 ERCDT의 모든 구조
  - 서비스 객체는 이벤트(Event)가 들어오면 수행(Transaction)을 하게 되는 ET의 구조



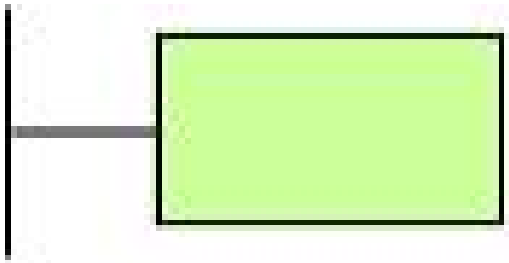
E: Event
R: Recognition
C: Communication
D: Decision
T: Transaction

객체의 표현방법 확장

Element	Notation	Element	Notation
Actor		Object's Role (ERCDT)	
Object			E: Event R: Recognition C: Communication D: Decision T: Transaction
Interface (Boundary) Object		Object's Rule (ECA)	
Control Object			
Entity (Service) Object			

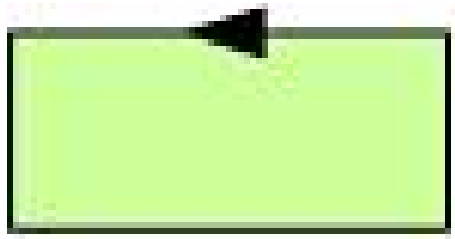
Stereotype of Object

- Interface Object
  - 단지 객체이름과 메소드들로 구성
  - Passive Object



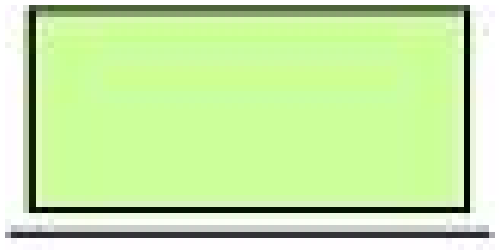
Stereotype of Object (cont.)

- Control Object
  - 객체이름과 속성, 메소드를 모두 존재
  - Active Object



Stereotype of Object (cont.)

- Service(Entity) Object
  - 객체이름과 속성 존재
  - Passive Object
  - Data store



Lecture 2
UML 기반 모델링

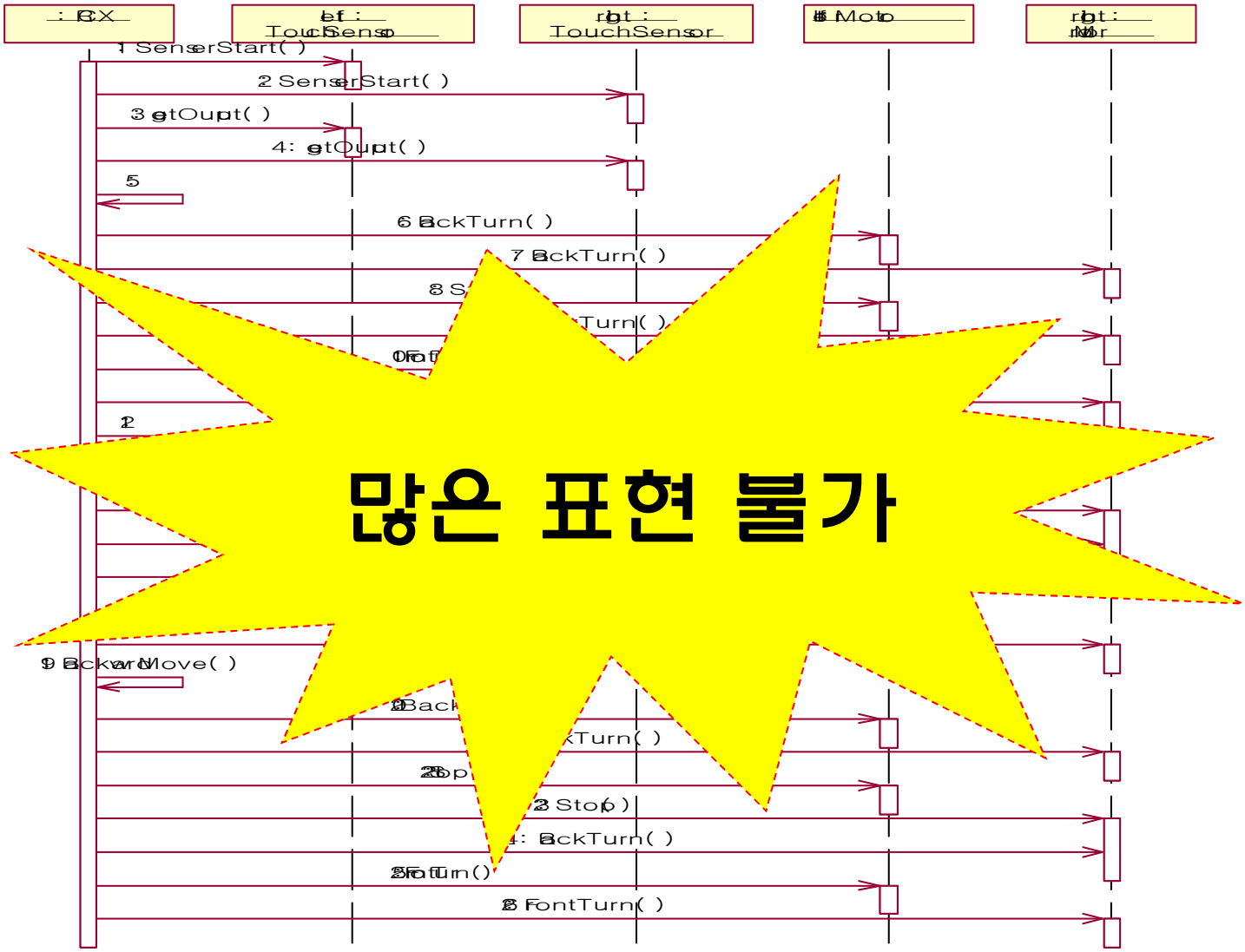
병렬 메시지 다이어그램

Extended Message Sequence Diagram

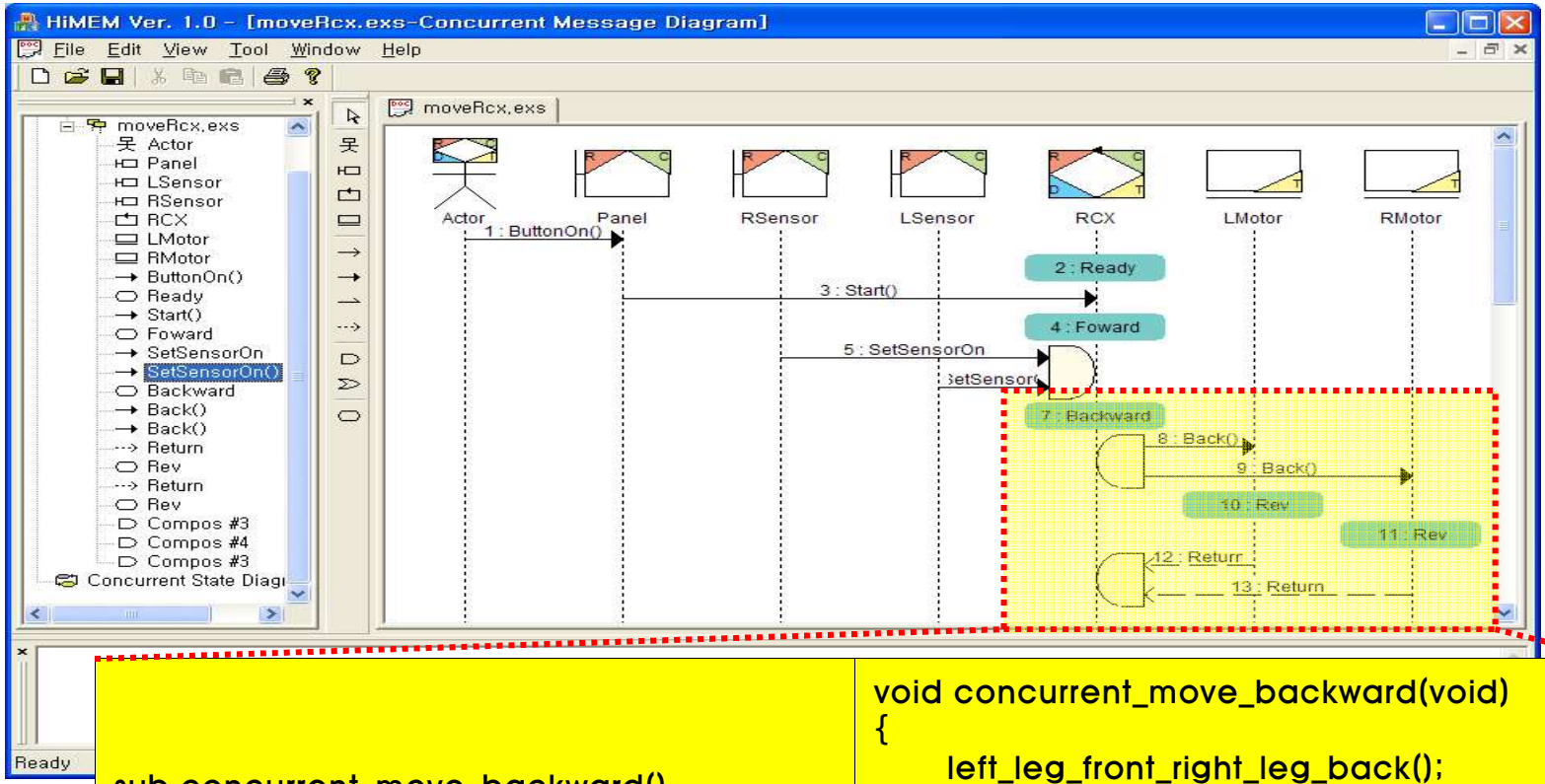
❖ 병렬 메시지 다이어그램(CMD: Concurrent Message Diagram)

- 기존의 시퀀스 다이어그램에 실세계에서 발생될 수 있는 병렬 메커니즘 (fork-join, reverse fork-join 등)을 포함하도록 확장
- 확장된 xUML은 동기 메시지를 기본으로 함
- 각각의 객체에 역할(Role)이 있음
- ECA(Event/Condition/Action) 법칙을 따름

기존 모델링 – 동적


: NXT
TouchSensor
TouchSensor
Motor
Motor
1: SensorStart( )
2: SensorStart( )
3: getOutput( )
4: getOutput( )
5
6: BackTurn( )
7: BackTurn( )
8: S
Turn( )
9: BackwardMove( )
Back
kTurn( )
Stop( )
3: Stop( )
4: BackTurn( )
FontTurn( )
많은 표현 불가

확장된 MSD를 사용한 모델링




```
sub concurrent_move_backward()
{
  OnRev(OUT_A+OUT_C); Wait(MOVE_TIME);
  Wait(MOVE_TIME);
}
```

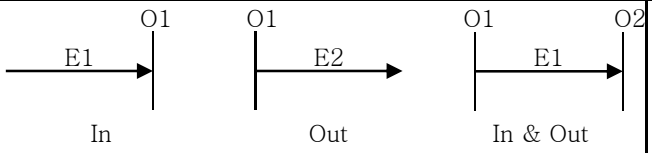
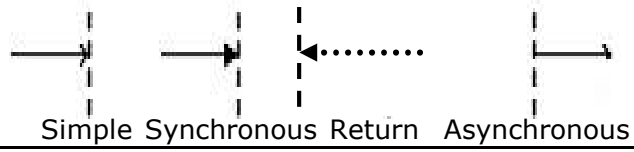
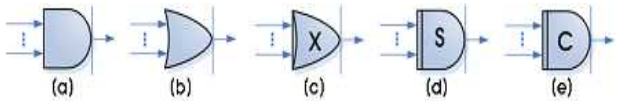
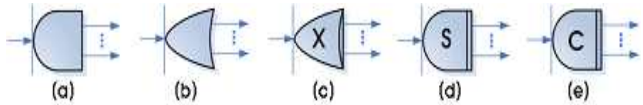
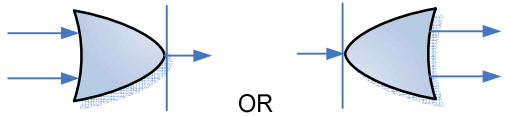
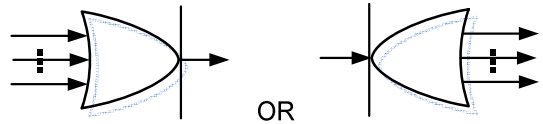
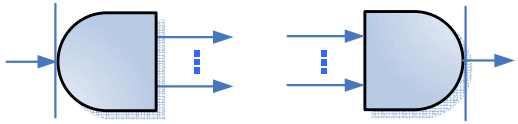
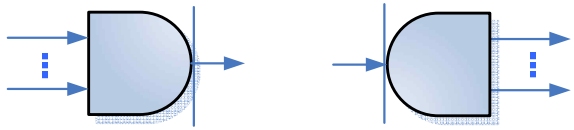
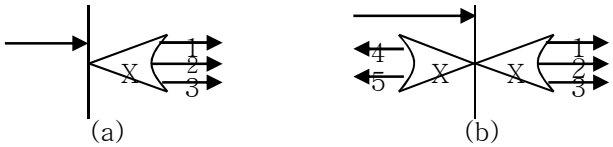
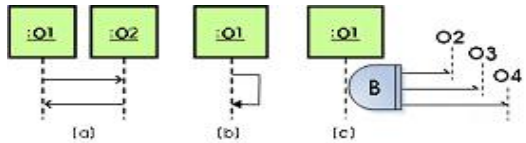
Lego

```
void concurrent_move_backward(void)
{
        left_leg_front_right_leg_back();
        delay(movement_delay);
        delay(movement_delay);

        left_leg_back_right_leg_front();
        delay(movement_delay);
        delay(movement_delay);
}
```

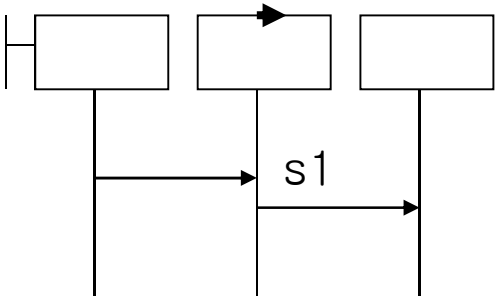
HexAvoider

병렬 메시지 다이어그램 노테이션

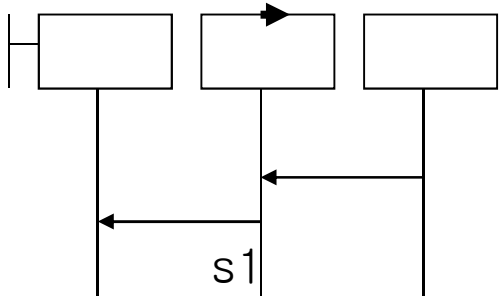
Element	Notation	Element	Notation
Event		Message	
Incoming		Outcoming	
Choice		Multiple Choice	
Fork/Join		Reverse Fork/Join	
Communi-cation		Broad-casting	
Temporal Delay	T	...	...

Message Sequence Diagram과 State Diagram의 접목

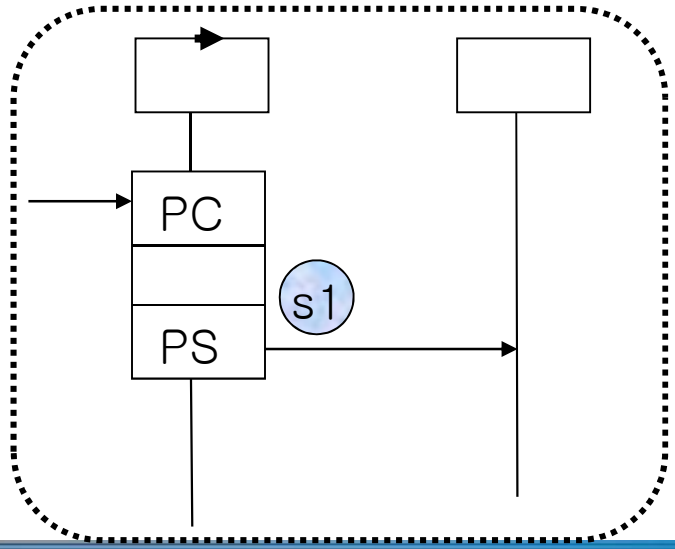
❖ **One Control Object**

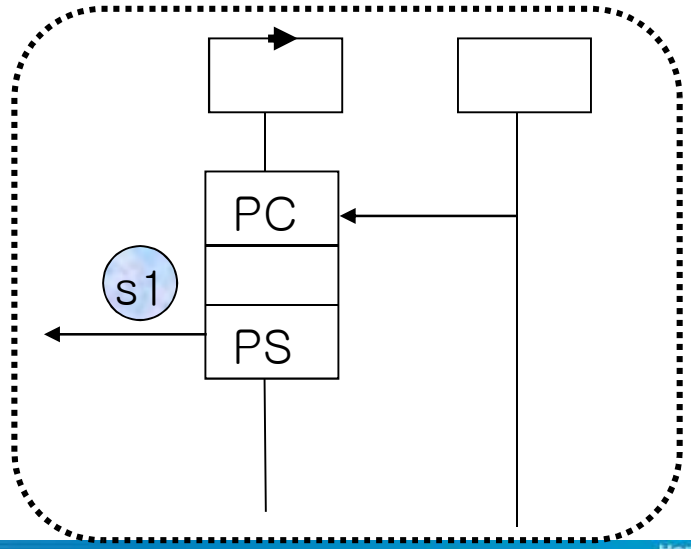


(1)

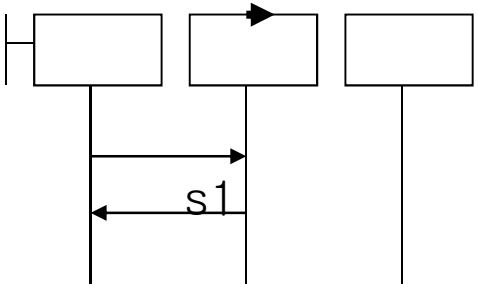


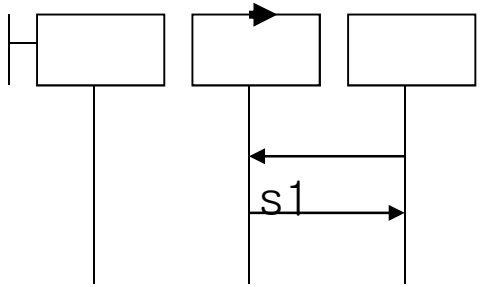
(2)



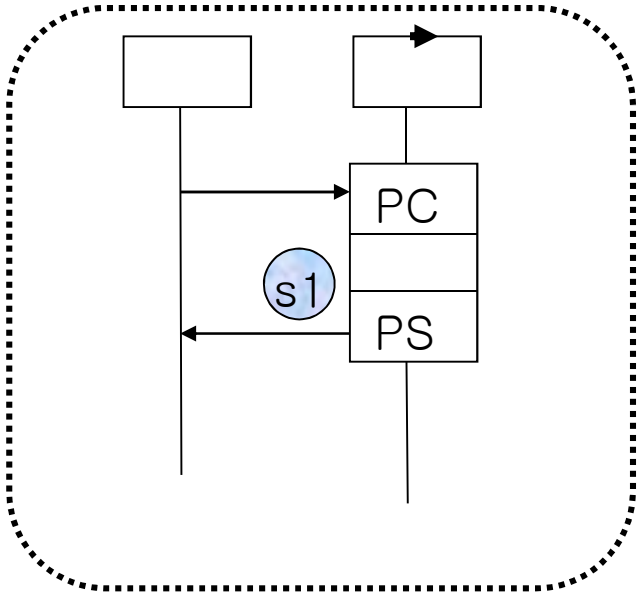


Message Sequence Diagram과 State Diagram의 접목 (cont.)

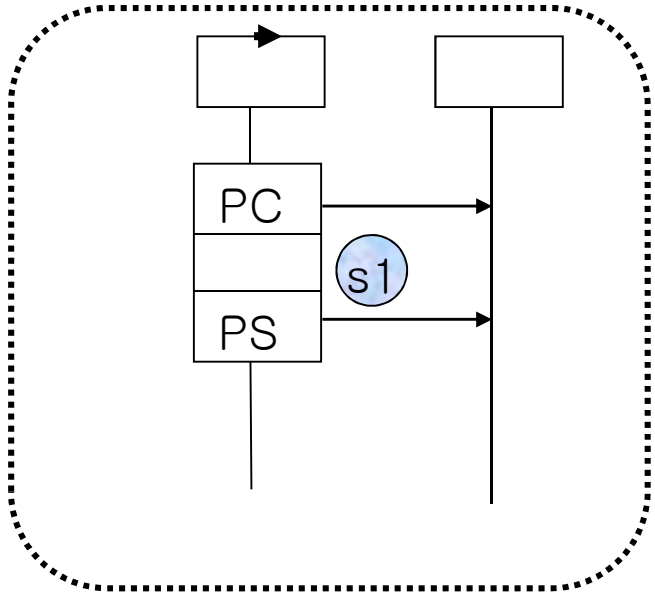




(3)

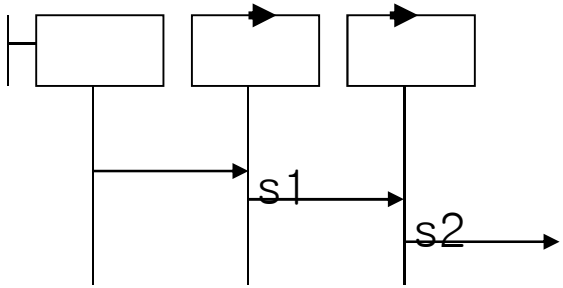


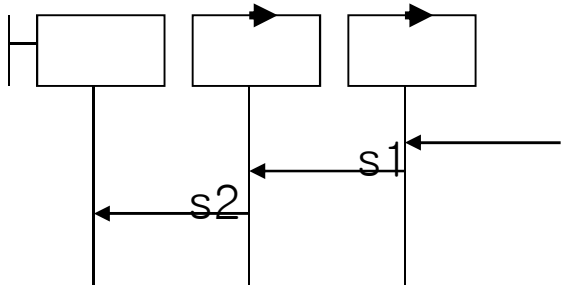
(4)



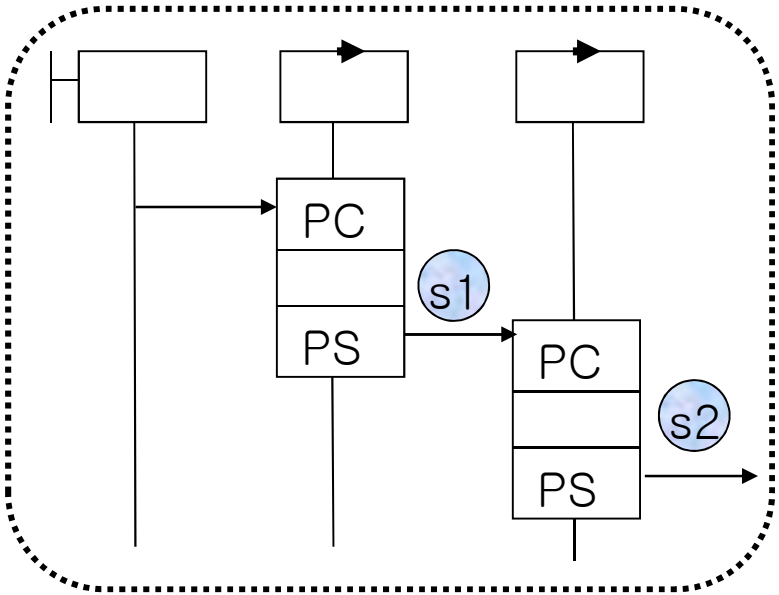
Message Sequence Diagram과 State Diagram의 접목 (cont.)

❖ **More than One Control Object**

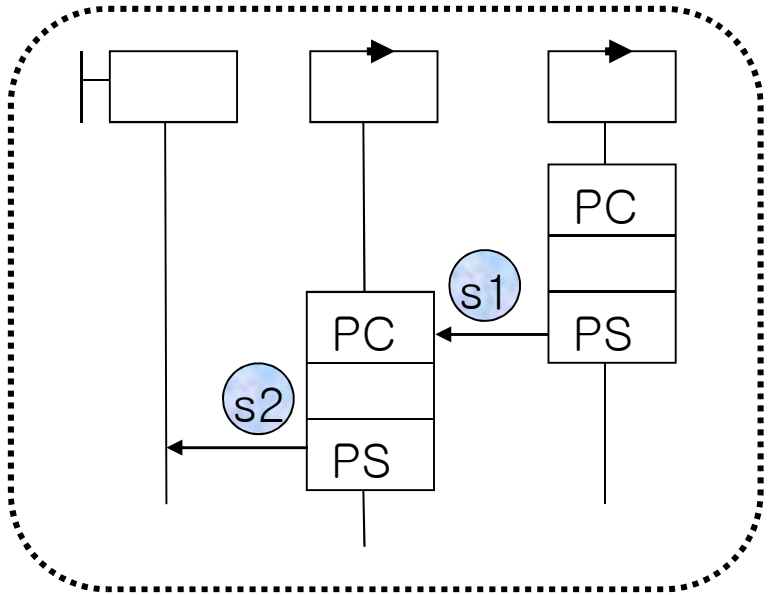




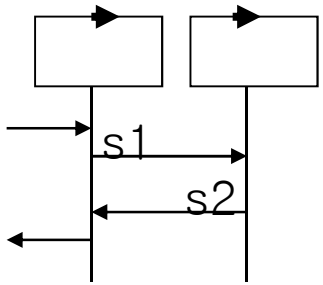
(1)



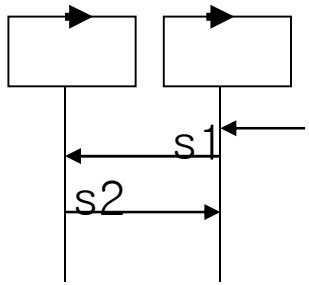
(2)

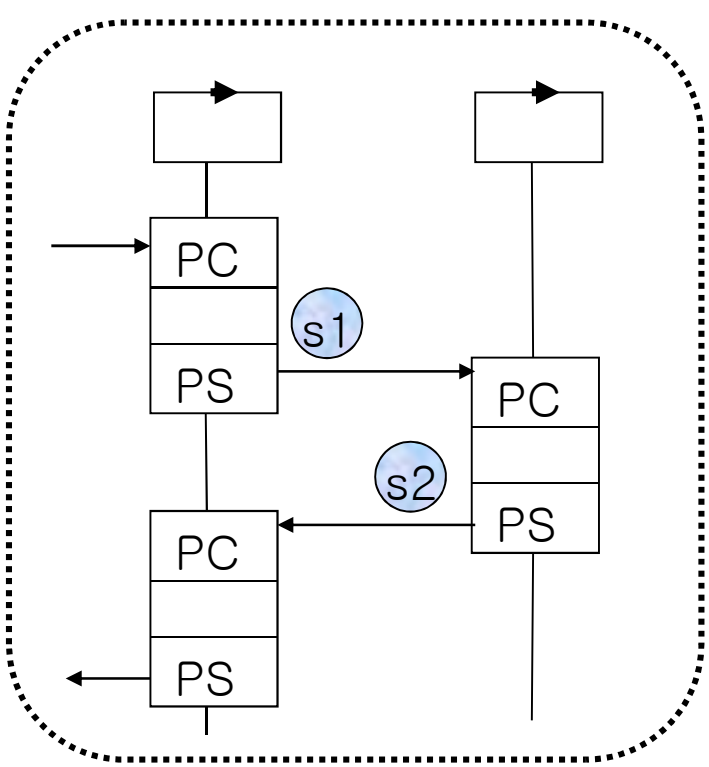


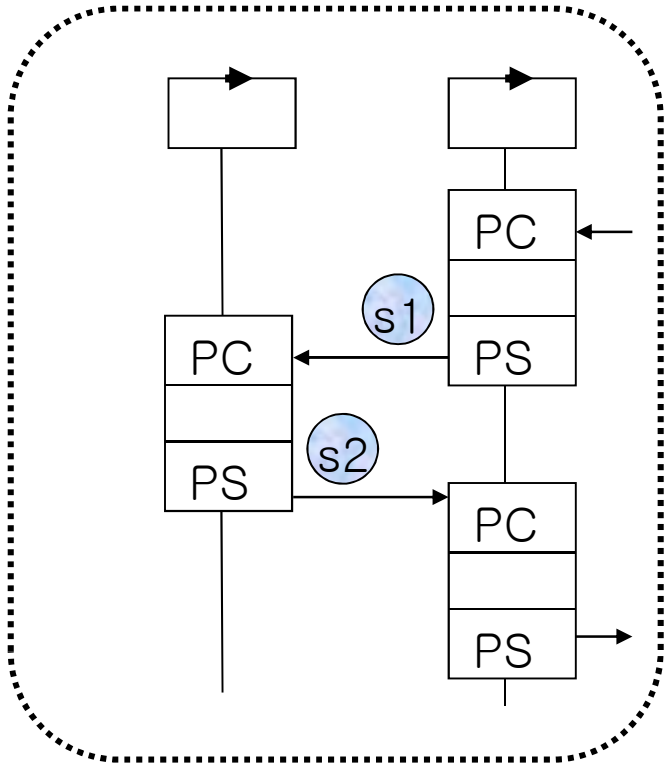
Message Sequence Diagram과 State Diagram의 접목 (cont.)



(3)







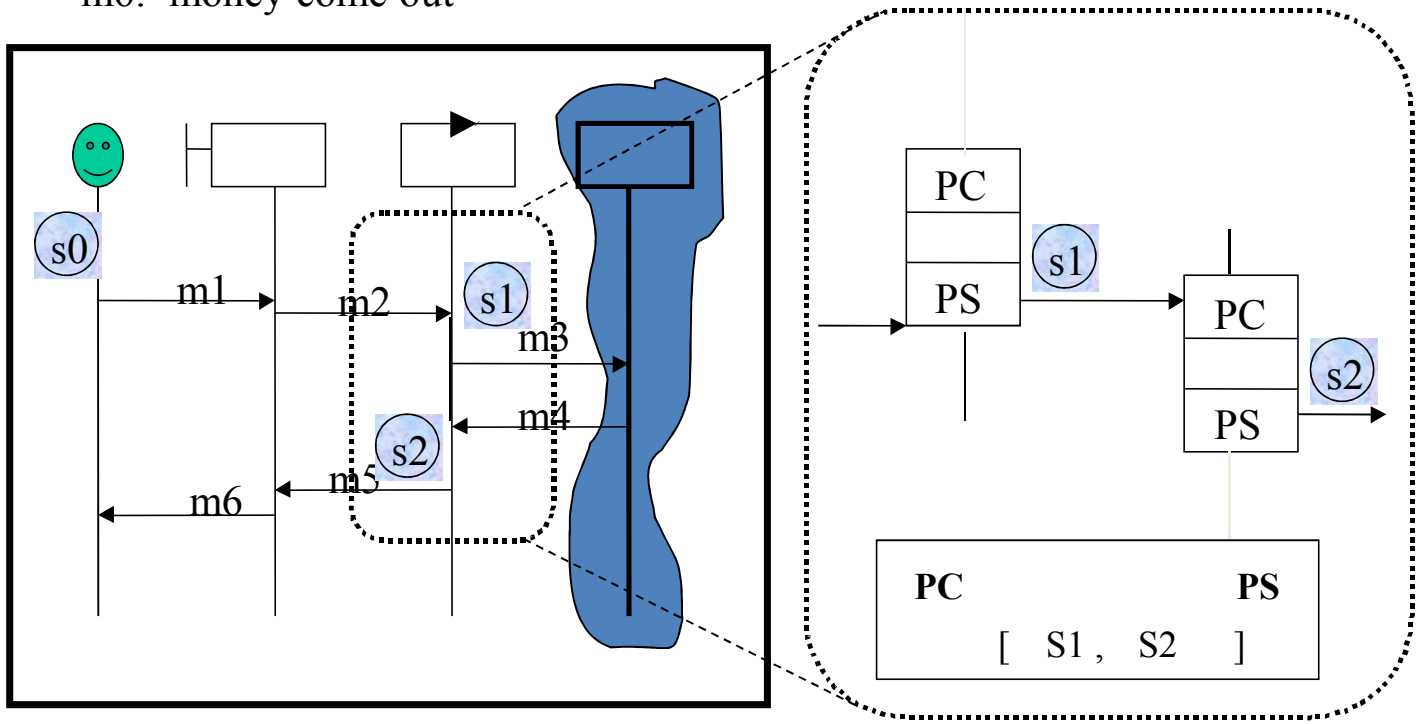
State Mechanism [Carlson99] [Kim99]

Simple ATM Machine:

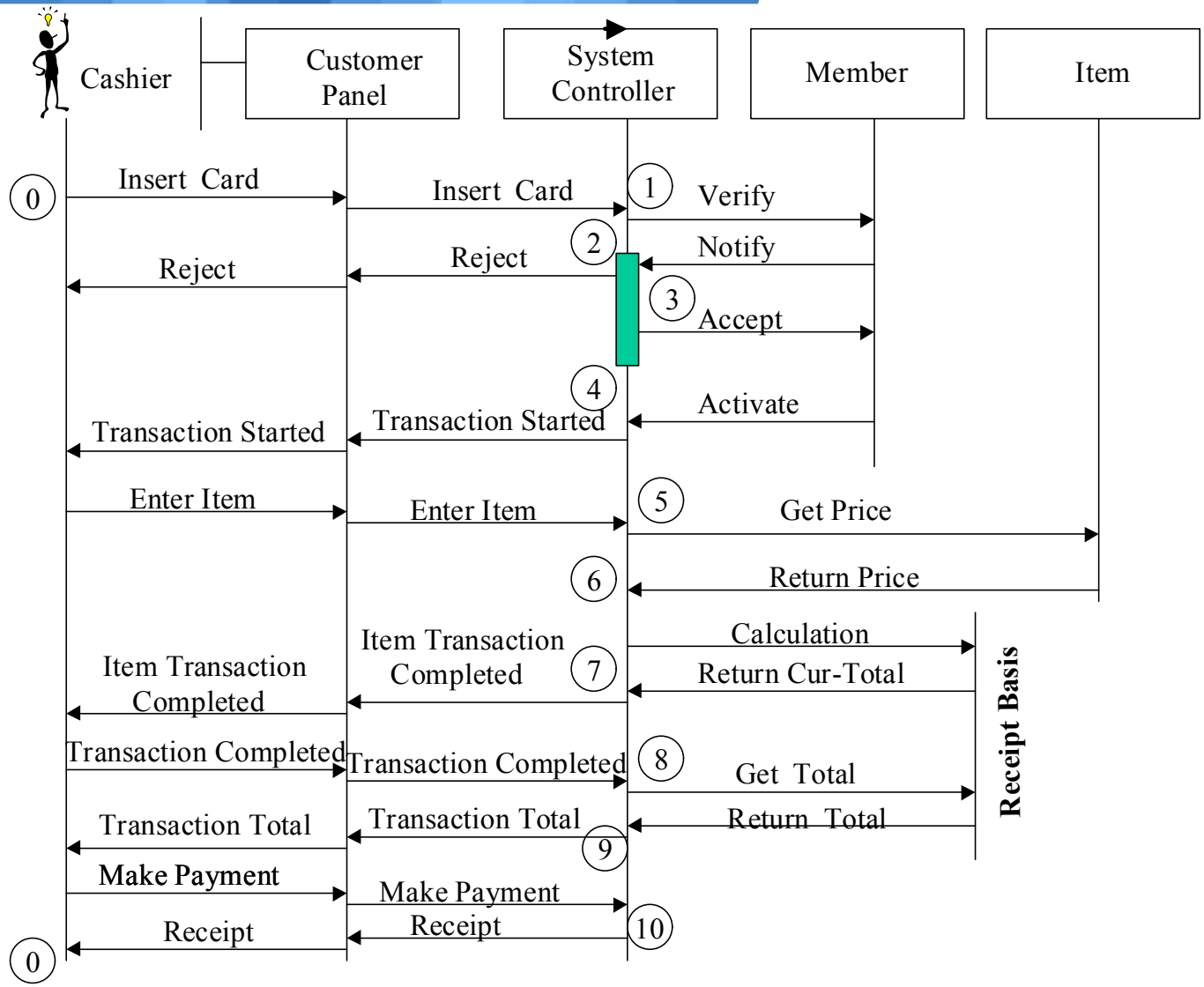
m1: push withdraw button
m2: request withdraw
m3: request a user's status to the main bank computer
m4: check the user's status
m5: withdraw money
m6: money come out

S0: an initial state
S1: a withdraw requesting state
S2: a withdraw state

PC: pre-condition
PS: post-condition



예: State Mechanism [Carlson99] [Kim99]



Lecture 2
UML 기반 모델링

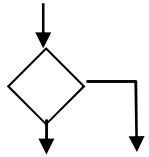
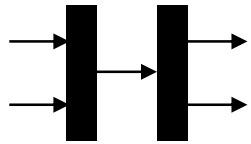
병렬 상태 다이어그램

Extended State Diagram

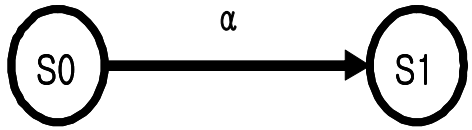
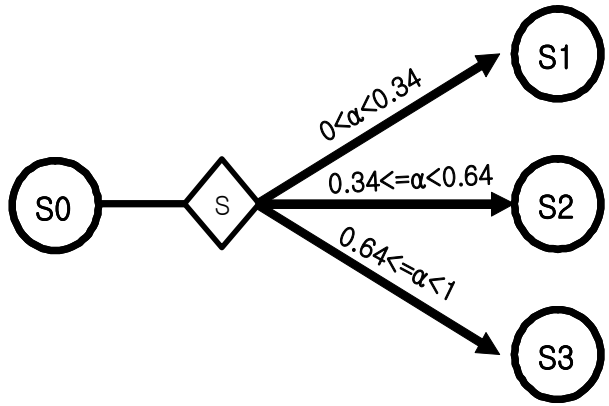
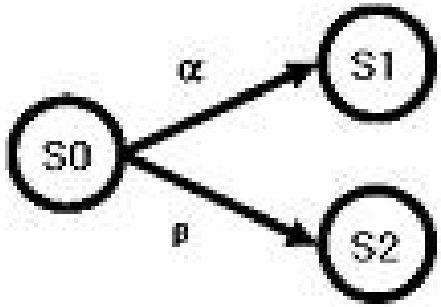
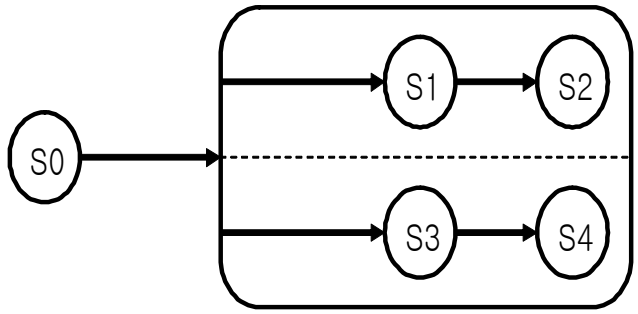
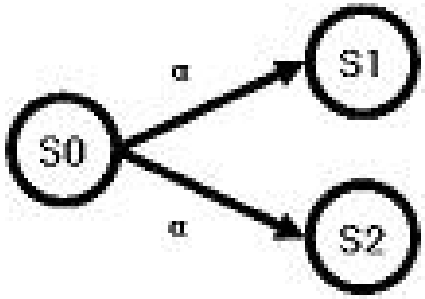
❖ 병렬 상태 다이어그램(CSD: Concurrent State Diagram)

- 기본적으로 OCL(Object Constraint Language)을 포함하여 서술
- Deterministic/Stochastic 메커니즘을 포함
- 향후 Non-deterministic 상태를 Deterministic 상태로 자동 변환

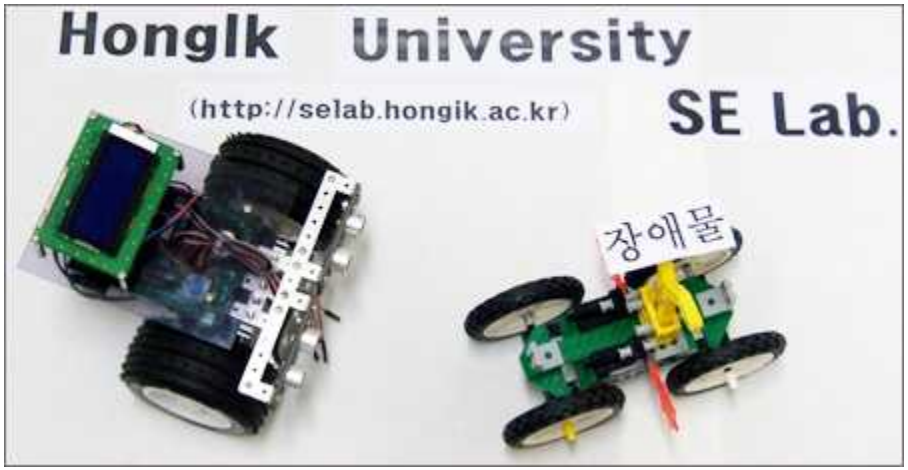
병렬 상태 다이어그램 노테이션

Element	Notation	Element	Notation
Initial State: 상태 머신의 시작을 표시		**Shallow History:** 가장 최근의 스테이트로 돌아감	
Choice Point: 동적 분기 제공		**Join, Fork:** 동기화에 사용됨	
Junction Point: 정적 분기 제공		**Entry, Exit Point:** 합성 상태 또는 상태 머신에 입력과 출력을 지정함	
Deep History: 가장 최근의 서브 스테이트로 돌아감		**Terminate:** 상태 머신의 수행이 끝났을때 표시	

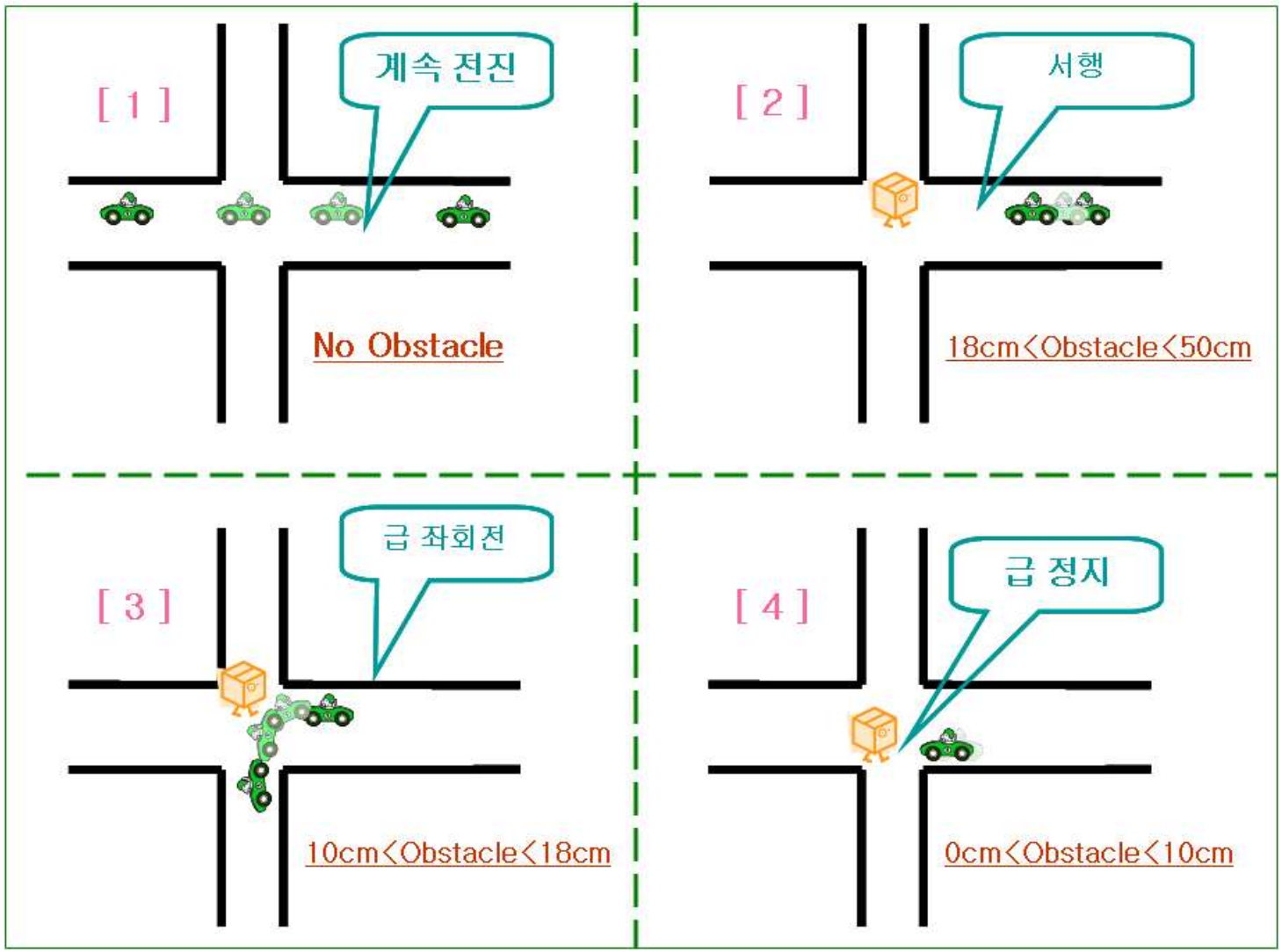
병렬 상태 다이어그램 노테이션

설명	노테이션	설명	노테이션
단일: 단순히 다음 상태로 이동하는 경우의 형태		**확률** 비결정과 비슷한 맥락에서 하나의 출력이라도 랜덤 변수라면 랜덤한 확률적인 값을 갖게 되는 경우에 다음 상태로 전이되는 경우의 형태	
결정:구체적인 상태로의 상태 전이 필요시 사용할 수 있는 결정 선택으로 상태가 이동하는 형태		**동시성**: 여러 개의 쓰레드가 발생되어 동시 수행되는 경우의 형태	
비결정:다음 상태로 이동할 때 무작위로 선택되어 이동하는 경우의 형태			

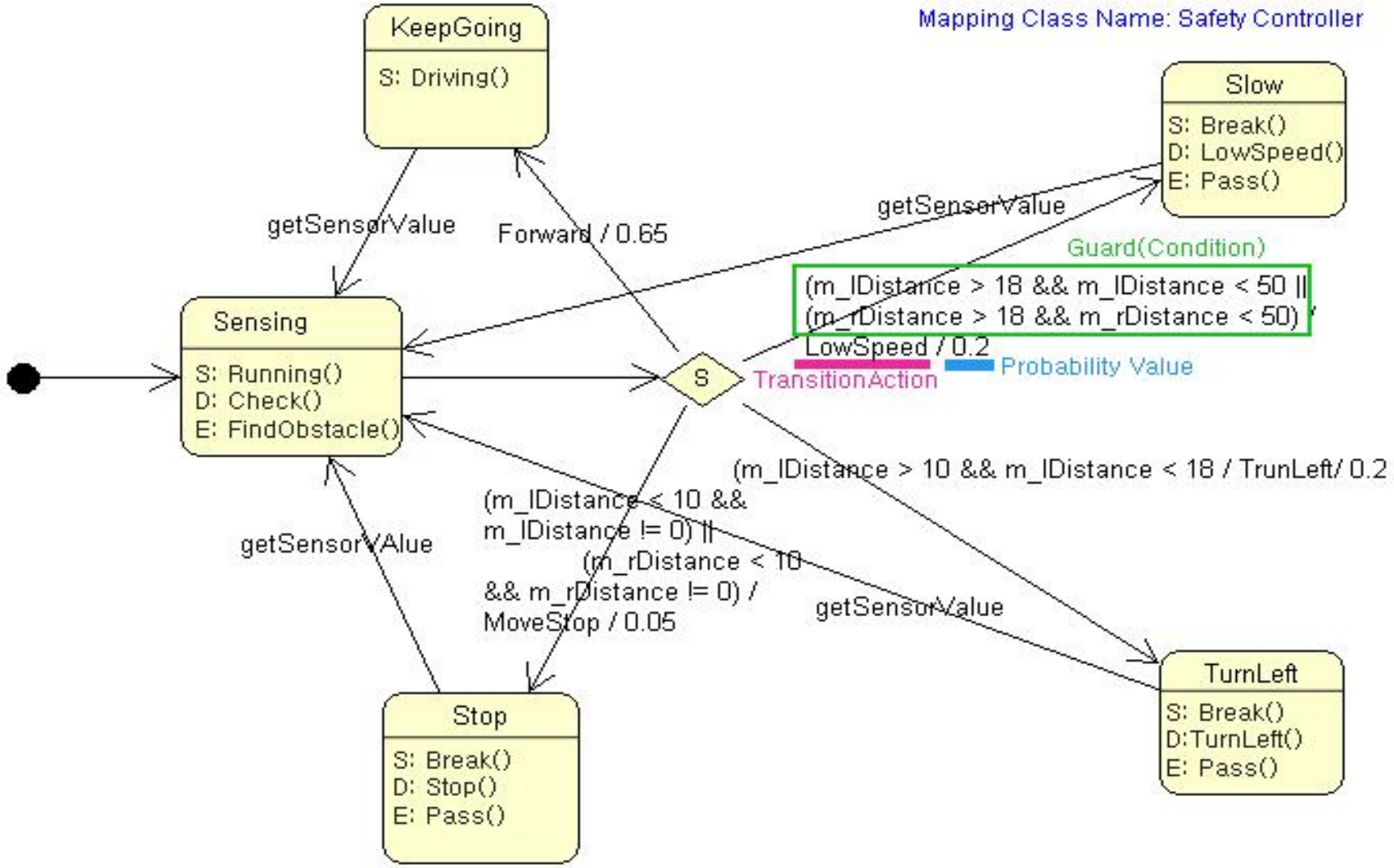
병렬 상태 다이어그램 적용사례 1



시뮬레이션을 위한 시나리오



Safety Controller에 대한 State Diagram



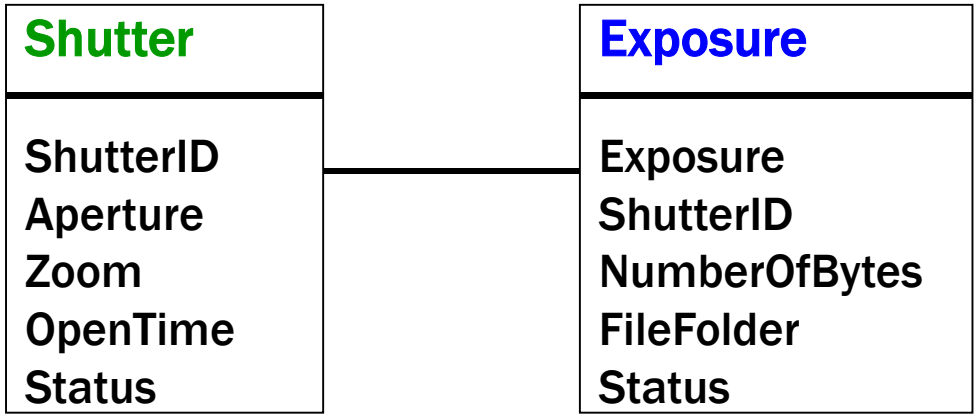
Source code about State Diagram

```
SensorMonitor(50,i); //m_lDistance,m_rDistance < input
        if( (m_lDistance < RANGE_BASE && m_lDistance != 0) ||
            (m_rDistance < RANGE_BASE && m_rDistance != 0) )
        {
            MoveStop(100,i);
            duration = 0;
            leftFlag = false;
            rightFlag = false;
        }
        else if( (m_lDistance > RANGE_BASE &&
                  m_lDistance < RANGE_BASE+RANGE_RATE) ||
                  leftFlag )
        {
            if(!leftFlag)
            {
              duration = 0;
              leftFlag = true;
              rightFlag = false;
            }
            else
            {
              duration++;
            }
            LowSpeed();
            TurnLeft(100,i);
        }
```

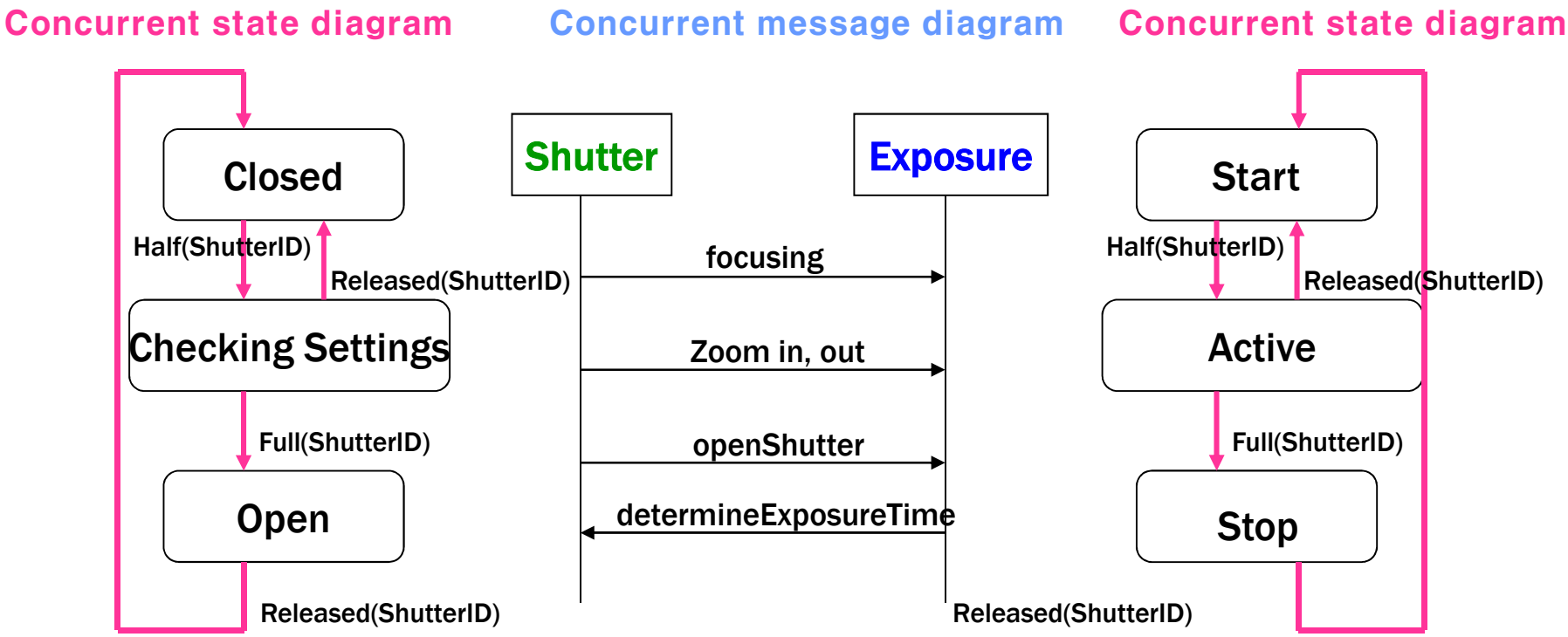
```
        else if( (m_lDistance > RANGE_BASE+RANGE_RATE &&
                  m_lDistance < RANGE_BASE+RANGE_RATE*5) ||
                  (m_rDistance > RANGE_BASE+RANGE_RATE &&
                  m_rDistance < RANGE_BASE+RANGE_RATE*5) )
        {
            LowSpeed();
            Forward(100,i);
        }
        else
        {
            DefSpeed();
            Forward(100,i);
        }
```

Object Lifecycle

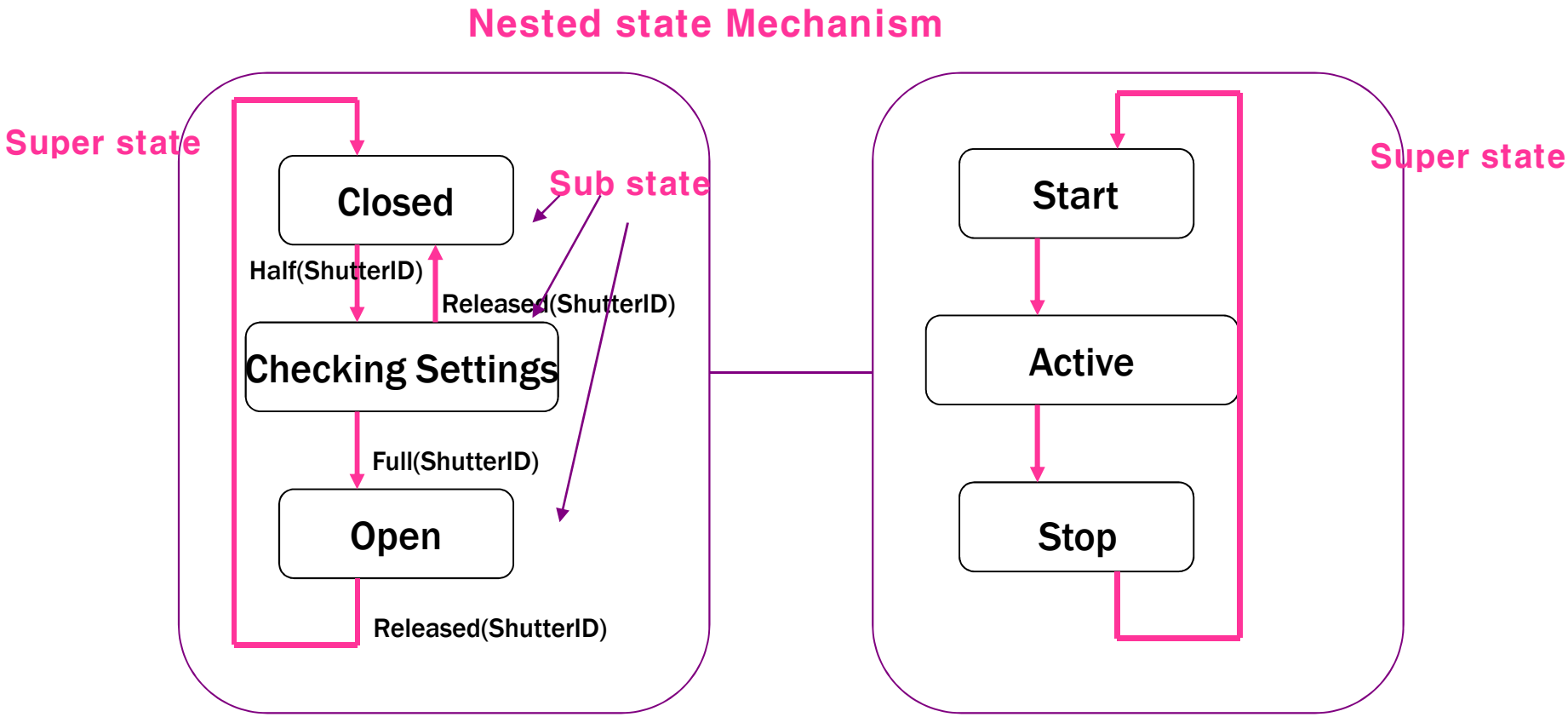
❖ Class diagram



Object Lifecycle (cont.)



Object Lifecycle (cont.)



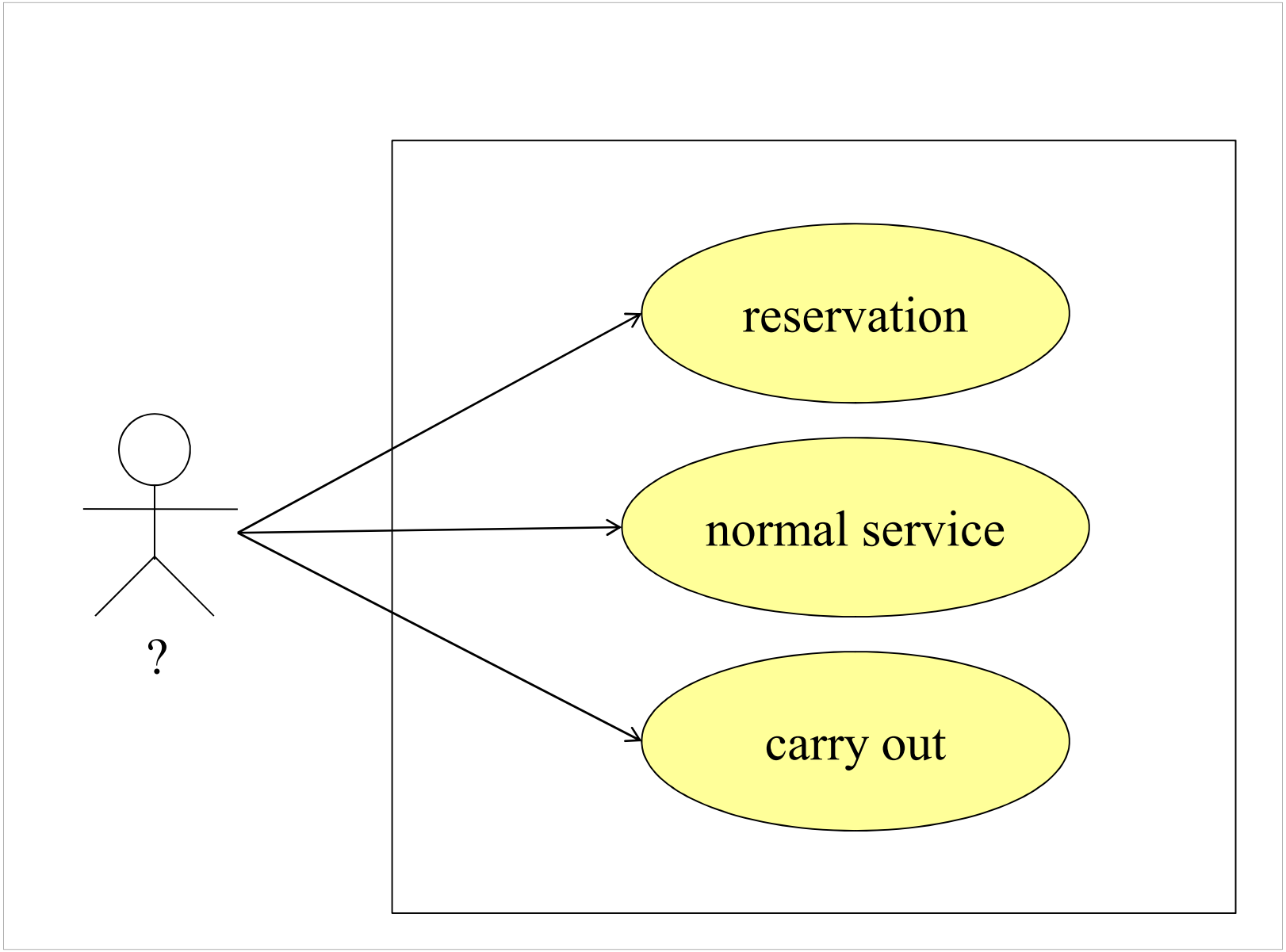
Lecture 2
UML 기반 모델링

설계 사례 및 실습 (Restaurant)

-Use case Diagram

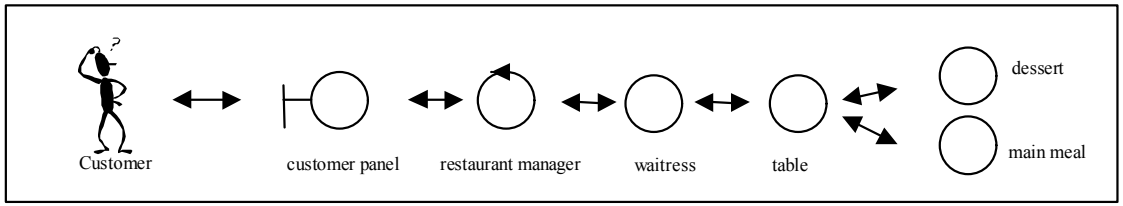
-Message sequence Diagram

1. Use case Diagram

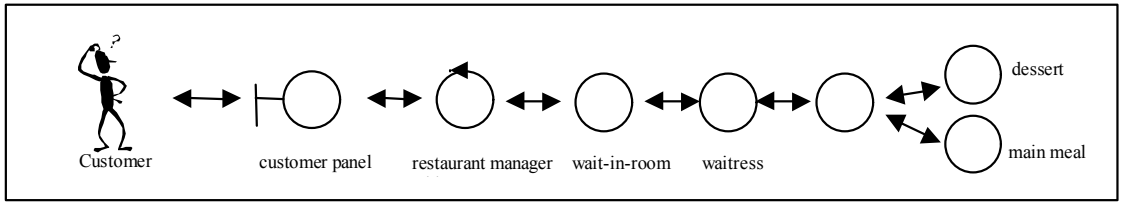


2. Use case Scenarios

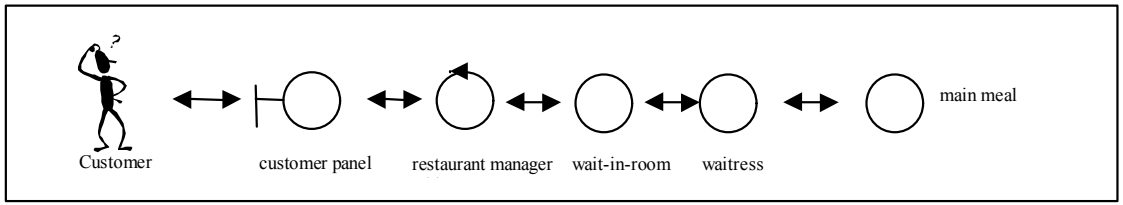
USE CASE SCENARIO : reservation



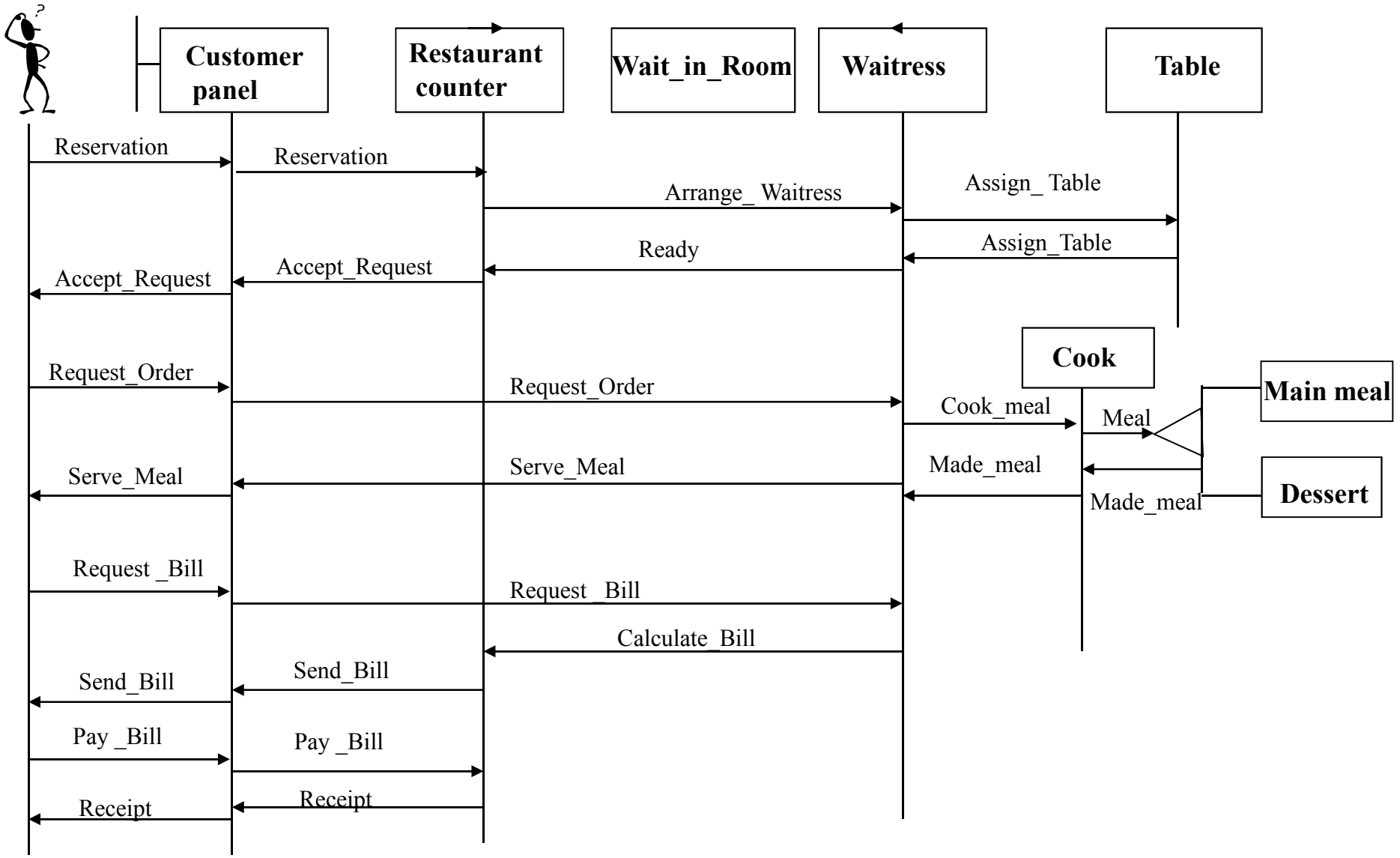
USE CASE SCENARIO : normal service



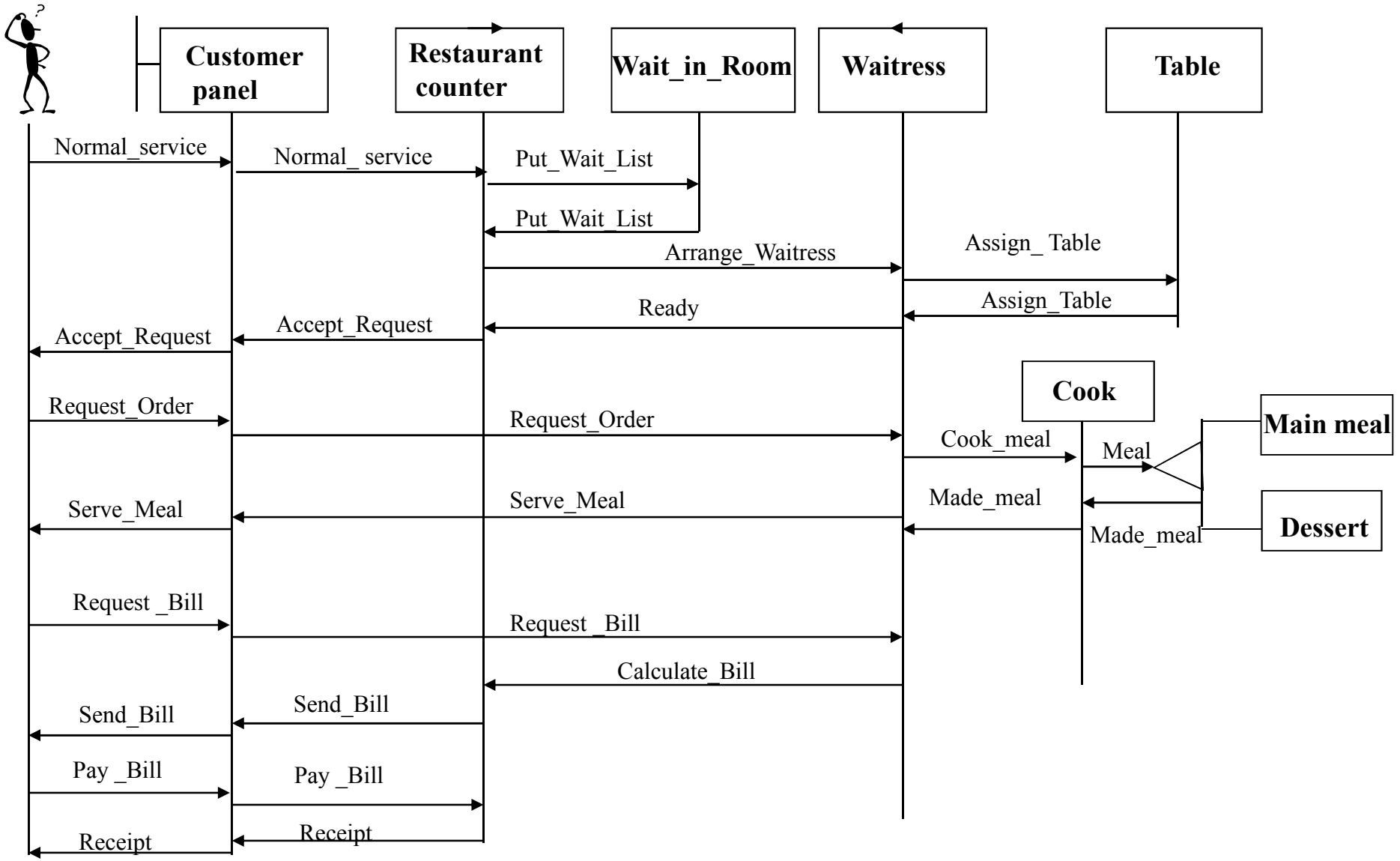
USE CASE SCENARIO : carry out



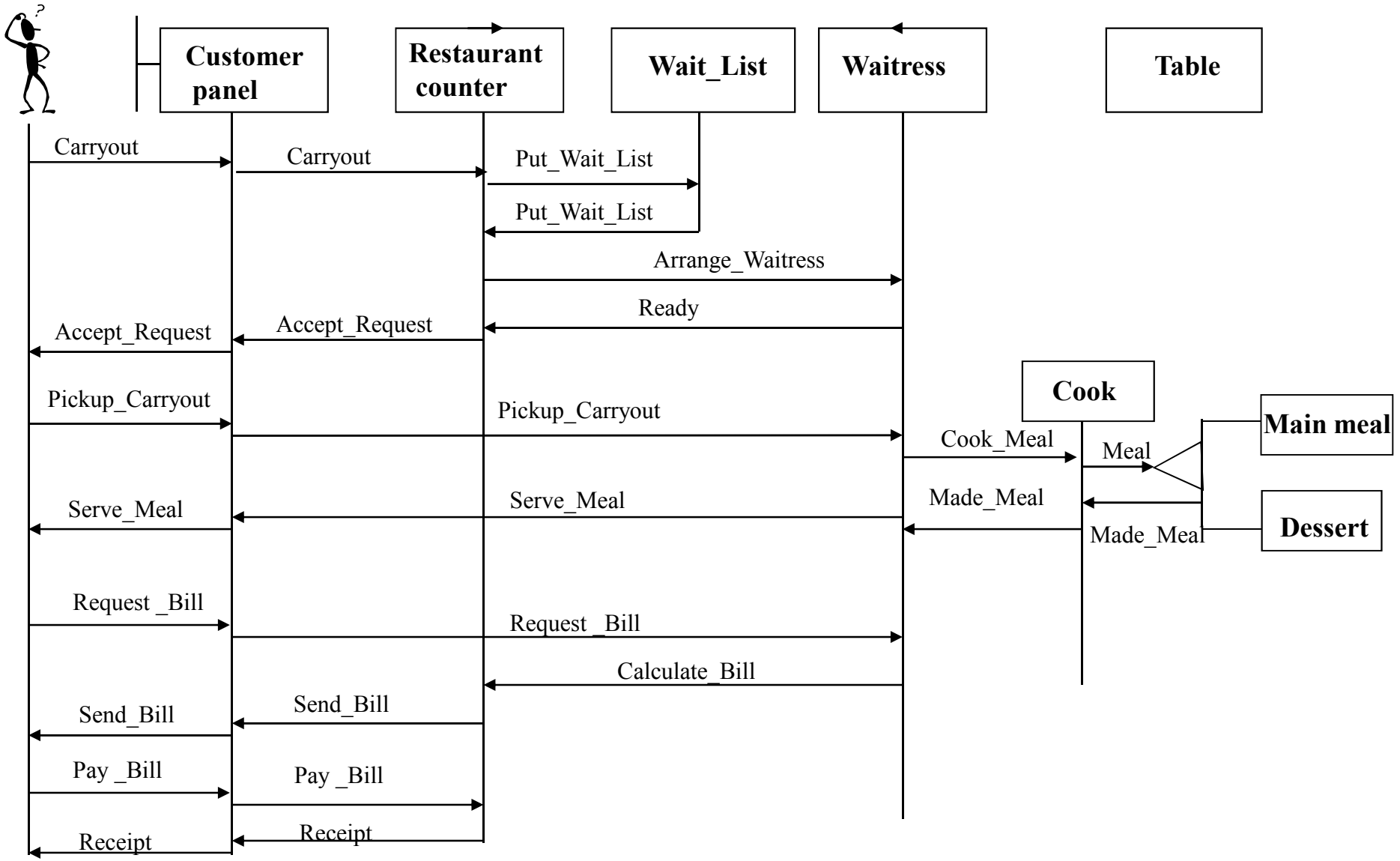
3. Message Sequence Diagram-Reservation



3. Message Sequence Diagram-Normal Service



3. Message Sequence Diagram-Carry out



Lecture 3
설계 사례

상세 설계 사례(Microwave)

-Class Diagram

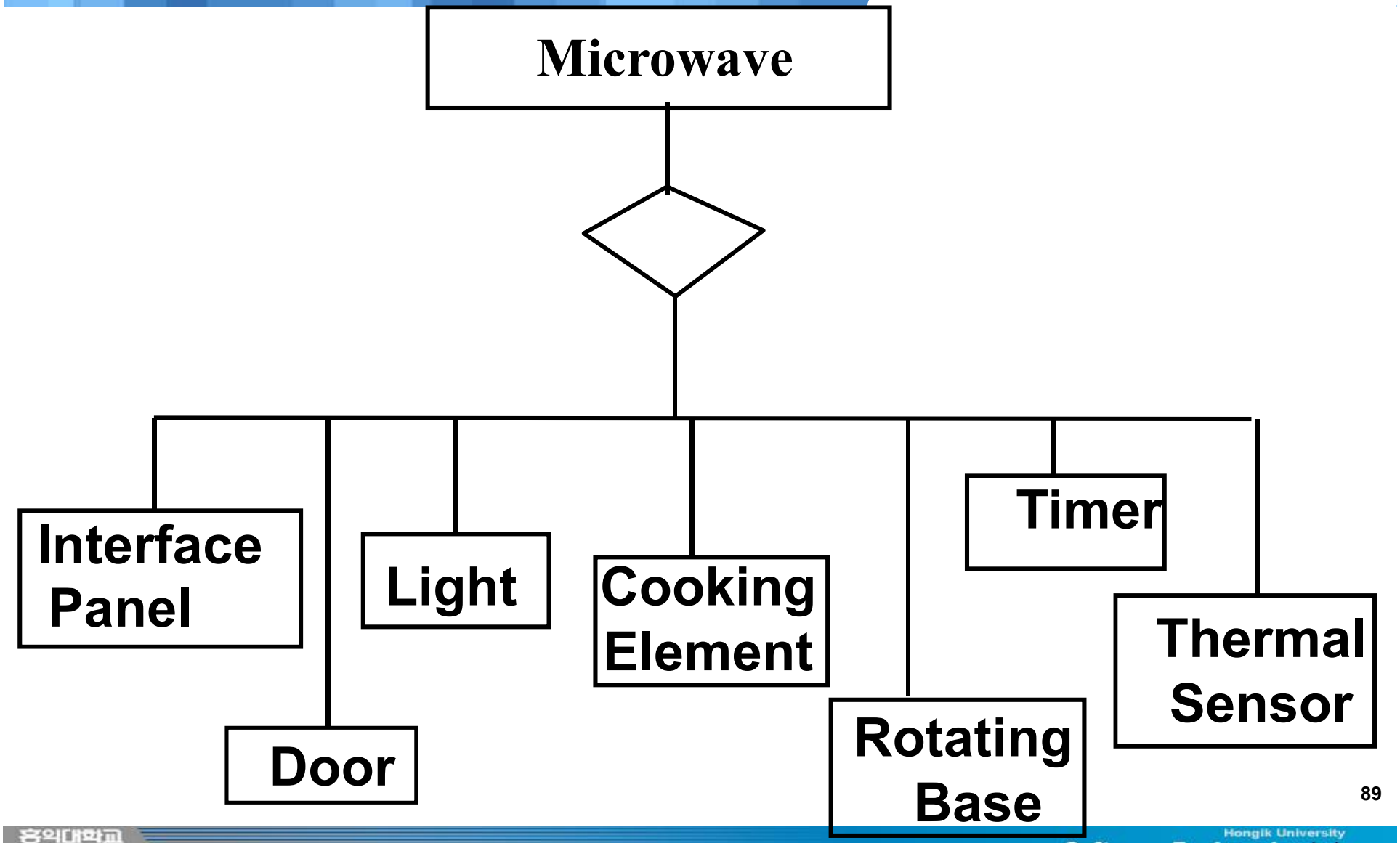
-State Diagram

-State Table

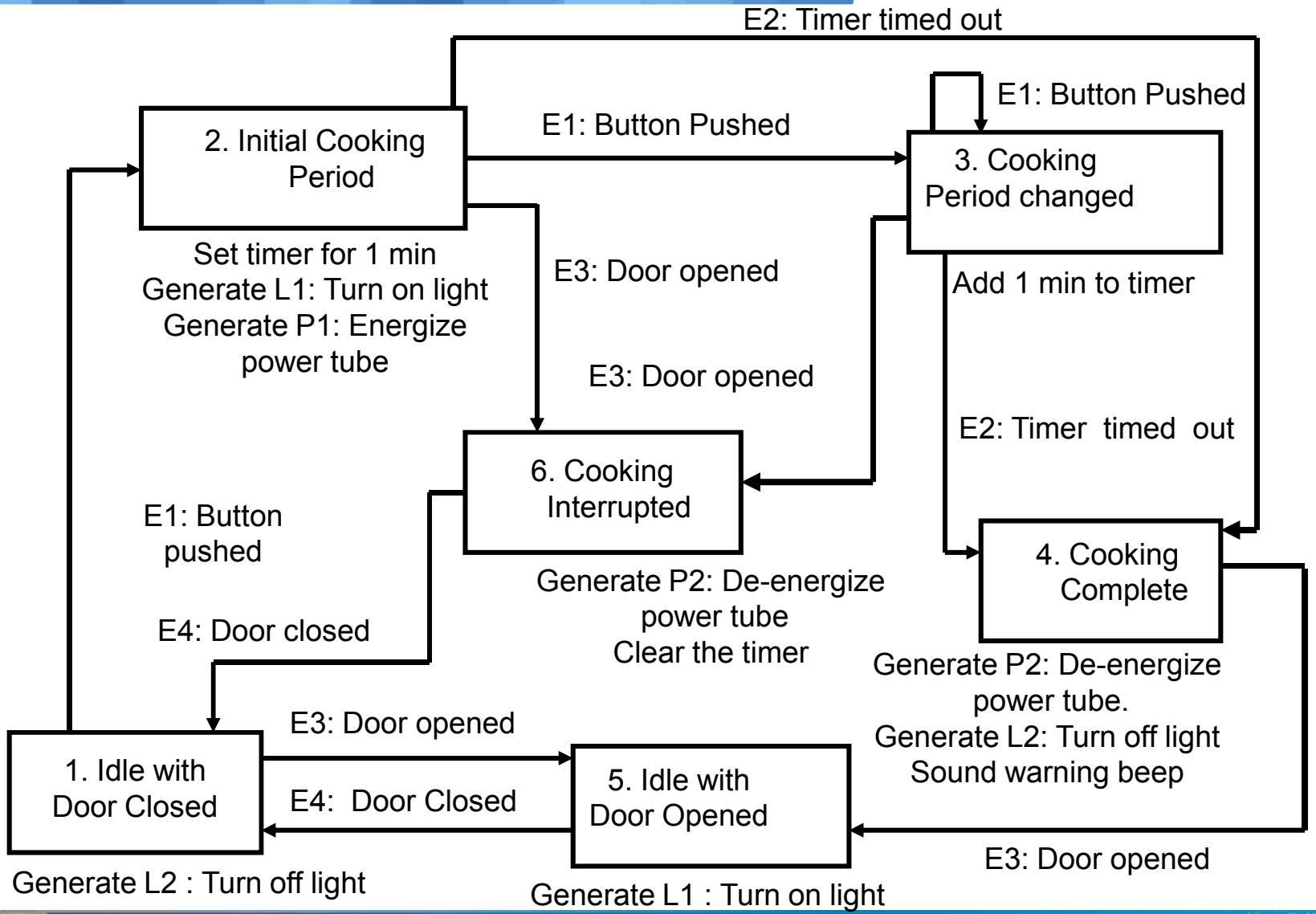
-State & Code Mapping

-Optimization

1. Class Diagram



2. State Diagram



3. State Table

	E1: Button Pushed	E2: Timer Timed out	E3: Door Opened	E4 : Door Closed
1. Idle with door closed	2	Can't happen [1]	5	Can't happen [3]
2. Initial Cooking period	3	4	6	Can't happen [3]
3. cooking period extended	3	4	6	Can't happen [3]
4. Cooking complete	event ignored	Can't happen [1]	5	Can't happen [3]
5. Idle with door open	event ignored	Can't happen [1]	Can't happen [2]	1
6. Cooking Interrupted	event ignored	Can't happen [1]	Can't happen [2]	1

Notes: [1] Timer is not running, [2] Door is already open,
[3] Door is already closed

각 상태에 대한 설명

1

IDLE DOOR CLOSED	
DOOR	CLOSED
LIGHT	OFF
COOKING ELEMENT	NOENERGY
ROTATING BASE	IMMOBILE
TIMER	STOPPED
THERMAL SENSOR	TEMP_OK
INTERFACE PANEL	READY

5

IDLE DOOR OPEN	
DOOR	OPEN
LIGHT	ON
COOKING ELEMENT	NOENERGY
ROTATING BASE	IMMOBILE
TIMER	STOPPED
THERMAL SENSOR	TEMP_OK
INTERFACE PANEL	IDLE

2

INITIAL COOKING	
DOOR	CLOSED
LIGHT	ON
COOKING ELEMENT	ENERGIZED
ROTATING BASE	ROTATING
TIMER	INITIAL_RUN
THERMAL SENSOR	TEMP_OK
INTERFACE PANEL	COOKING

3

EXTENDED COOKING	
DOOR	CLOSED
LIGHT	ON
COOKING ELEMENT	ENERGIZED
ROTATING BASE	ROTATING
TIMER	EXTD_RUN
THERMAL SENSOR	TEMP_OK
INTERFACE PANEL	COOKING

6

COOKING INTERRUPT	
DOOR	OPENED
LIGHT	ON
COOKING ELEMENT	NOENERGY
ROTATING BASE	IMMOBILE
TIMER	STOPPED
THERMAL SENSOR	TEMP_OK
INTERFACE PANEL	COOKED

4

COOKED	
DOOR	CLOSED
LIGHT	OFF
COOKING ELEMENT	NOENERGY
ROTATING BASE	IMMOBILE
TIMER	EXPIRED
THERMAL SENSOR	TEMP_OK
INTERFACE PANEL	COOKED

4. State & Code Mapping

STATE	ACTIONS
State 1	- Turn off light
State 2	- Set timer for 1 min. - Turn on light - Energize power tube
State 3	- Add 1 min. to timer
State 4	- De-energize power tube - Turn off light - Sound warning beep
State 5	- Turn on light
State 6	- De-energize power tube - Clear the timer

Sample Code

```
CASE
        * State 1 :   Event        Action/Code
        * State 2 :   Event        Action/Code
                      :
                      :
        * State 6 :   Event:  V3 (State 2 or State 3)
                      Action: De-energize power tube
                              Clear the timer

        * Can't happen : Error Code
        * Event / Ignore : No-Op
END CASE
```

5. Optimization of State Table

	V1: Button Pushed	V2: Timer Timed out	V3: Door Opened	V4 : Door Closed
1. Idle with door closed	2	Can't happen [1]	5	Can't happen [3]
2. Initial Cooking period	3	4	6	Can't happen [3]
3. cooking period extended	3	4	6	Can't happen [3]
4. Cooking complete	event ignored	Can't happen [1]	5	Can't happen [3]
5. Idle with door open	event ignored	Can't happen [1]	Can't happen [2]	1
6. Cooking Interrupted	event ignored	Can't happen [1]	Can't happen [2]	1

Notes: [1] Timer is not running, [2] Door is already open,
[3] Door is already closed

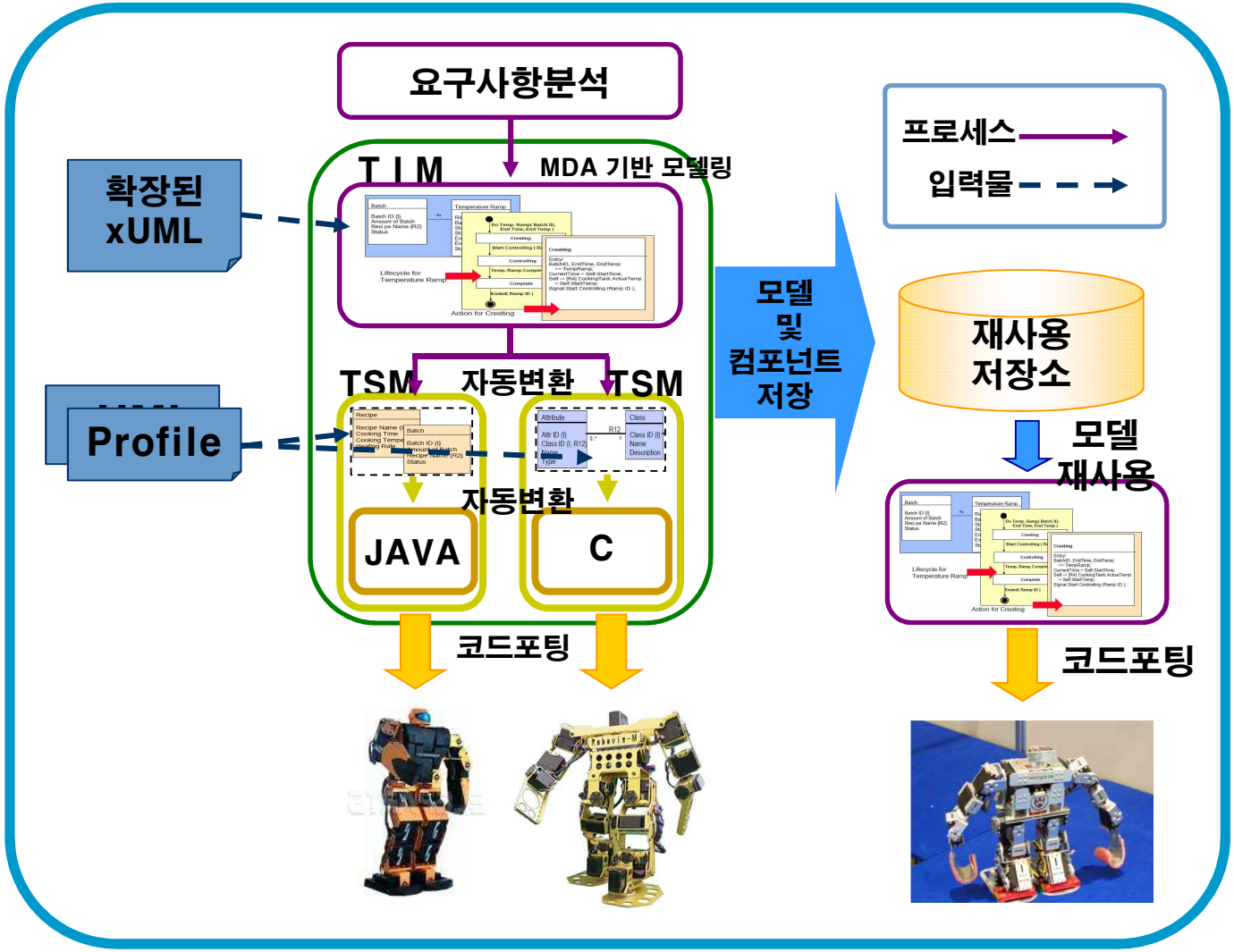
Lecture 4
실습

5일차 금요일

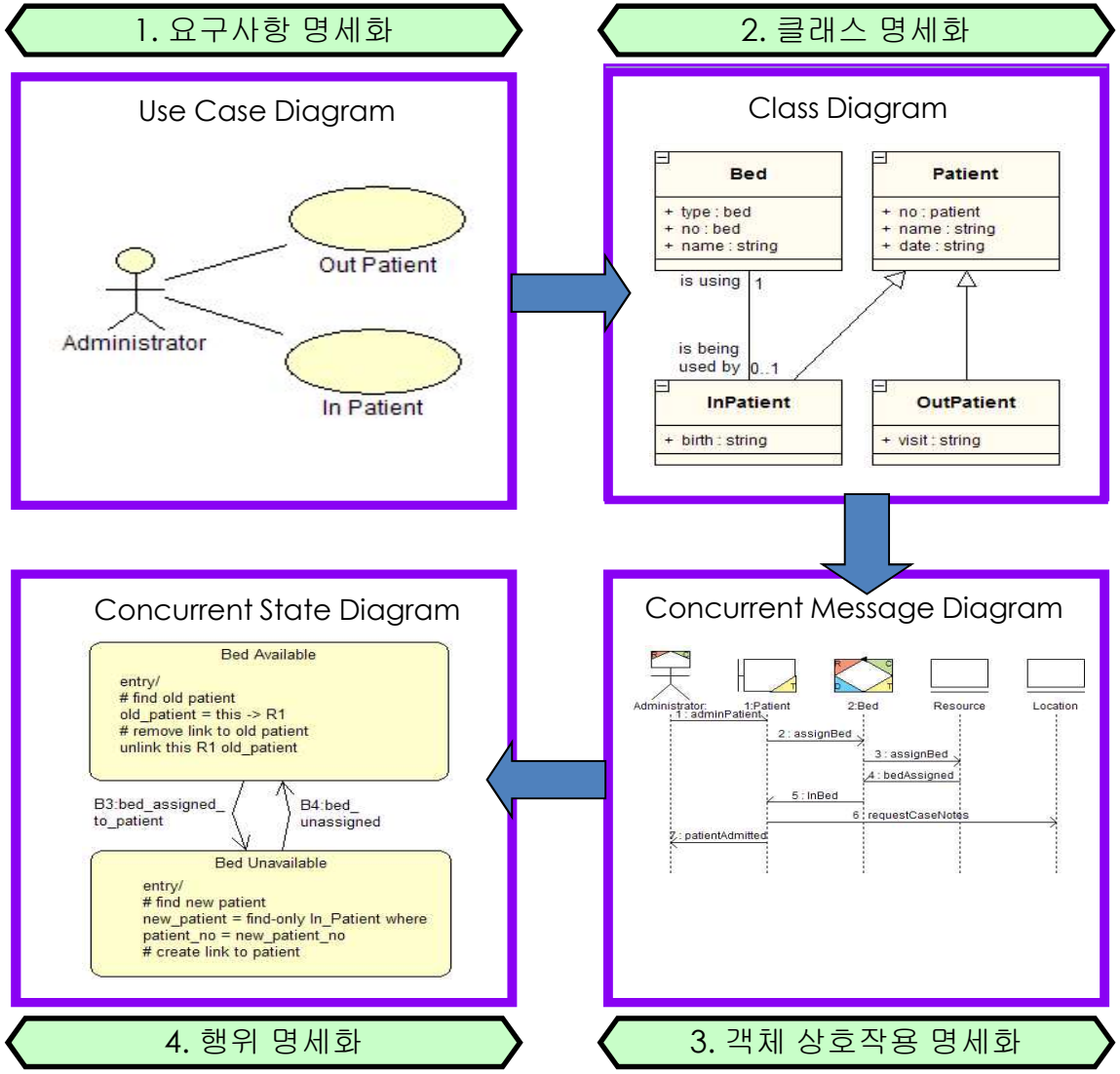
Lecture 5
모델링과 코드 관련성 소개

모델간의 변환

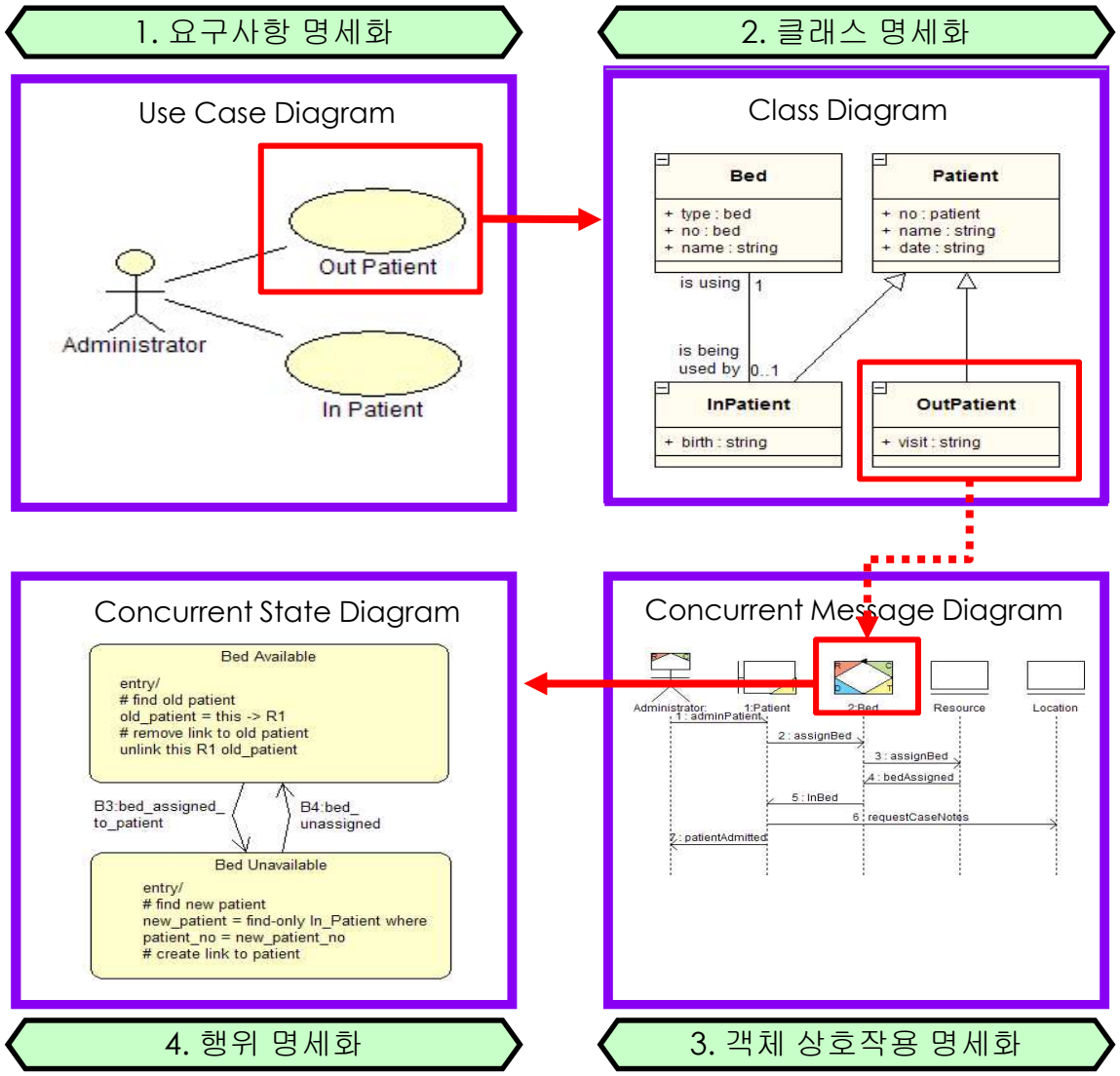
Model to Model Transformation



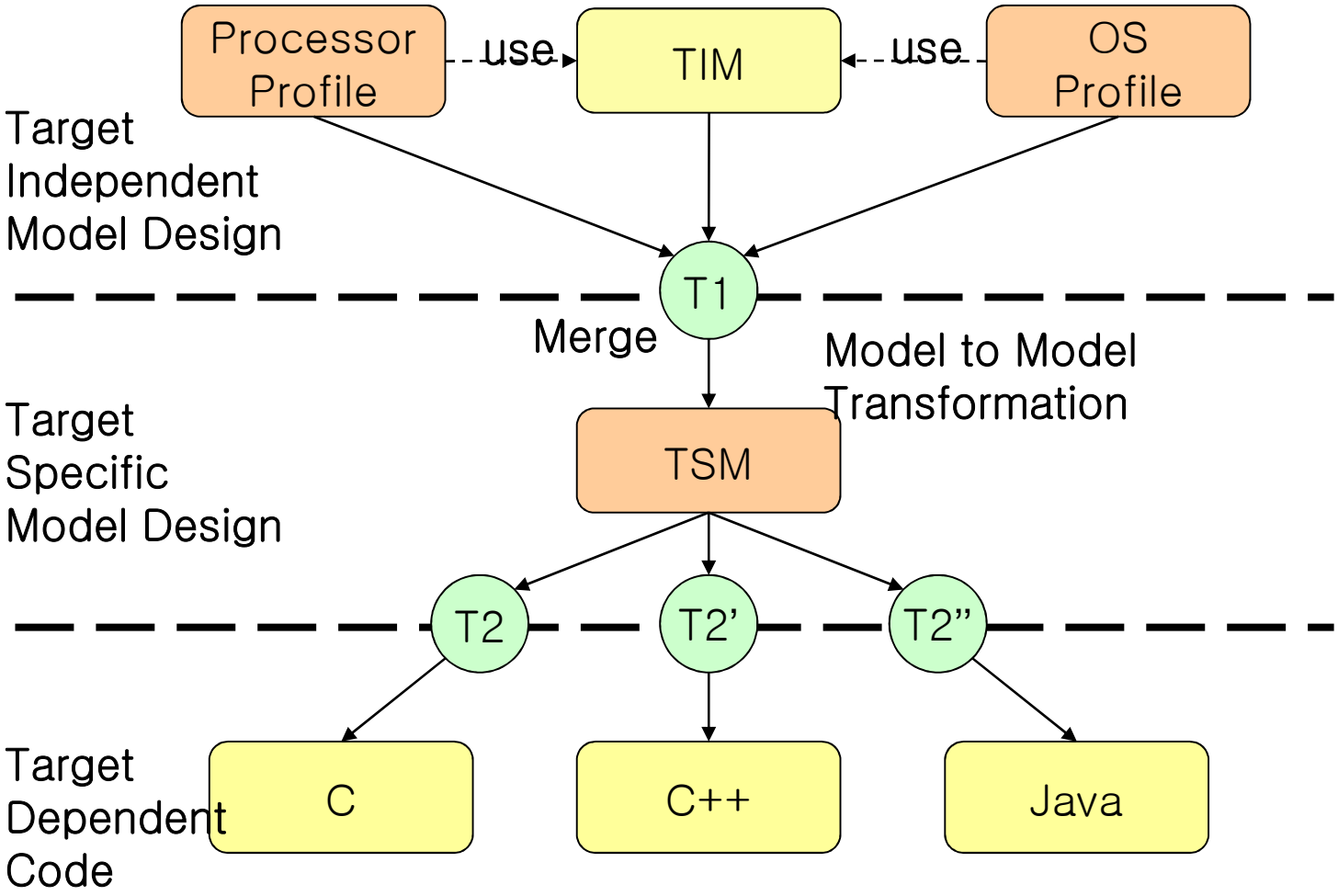
Basic UML Diagram Set



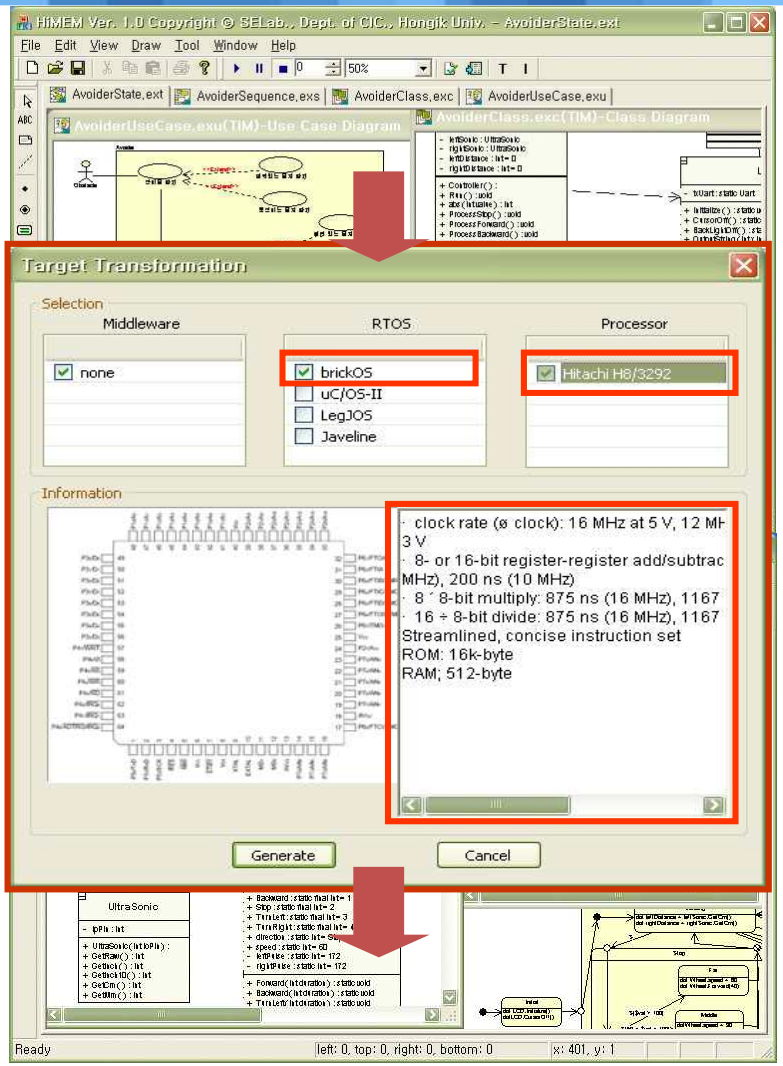
Relationship between UML Diagrams



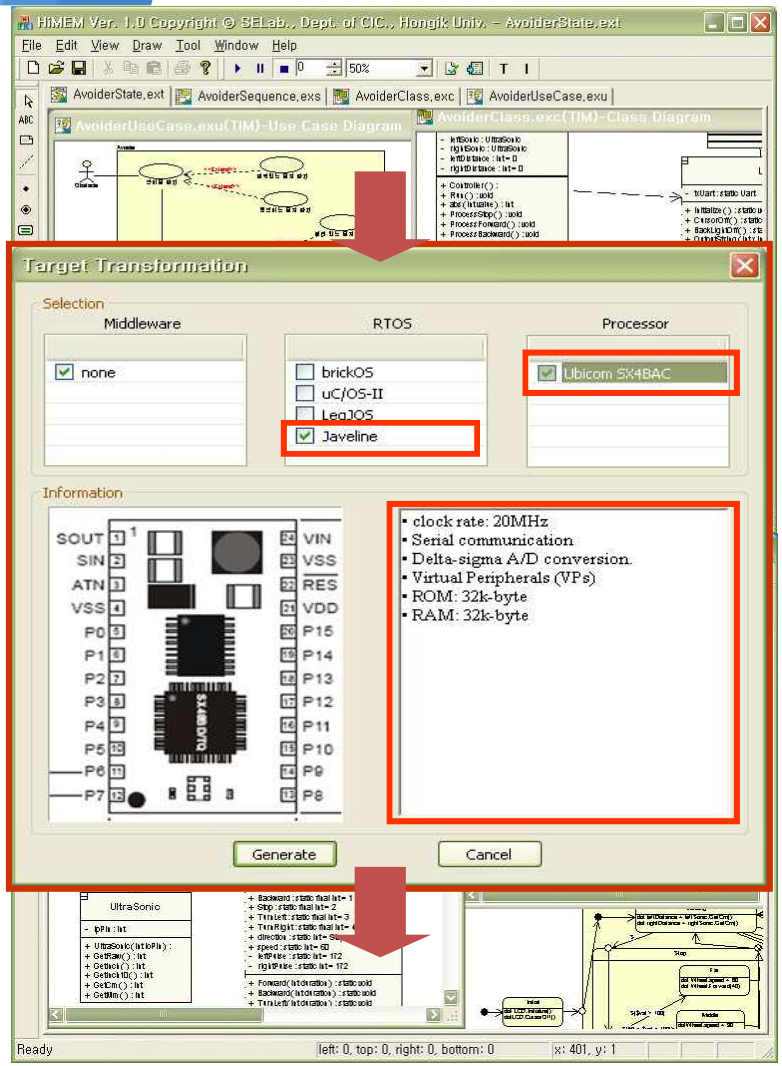
모델 변형 관계도


Processor Profile
use
TIM
use
OS Profile
Target Independent Model Design
T1
Merge
Model to Model Transformation
Target Specific Model Design
TSM
T2
T2'
T2''
Target Dependent Code
C
C++
Java

자동도구 상에서 TIM → TSM



(a) SUGV 1



(b) SUGV2

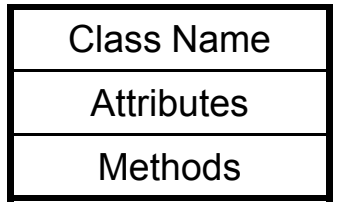
Lecture 5
모델링과 코드 관련성 소개

코드와의 관계

클래스

❖ 클래스

- 사각형 안에 클래스 이름, 속성, 메소드 포함
- 클래스이름
  - 일반 클래스, 정적클래스
  - *추상 클래스*
- 속성
  - [+/-/#]속성이름[멀티플리시티][:타입][=초기값][{속성}]
  - 접근제한자 : + : public, - : private, # : protected
  - Example
    - -age:int = 10



클래스

❖ 클래스

– Method

- 형식
- [+/-#]메소드이름[([파라미터:타입],…)][:반환값 타입] [{속성}]
- 예> +sleep(hours:int):void
- *calcTax()*
- 페이지 68

❖ 객체

– 객체이름:클래스이름

obj:OrderClass

관계 표현

❖ 관계표현

- 클래스들간의 관계표현
- 관계 종류(Relationship)
  - Association (연관 관계)
  - Aggregation (전체와 부분 집합관계)
  - Inheritance (상속 관계)
  - Dependency (의존 관계)

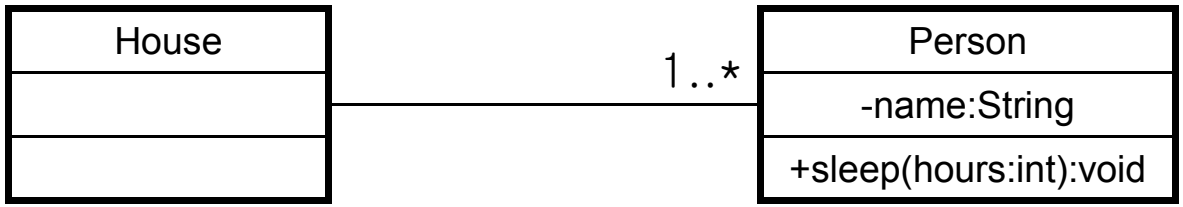
관계 표현

❖ Association 관계

- 한 객체가 다른 객체와 연결되어 있음
- 연관관계는 방향성(navigability)과 멀티플리시티(multiplicity) 를 가진다.
- Navigability
  - A → B : A 클래스만이 B 클래스 존재 인식
  - A – B : 두 클래스 모두 서로의 존재 인식
- Multiplicity
  - 서로 연관된 객체의 개수의미
  - 1, 1..*, * or 0..*, 0..1, default : 1

관계 표현

❖ 연관 관계



Implementation

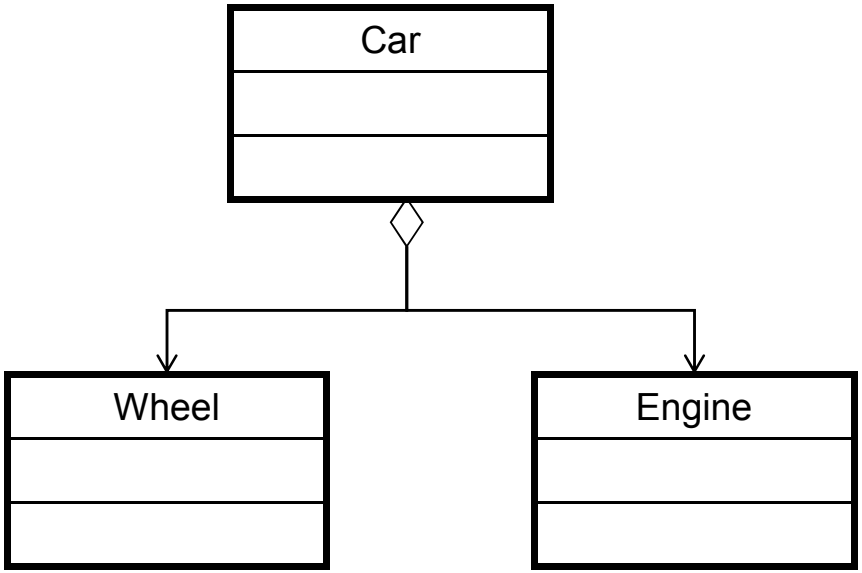
```
public class House{
   Person[] persons;
   ...
}
```

```
public class Person{
  House house;
  private String name;
  public sleep(int hour){
    //...
  }
  ...
}
```

관계 표현

❖ 집합 관계 (Aggregation)

- 연관 관계의 특별한 형태
- 전체와 부분의 개념을 포함하는 관계
- 구현 코드는 동일



Implementation

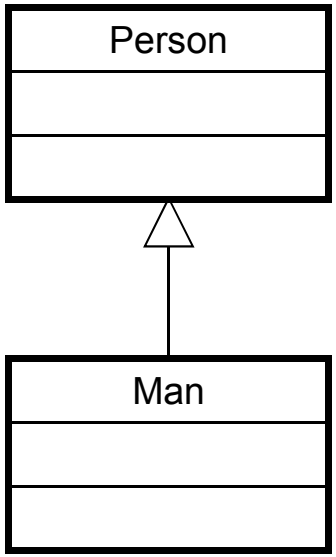
```
public class Car{

   private Wheel wheel;
   private Engine engine;
...
}
```

관계 표현

❖ 상속관계

- 클래스 상속은 두 클래스사이의 IS-A 관계 표현
- 부모 속성 포함관계



Implementation

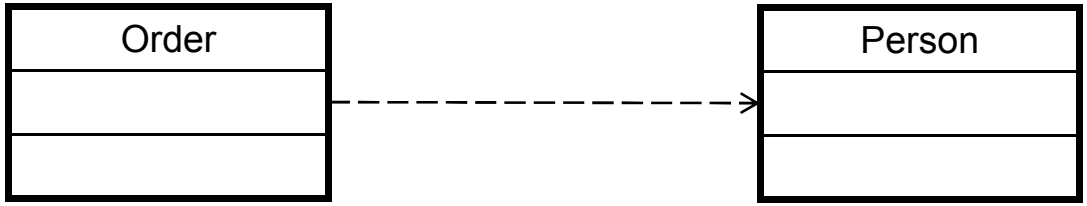
```
Public class Man extends Person{
  //...

 }
```

관계 표현

❖ **의존 관계**

- **한 클래스가 변경될 때 이것을 사용하는 다른 클래스에 영향이 미치는 관계 표현**
- **점선 관계로 표현**



Order 클래스가 Person 클래스를 사용한다.

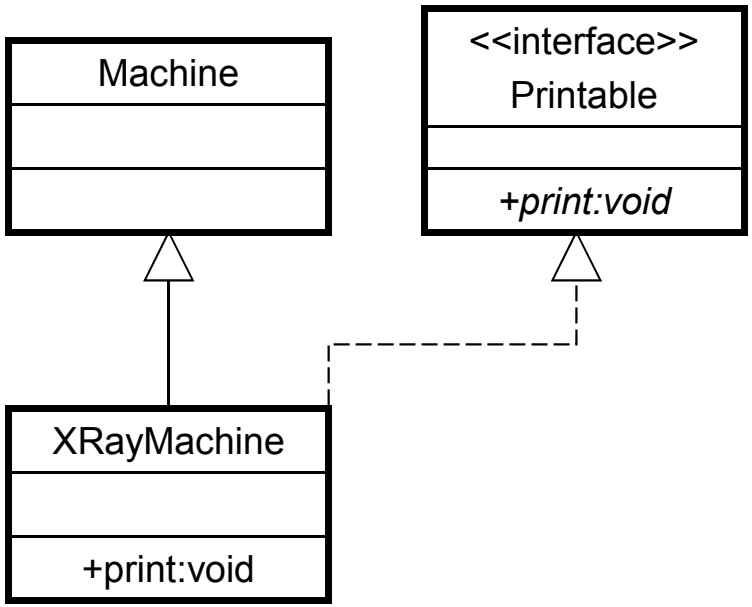
의존관계 존재 여부

1. 한 클래스의 메소드가 다른 클래스의 객체를 인자로 받아 사용(가장일반적)
2. 한 클래스의 메소드 내부에서 객체 생성
3. 다른 클래스의 메소드가 다른 클래스의 객체 반환

인터페이스 구현

❖ 구체적 표기 문법

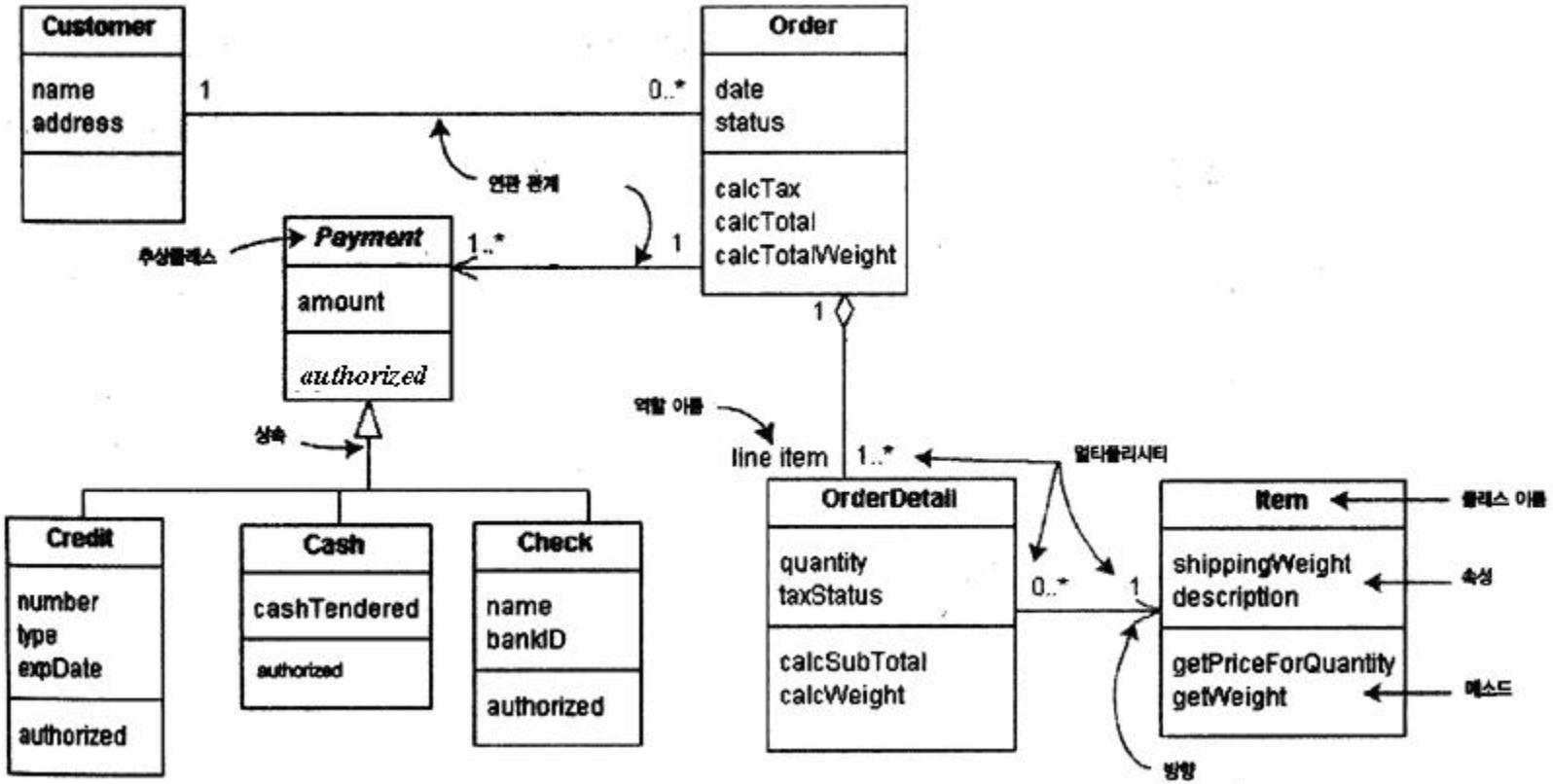
- 클래스의 형태에 <<스테레오타입>> 추가
- 인터페이스 구현 클래스 관계 점선




```
Public class XRayMachine
  extends Machine
  implements Printable {
  public void print(){
  //...
  }
}
```

클래스 다이어그램

❖ 클래스 다이어그램 예



Lecture 5
모델링과 코드 관련성 소개

모델과 코드와의 매핑

코드 생성 방법

Class Name: string	
Package List: List	
Parent List : List	
Interface List : List	
Association List: List	
Association List	Attribute
	SetFunction
Attribute List: List	
Function List: List	
Function List	Head
	Body

메타모델

+

Java

```
import [Package List]
class [Class Name]
extends [Parent List]
implements [Interface List]
{
    //association
    protected
    [Association List(Attribute)]
    public
    [Assocation List(SetFunction)]
    //attribute
    [Attribute List]
    //Function
    [Function List(Head)]
    {
            [Function List(Body)]
    }
};
```

C++

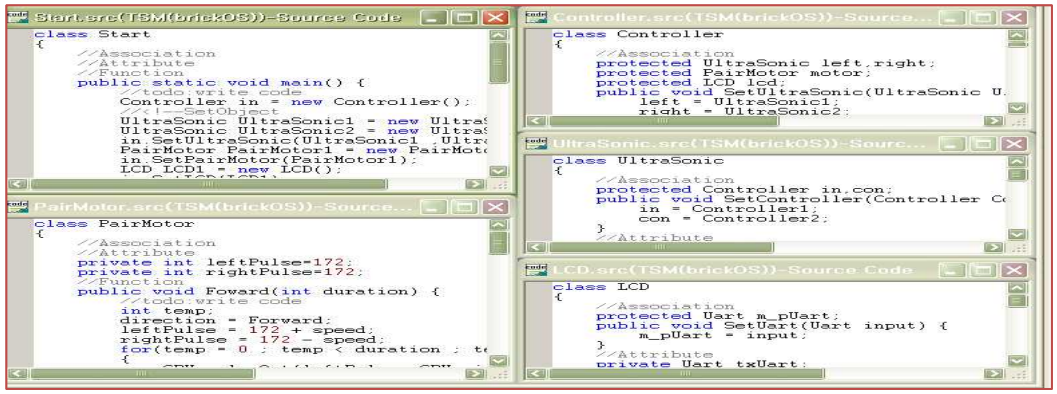
```
#include [Package List]
class [Class Name]
: [Parent List], [Interface List]
{
    //association
    protected :
      [Association List(Attribute)]
    public :
      [Assocation List(SetFunction)]
    //attribute
    [Attribute List]
    //Function
    [Function List(Head)]
    {
            [Function List(Body)]
    }
};
```

C

```
#include [Package List]
//attribute
[Attribute List]
//Function
[Function List(Head)]

[Function List(Head)]
{
    [Function List(Body)]
}
```

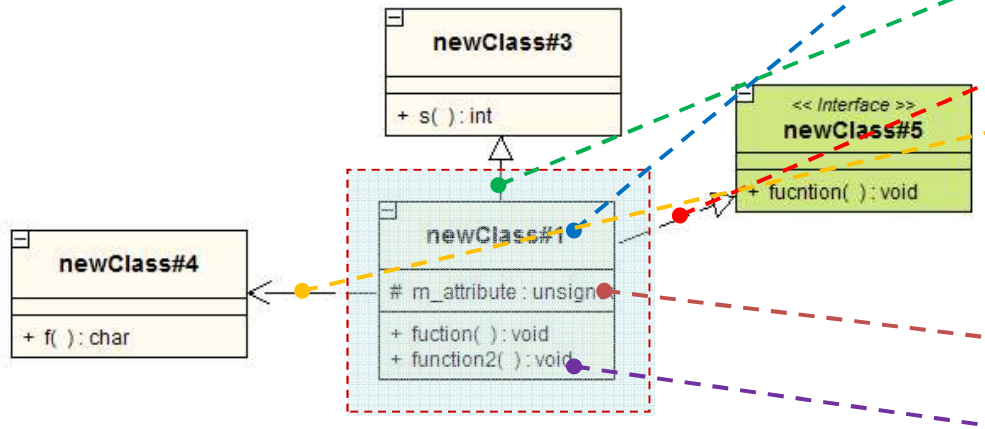
코드템플릿



생성된 코드

코드 생성(Class Diagram 매핑)

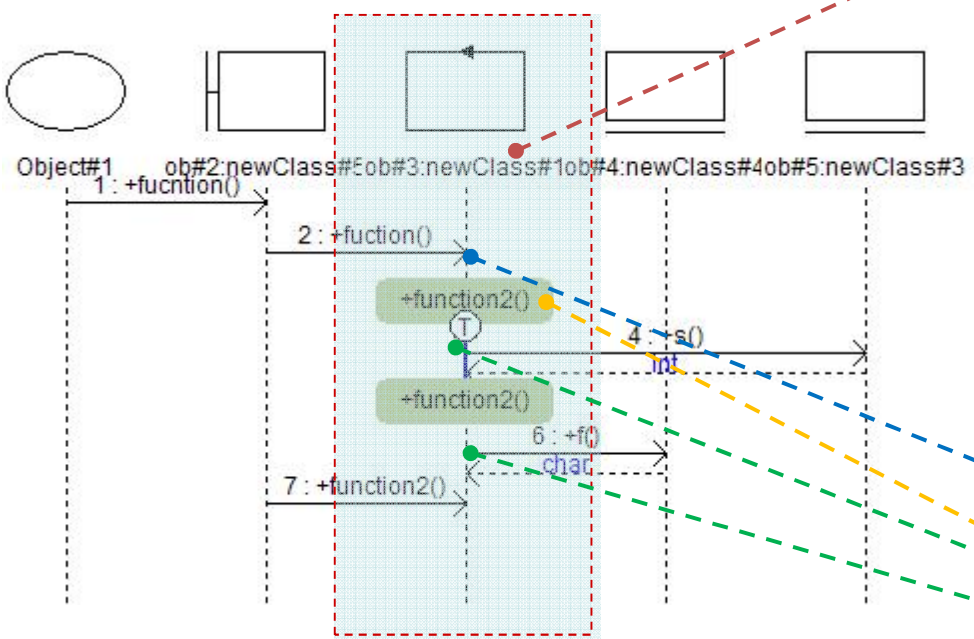
❖ **Class Diagram**



Class Name: string	
Package List: List	
Parent List : List	
Interface List : List	
Association List: List	
Association List	Attribute
	SetFunction
Attribute List: List	
Function List: List	
Function List	Head
	Body

코드 생성(Concurrent Message Diagram 매핑)

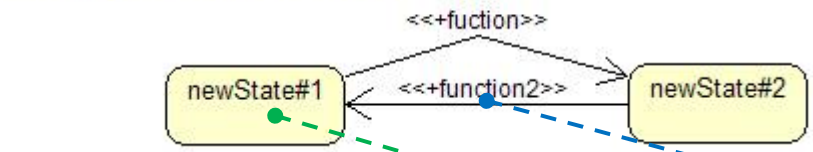
❖ **Concurrent Message Diagram**



Class Name: string	
Package List: List	
Parent List : List	
Interface List : List	
Association List: List	
Association List	Attribute
	SetFunction
Attribute List: List	
Function List: List	
Function List	Head
	Body

코드 생성(Concurrent State Diagram 매핑)

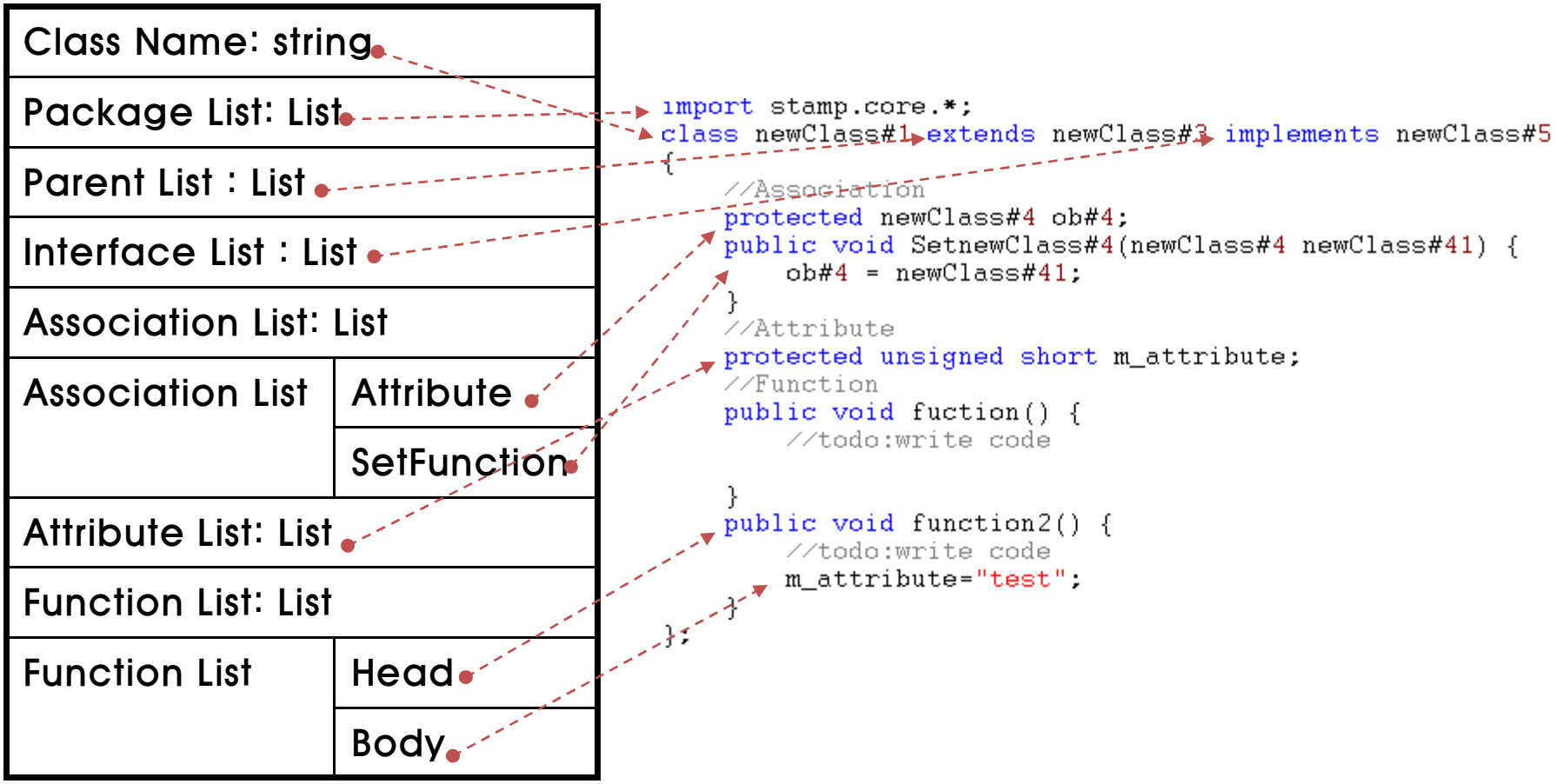
❖ **Concurrent State Diagram**



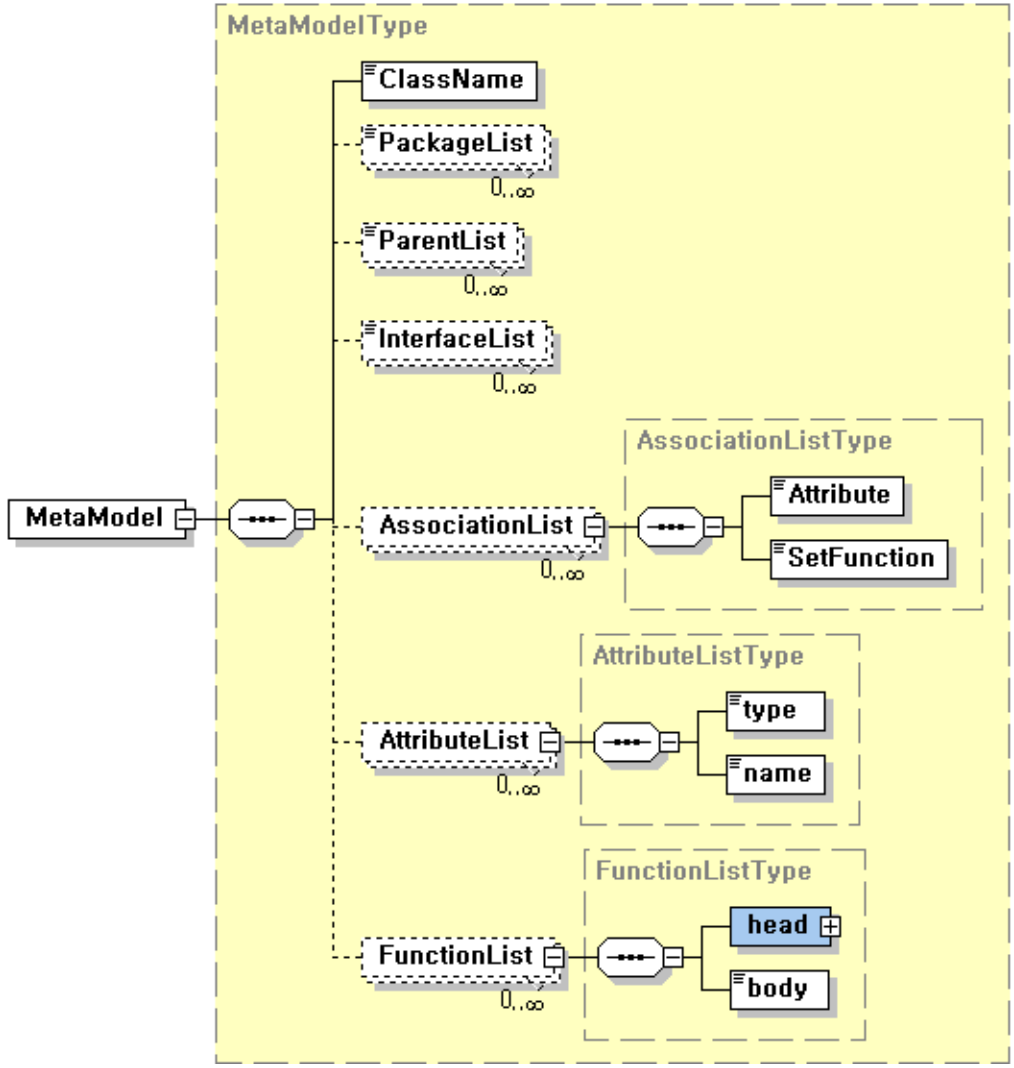
Class Name: string	
Package List: List	
Parent List : List	
Interface List : List	
Association List: List	
Association List	Attribute
	SetFunction
Attribute List: List	
Function List: List	
Function List	Head
	Body

코드 생성(메타모델과 코드템플릿 매핑)

❖ **Source Code**



코드 생성(각각의 코드생성을 위한 메타모델)




```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="headType">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="parameter" type="xsd:string" minOccurs
="0" maxOccurs="unbounded"/>
      <xsd:element name="return" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="FunctionListType">
    <xsd:sequence>
      <xsd:element name="head" type="headType"/>
      <xsd:element name="body" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="AttributeListType">
    <xsd:sequence>
      <xsd:element name="type" type="xsd:string"/>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="AssociationListType">
    <xsd:sequence>
      <xsd:element name="Attribute" type="xsd:string"/>
      <xsd:element name="SetFunction" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="MetaModelType">
    <xsd:sequence>
      <xsd:element name="ClassName" type="xsd:string"/>
      <xsd:element name="PackageList" type="xsd:string"
minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="ParentList" type="xsd:string" minOccurs="
0" maxOccurs="unbounded"/>
      <xsd:element name="InterfaceList" type="xsd:string"
minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="AssociationList" type="
AssociationListType" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="AttributeList" type="AttributeListType"
minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="FunctionList" type="FunctionListType"
minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="MetaModel" type="MetaModelType"/>
</xsd:schema>
```

코드 생성(각각의 코드생성 템플릿)

Java

```
import [Package List]
class [Class Name]
extends [Parent List]
implements [Interface List]
{
    //association
    protected
    [Association List(Attribute)]
    public
    [Assocation List(SetFunction)]
    //attribute
    [Attribute List]
    //Function
    [Function List(Head)]
    {
            [Function List(Body)]
    }
};
```

C++

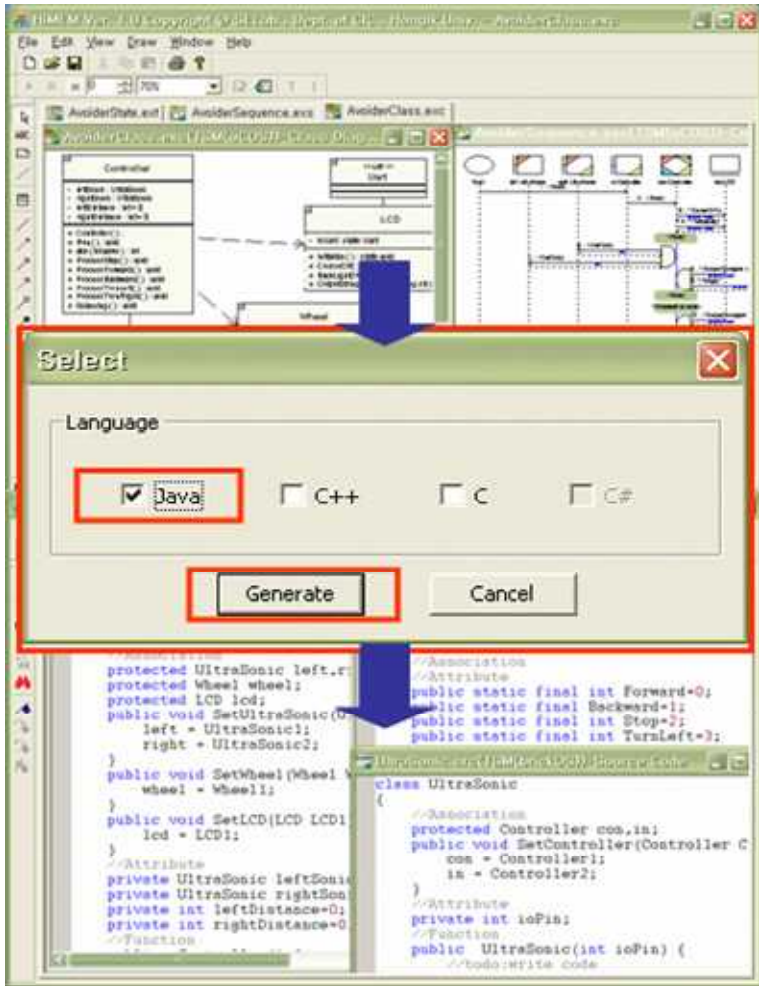
```
#include [Package List]
class [Class Name]
: [Parent List], [Interface List]
{
    //association
    protected :
        [Association List(Attribute)]
    public :
        [Assocation List(SetFunction)]
    //attribute
    [Attribute List]
    //Function
    [Function List(Head)]
    {
            [Function List(Body)]
    }
};
```

C

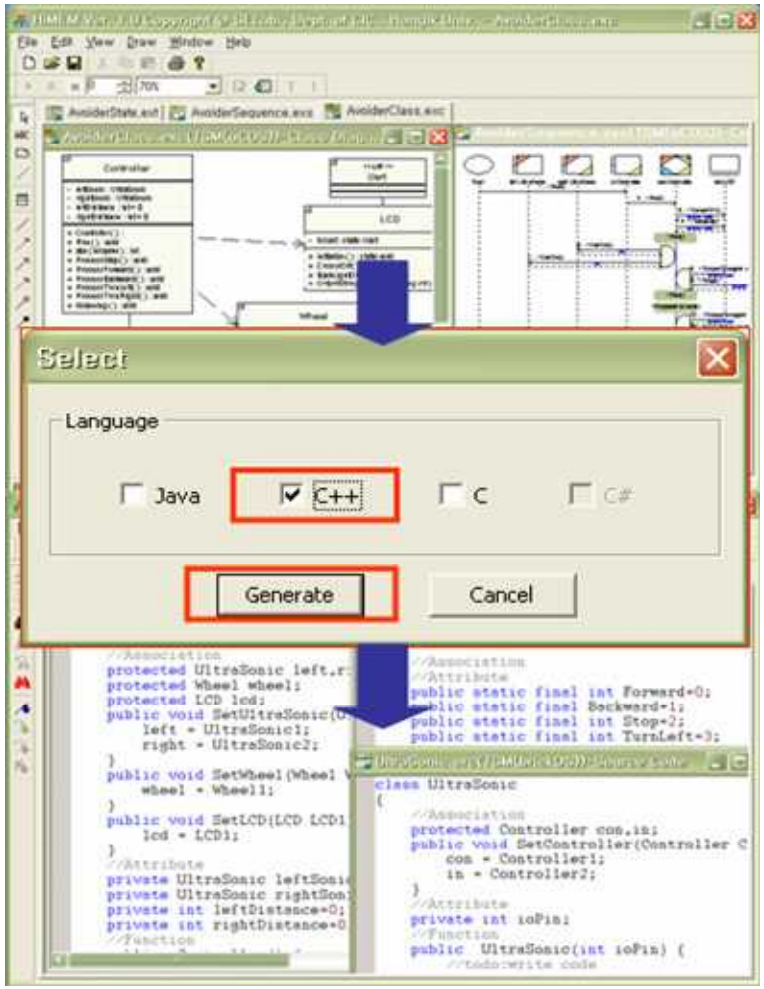
```
#include [Package List]
//attribute
[Attribute List]
//Function
[Function List(Head)]

[Function List(Head)]
{
    [Function List(Body)]
}
```

자동도구상에서 TSM → TDC



(a) SUGV 1



(b) SUGV2

Lecture 6
임베디드 UML 도구 소개

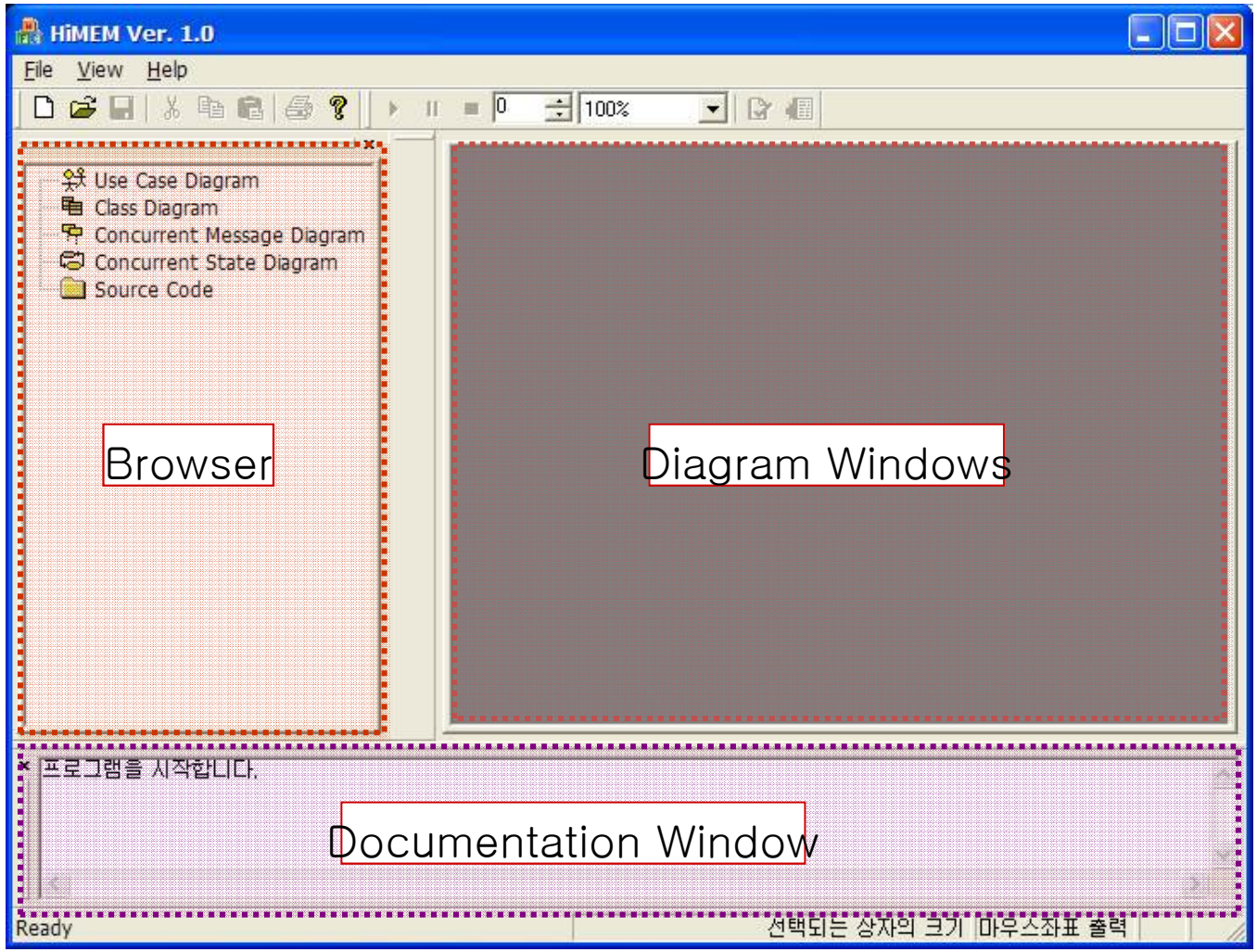
UML 및 도구 비교

도구의 개요

❖ UML(Unified Modeling Language)을 지원하는 소프트웨어 모델링

- Executable UML 기반
- 총 4가지의 다이어그램 제공
  - Use Case Diagram, Class Diagram, Concurrent Message Diagram, Concurrent State Diagram
- UML 프로파일 개념 및 코드 생성 지원
- 임베디드에 적응하는 UML 도구
  - 임베디드 환경에 적응할 수 있도록 설계
  - 임베디드 소프트웨어 개발 방법론, 프로젝트의 플랫폼, 언어 등에 적응할 수 있는 서비스를 제공
- MDA (Model Driven Architecture) 접근방법 지원
  - UML 1.4 표준 메타모델과 UML Profile 개념을 제공
  - 플랫폼에 독립적인 모델을 작성
  - 간단한 모델 작성만으로 사용자가 원하는 코드 생성

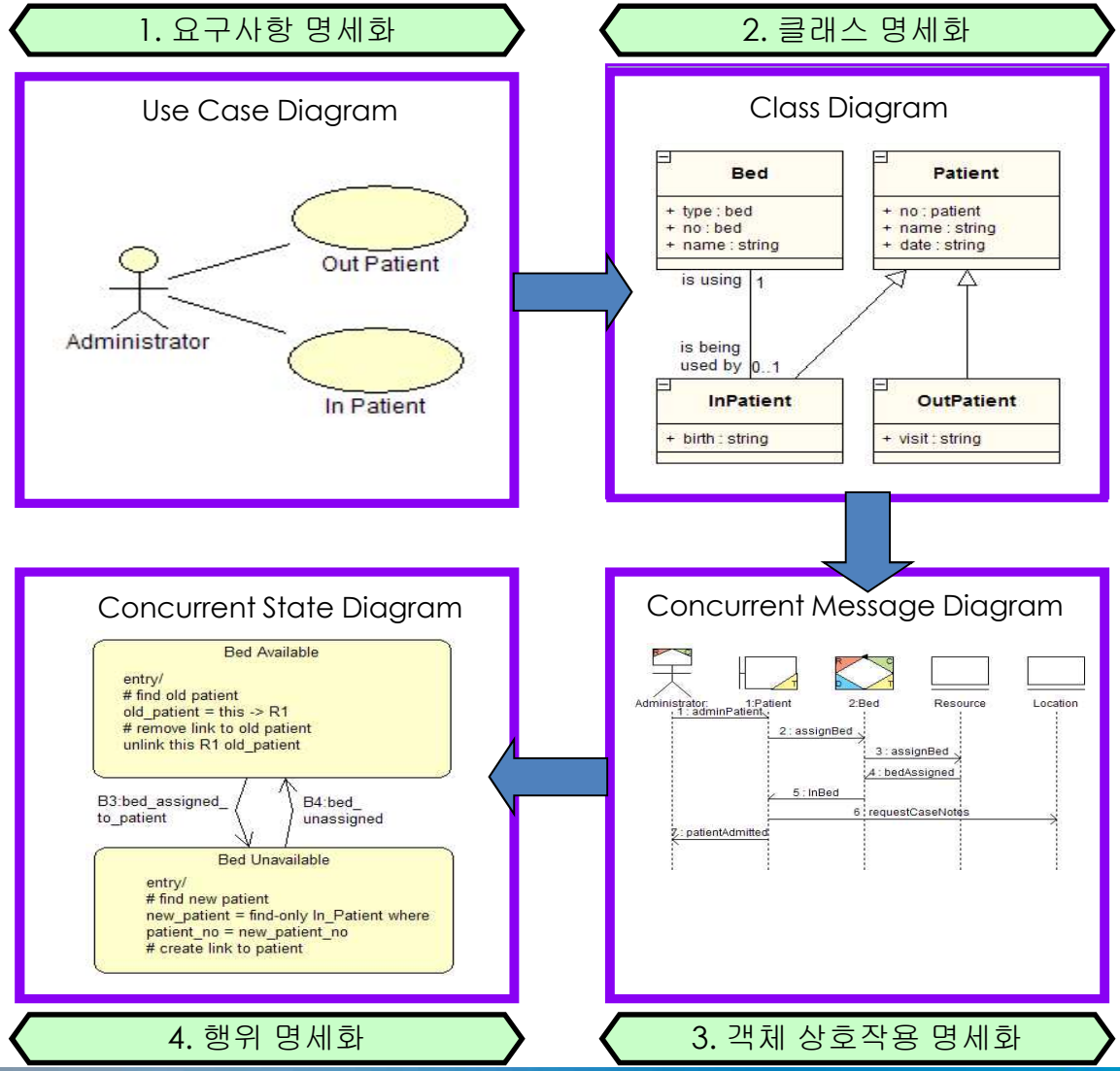
메인 윈도우


HiMEM Ver. 1.0
File View Help
100%
Use Case Diagram
Class Diagram
Concurrent Message Diagram
Concurrent State Diagram
Source Code
Browser
Diagram Windows
프로그램을 시작합니다.
Documentation Window
Ready
선택되는 상자의 크기 마우스좌표 출력

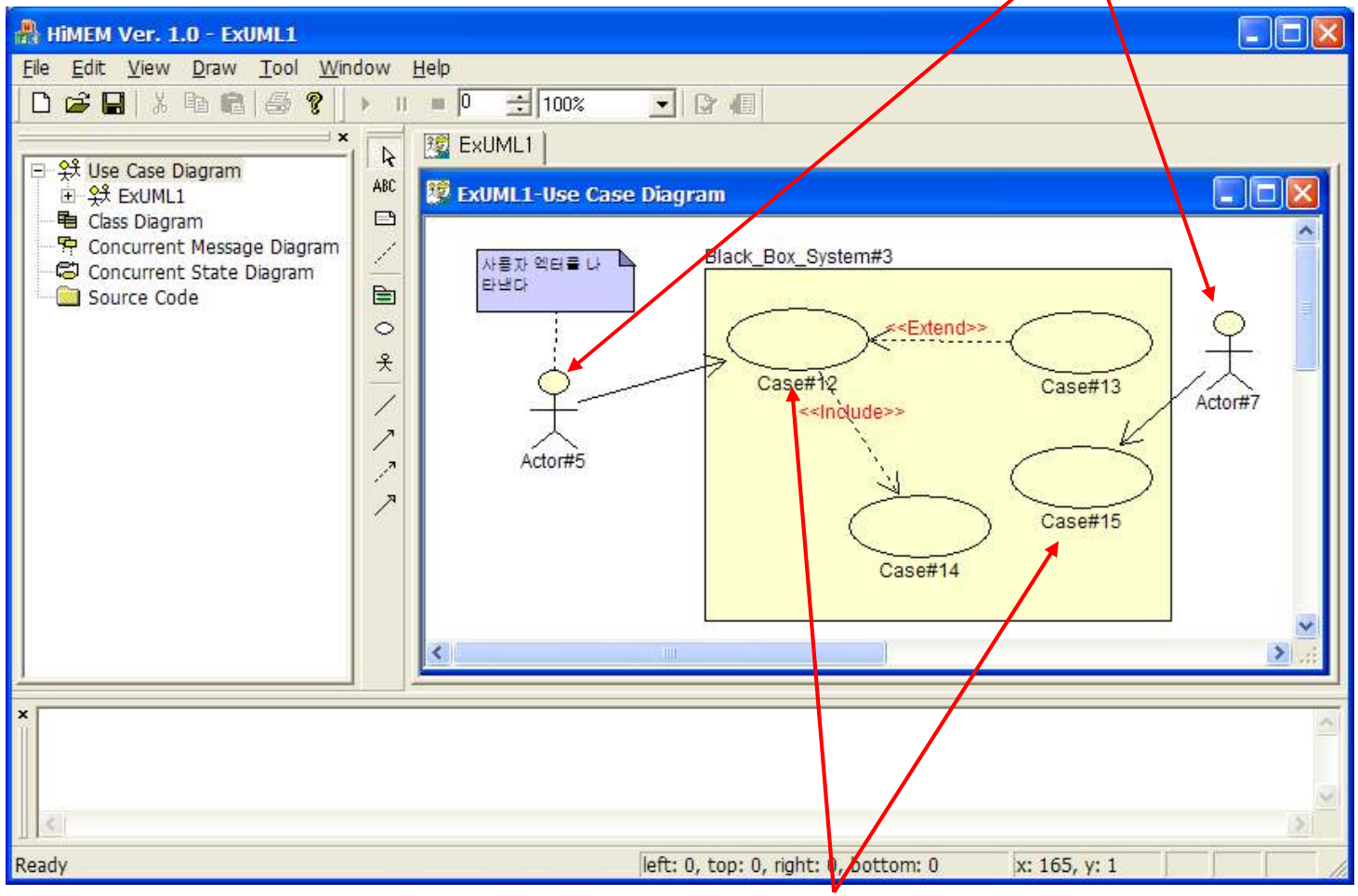
구성 다이어그램

- Use Case Diagram
  - 시스템과 사용자의 요구 분석 과정에 사용
  - Actor와 Use Case간의 상호 작용 표현
- Class Diagram
  - 시스템의 정적 구조 표현
  - 클래스들의 관계 표현
- Concurrent Message Diagram
  - 클래스로부터 생성된 객체들 간의 메시지 교환에 관한 시간순서 표현
- Concurrent State Diagram
  - 객체의 상태 및 상태 변화 표현

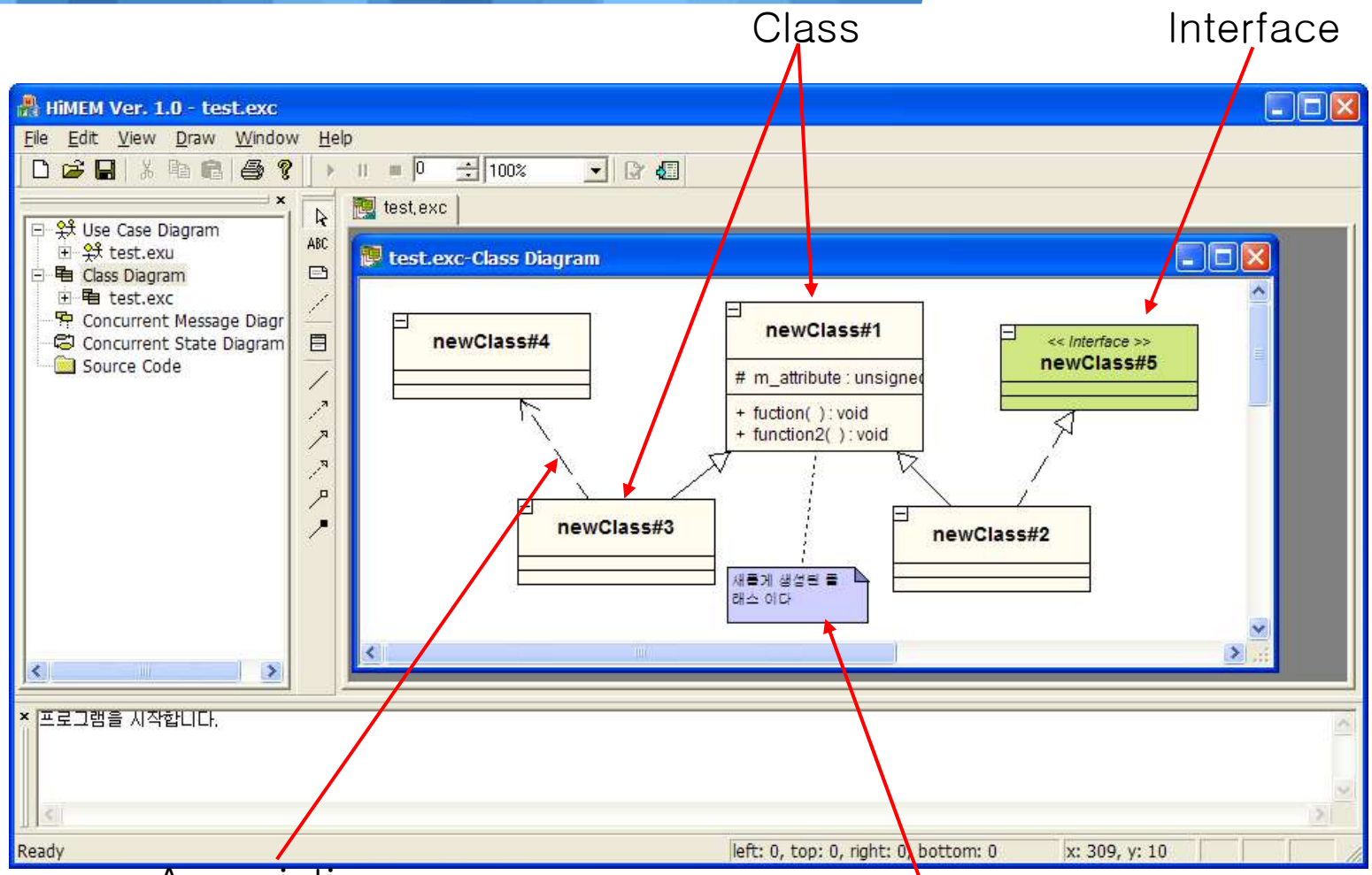
소프트웨어 모델링 절차


1. 요구사항 명세화
2. 클래스 명세화
Use Case Diagram
Out Patient
Administrator
In Patient
Class Diagram
Bed
+ type : bed
+ no : bed
+ name : string
Patient
+ no : patient
+ name : string
+ date : string
is using
1
is being used by
0..1
InPatient
+ birth : string
OutPatient
+ visit : string
Concurrent State Diagram
Bed Available
entry/
# find old patient
old_patient = this -> R1
# remove link to old patient
unlink this R1 old_patient
B3:bed_assigned_ to_patient
B4:bed_ unassigned
Bed Unavailable
entry/
# find new patient
new_patient = find-only In_Patient where patient_no = new_patient_no
# create link to patient
Concurrent Message Diagram
Administrator
1:Patient
2:Bed
Resource
Location
1 : adminPatient
2 : assignBed
3 : assignBed
4 : bedAssigned
5 : InBed
6 : requestCaseNotes
7 : patientAdmitted
4. 행위 명세화
3. 객체 상호작용 명세화

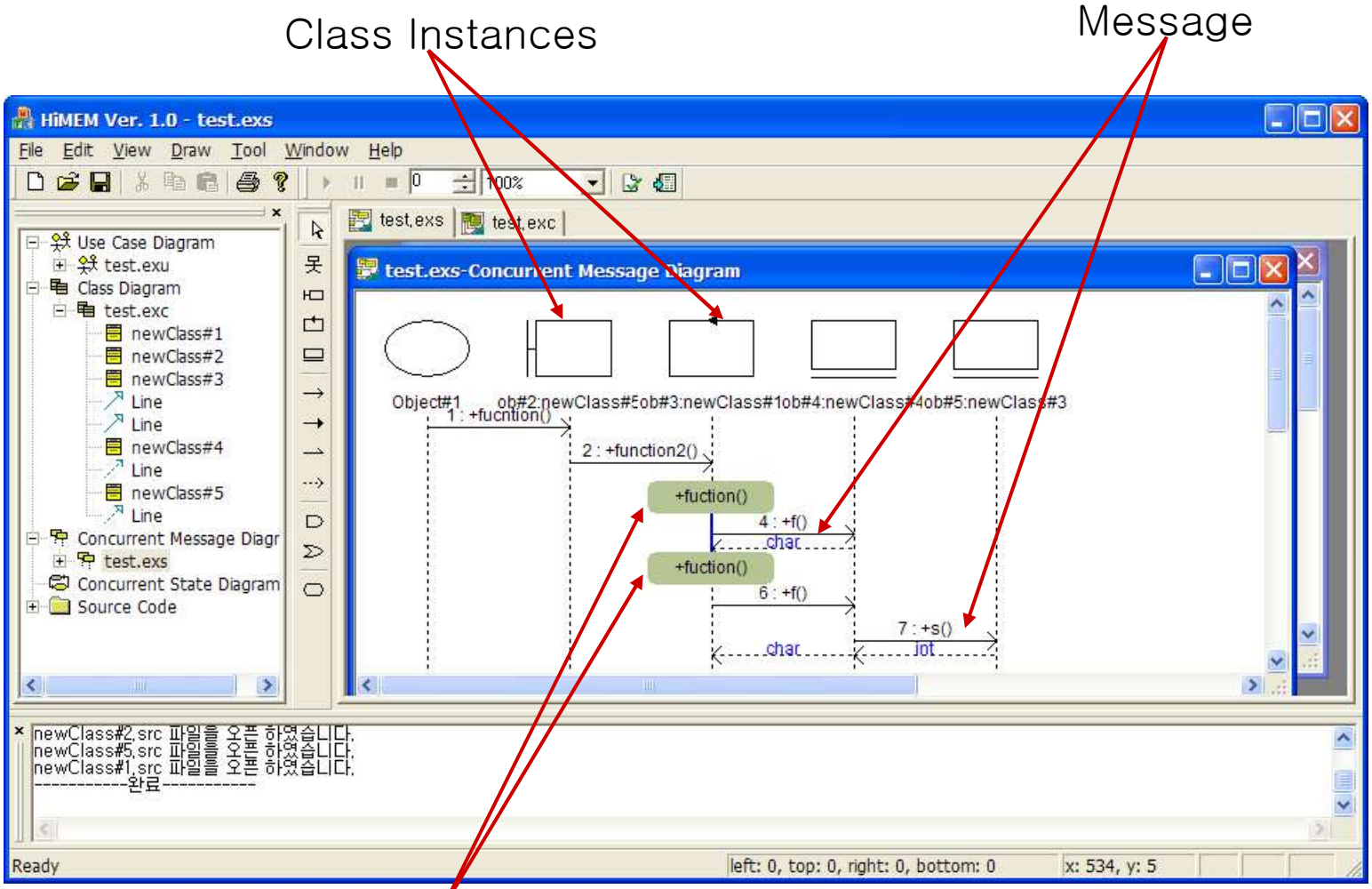
Use Case Diagram



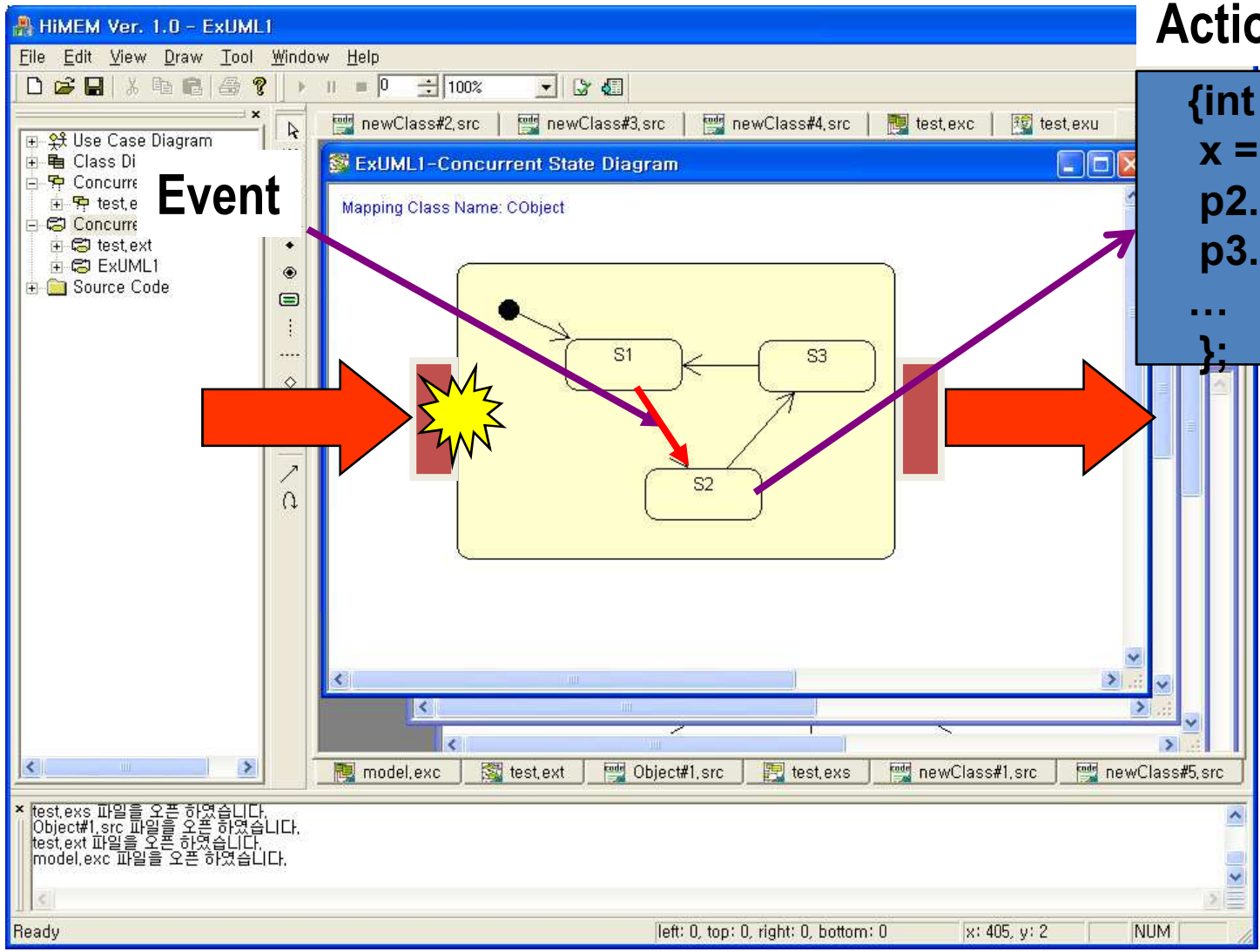
Class Diagram



Concurrent Message Diagram



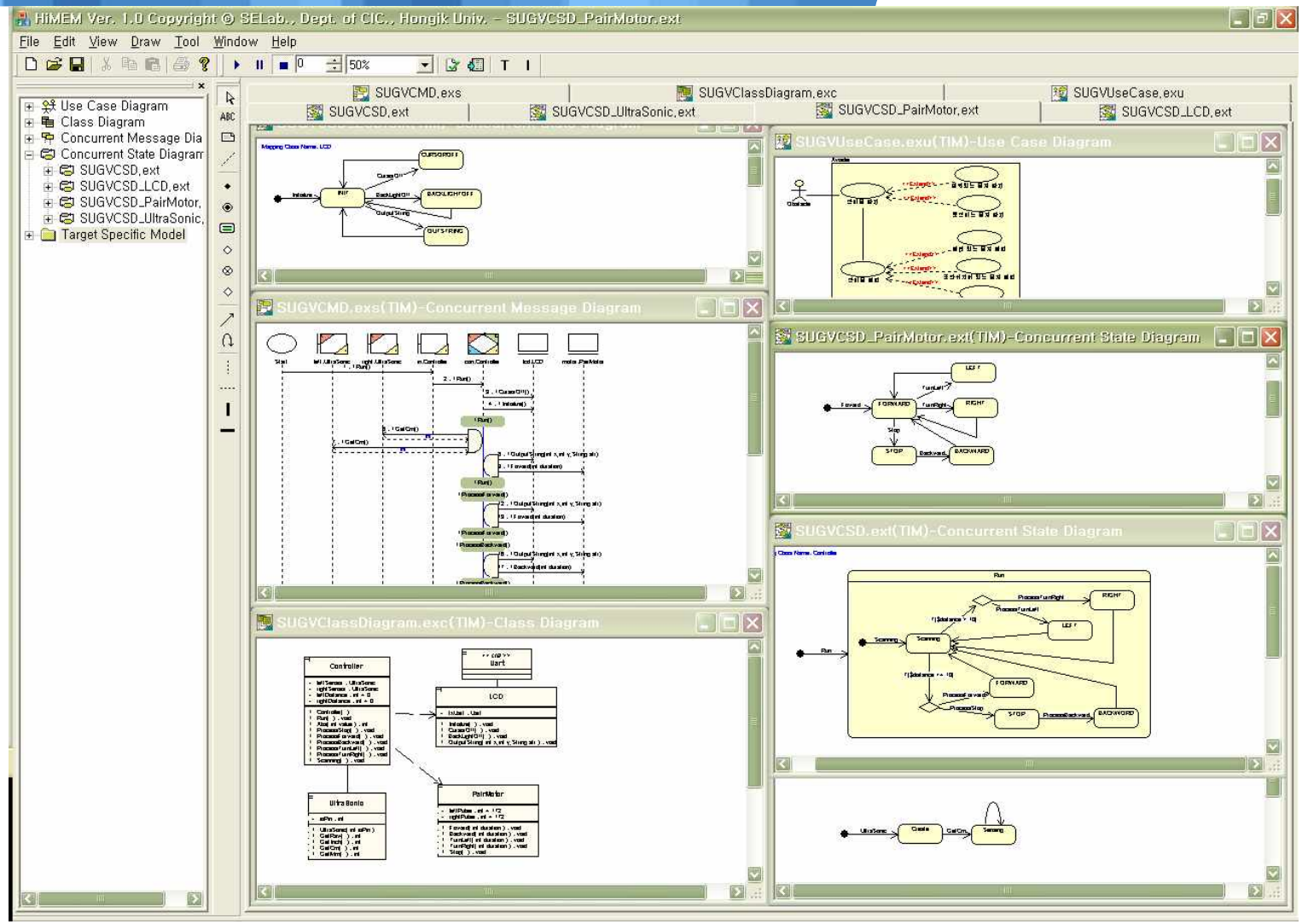
Concurrent State Diagram



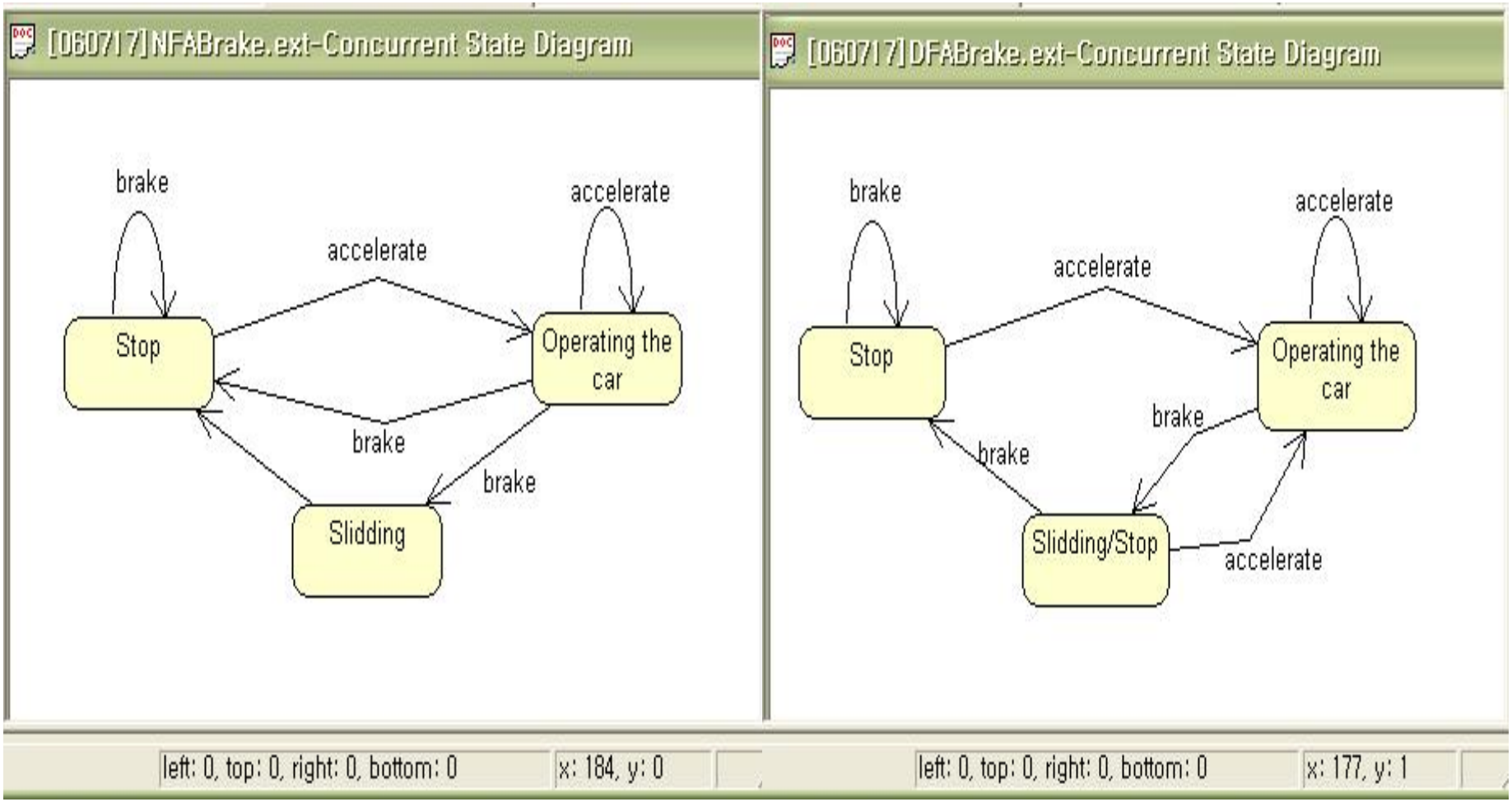
Action Code

```
{int x;
 x = 0;
 p2.send(sig1);
 p3.send(sig2);
 …
 };
```

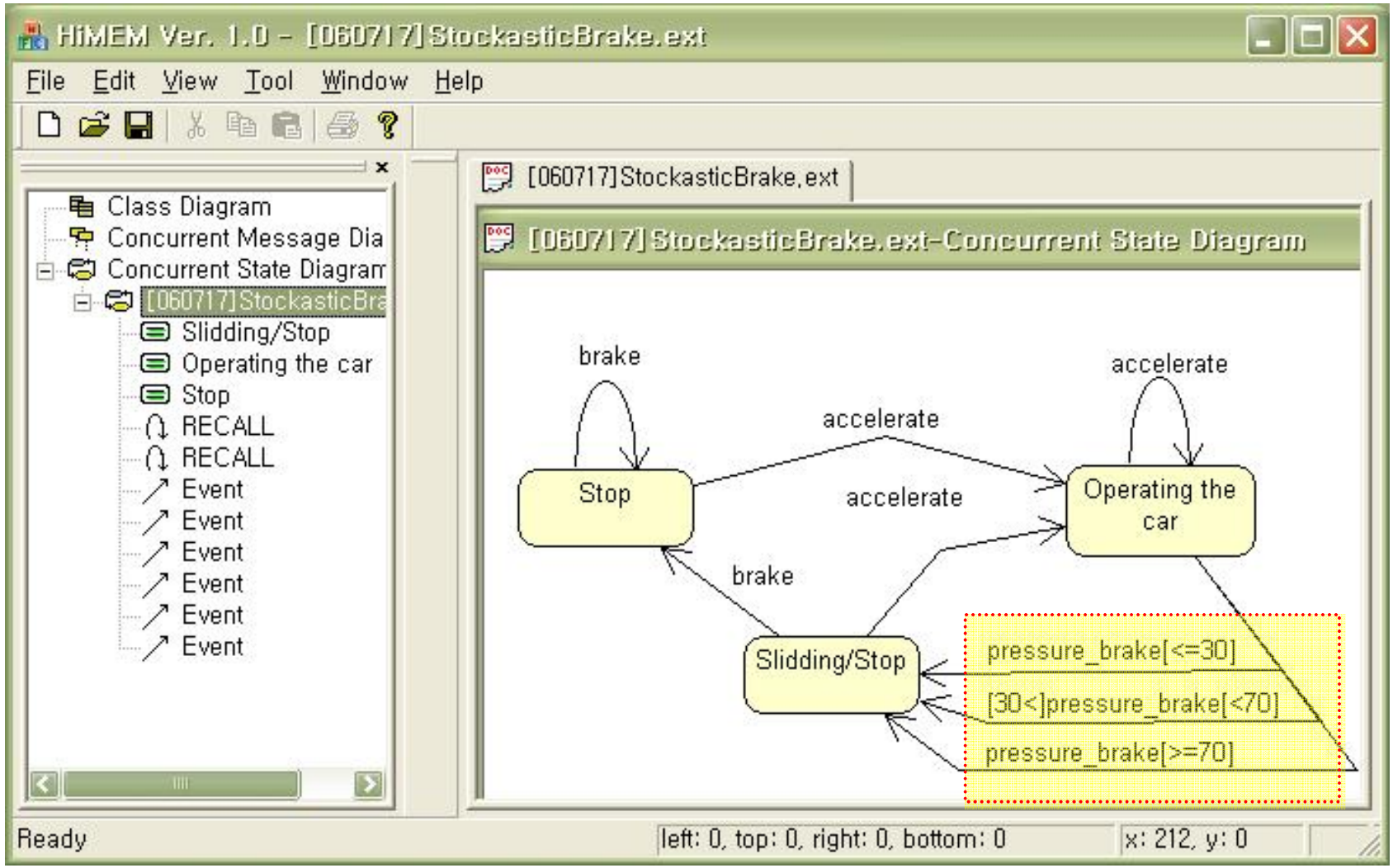
도구 설명 – 모델링


HiMEM Ver. 1.0 Copyright © SELab., Dept. of CIC., Hongik Univ. – SUGVCSD_PairMotor.ext
File Edit View Draw Tool Window Help
Use Case Diagram
Class Diagram
Concurrent Message Dia
Concurrent State Diagram
SUGVCSD.ext
SUGVCSD_LCD.ext
SUGVCSD_PairMotor.
SUGVCSD_UltraSonic.
Target Specific Model
SUGVCMD.exs
SUGVClassDiagram.exc
SUGVUseCase.exu
SUGVCSD.ext
SUGVCSD_UltraSonic.ext
SUGVCSD_PairMotor.ext
SUGVCSD_LCD.ext
SUGVUseCase.exu(TIM)-Use Case Diagram
SUGVCMD.exs(TIM)-Concurrent Message Diagram
SUGVCSD_PairMotor.ext(TIM)-Concurrent State Diagram
SUGVClassDiagram.exc(TIM)-Class Diagram
SUGVCSD.ext(TIM)-Concurrent State Diagram

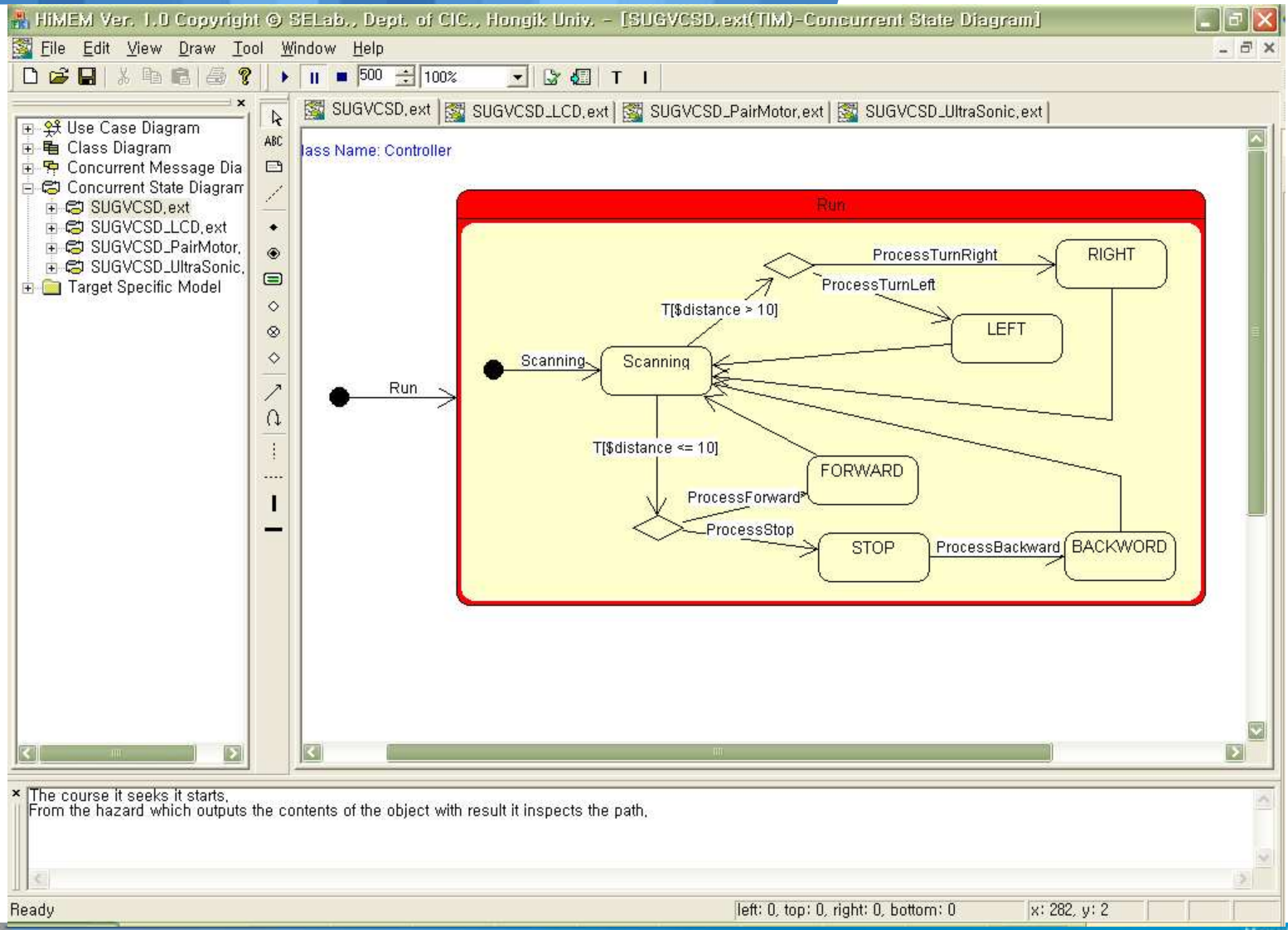
Nondeterministic to Deterministic(진행중)



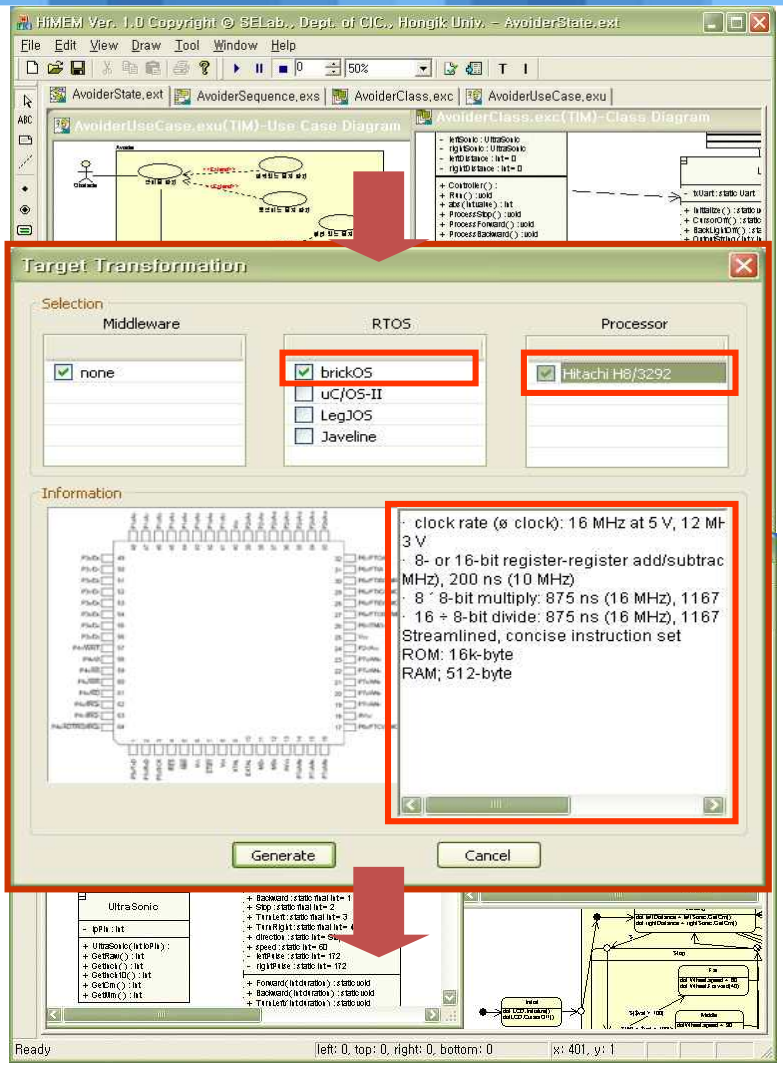
Stochastic Mechanism



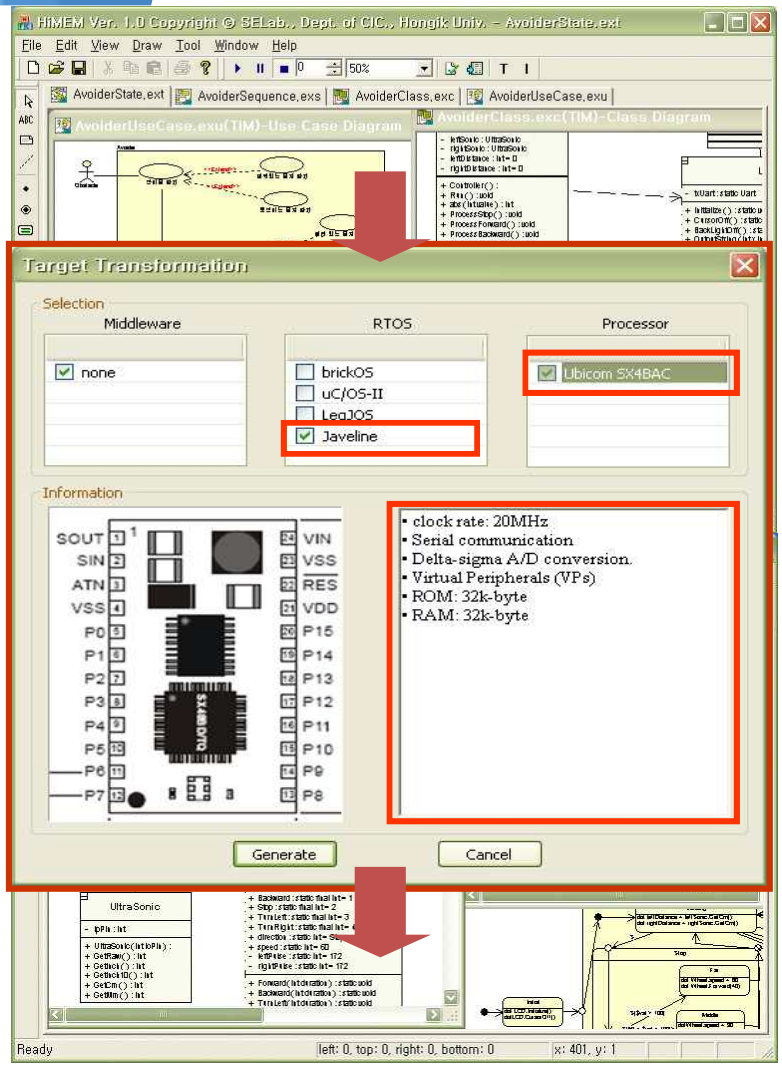
도구 설명 – 시뮬레이션


HiMEM Ver. 1.0 Copyright © SELab., Dept. of CIC., Hongik Univ. - [SUGVCSD.ext(TIM)-Concurrent State Diagram]
File Edit View Draw Tool Window Help
Use Case Diagram
Class Diagram
Concurrent Message Dia
Concurrent State Diagram
SUGVCSD.ext
SUGVCSD_LCD.ext
SUGVCSD_PairMotor.
SUGVCSD_UltraSonic.
Target Specific Model
SUGVCSD.ext
SUGVCSD_LCD.ext
SUGVCSD_PairMotor.ext
SUGVCSD_UltraSonic.ext
lass Name: Controller
Run
ProcessTurnRight
RIGHT
ProcessTurnLeft
T[$distance > 10]
LEFT
Scanning
Scanning
Run
T[$distance <= 10]
FORWARD
ProcessForward
ProcessStop
STOP
ProcessBackward
BACKWORD
The course it seeks it starts,
From the hazard which outputs the contents of the object with result it inspects the path,
Ready
left: 0, top: 0, right: 0, bottom: 0
x: 282, y: 2

TIM → TSM

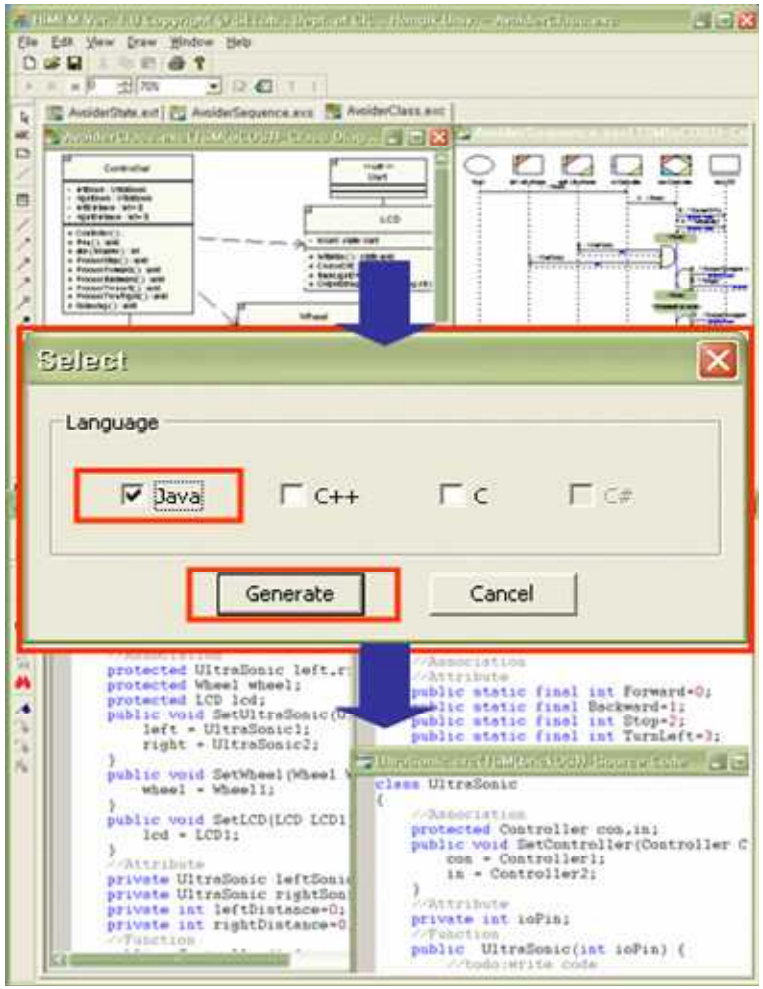


(a) SUGV 1

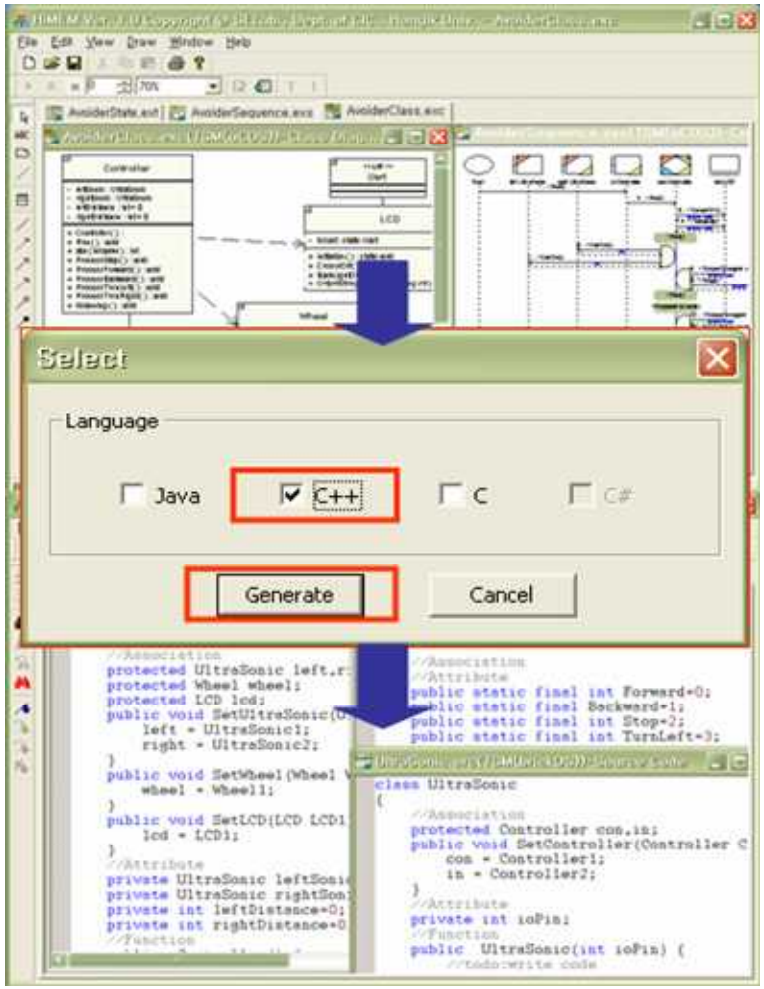


(b) SUGV2

TSM → TDC

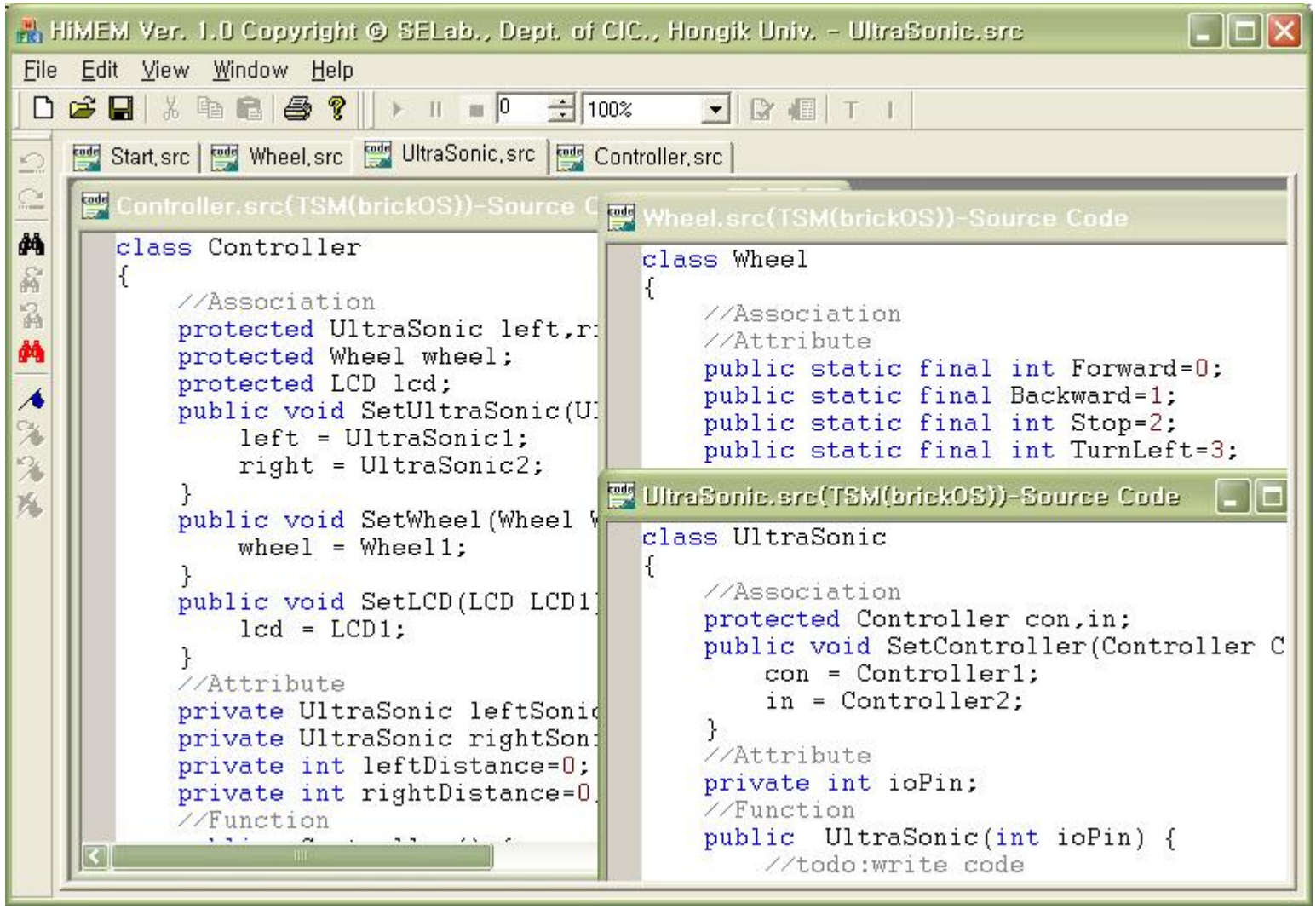


(a) SUGV 1



(b) SUGV2

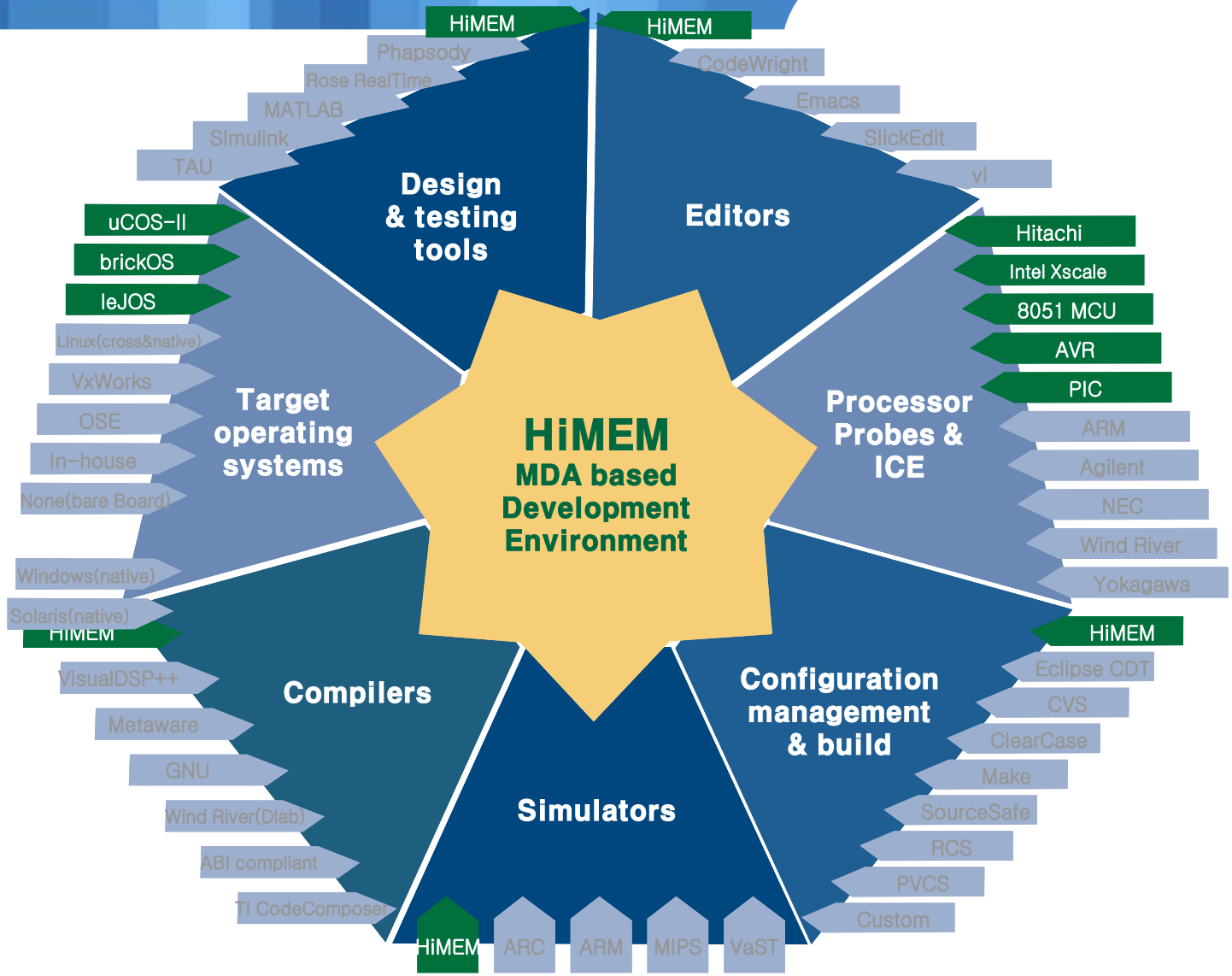
코드생성


HiMEM Ver. 1.0 Copyright © SELab., Dept. of CIC., Hongik Univ. - UltraSonic.src
File Edit View Window Help
Start.src Wheel.src UltraSonic.src Controller.src
Controller.src(TSM(brickOS))-Source C
class Controller
{
//Association
protected UltraSonic left,r
protected Wheel wheel;
protected LCD lcd;
public void SetUltraSonic(U
left = UltraSonic1;
right = UltraSonic2;
}
public void SetWheel(Wheel
wheel = Wheel1;
}
public void SetLCD(LCD LCD1
lcd = LCD1;
}
//Attribute
private UltraSonic leftSonic
private UltraSonic rightSon
private int leftDistance=0;
private int rightDistance=0
//Function
Wheel.src(TSM(brickOS))-Source Code
class Wheel
{
//Association
//Attribute
public static final int Forward=0;
public static final Backward=1;
public static final int Stop=2;
public static final int TurnLeft=3;
UltraSonic.src(TSM(brickOS))-Source Code
class UltraSonic
{
//Association
protected Controller con,in;
public void SetController(Controller C
con = Controller1;
in = Controller2;
}
//Attribute
private int ioPin;
//Function
public UltraSonic(int ioPin) {
//todo:write code

도구 비교

		Tau	Rose RT	Rhapsody	HiMEM
Primary Market		Telecom, Embedded	Telecom, Real-time	Embedded, Real-time	**Embedded, Real-time**
Product Code Generation	Code Generation	80~90%	80~90%	80~90%	**80~90%**
	Readable Codes	No	Yes	Yes	**Yes**
	Source level Debugging	Absent	Supported	Supported	**Supported**
	Rules based Code Generation	Absent	Supported	Supported	**Supported**
Roundtrip Engineering	Model/Code Associativity	Absent	Supported, but some restrictions	Supported	**Supported**
	Dynamic Model Code View	Absent	Absent	Supported	**Supported**
	Model/Code Synchronization	Absent	Absent	Supported	**Supported**
Reverse Engineering	Reverse Engineering	Supported	Supported	Supported	**Absent**
	Forward Generation	Supported	Supported	Supported	**Supported**
Model Execution	Executable Model	Supported	Supported, but using by VM	Supported	**Supported**
	Simulation	Supported	Supported	Supported	**Supported**
	Simulating Diagrams	Sequence, Statechart	Sequence, Statechart	Sequence, Statechart, Activity	**CMD, CSD**
	Design-level debugging, animation	Absent	Absent	Supported	**Supported**
	Requirements validation	Supported	Absent	Supported	**Supported**
	Use Case tracing	Absent	Absent	Supported	**Supported**
	Design-level debugging, animation	Absent	Absent	Supported	**Supported**
	Reverse Fork/Join	Absent	Absent	Absent	**Supported**

도구의 영역


HiMEM
MDA based
Development
Environment
Design
& testing
tools
HiMEM
Phapsody
Rose RealTime
MATLAB
Simulink
TAU
Editors
HiMEM
CodeWright
Emacs
SlickEdit
vi
Processor
Probes &
ICE
Hitachi
Intel Xscale
8051 MCU
AVR
PIC
ARM
Agilent
NEC
Wind River
Yokagawa
Configuration
management
& build
HiMEM
Eclipse CDT
CVS
ClearCase
Make
SourceSafe
RCS
PVCS
Custom
Simulators
HiMEM
ARC
ARM
MIPS
VaST
Compilers
HiMEM
VisualDSP++
Metaware
GNU
Wind River(Dlab)
ABI compliant
TI CodeComposer
Target
operating
systems
uCOS-II
brickOS
leJOS
Linux(cross&native)
VxWorks
OSE
In-house
None(bare Board)
Windows(native)
Solaris(native)

Lecture 6
임베디드 UML 도구 소개

도구 시연

Lecture 7
설계 사례

설계 사례 (Doorlock)

-Use case Diagram

-Class Diagram

-Message Diagram

-State Diagram

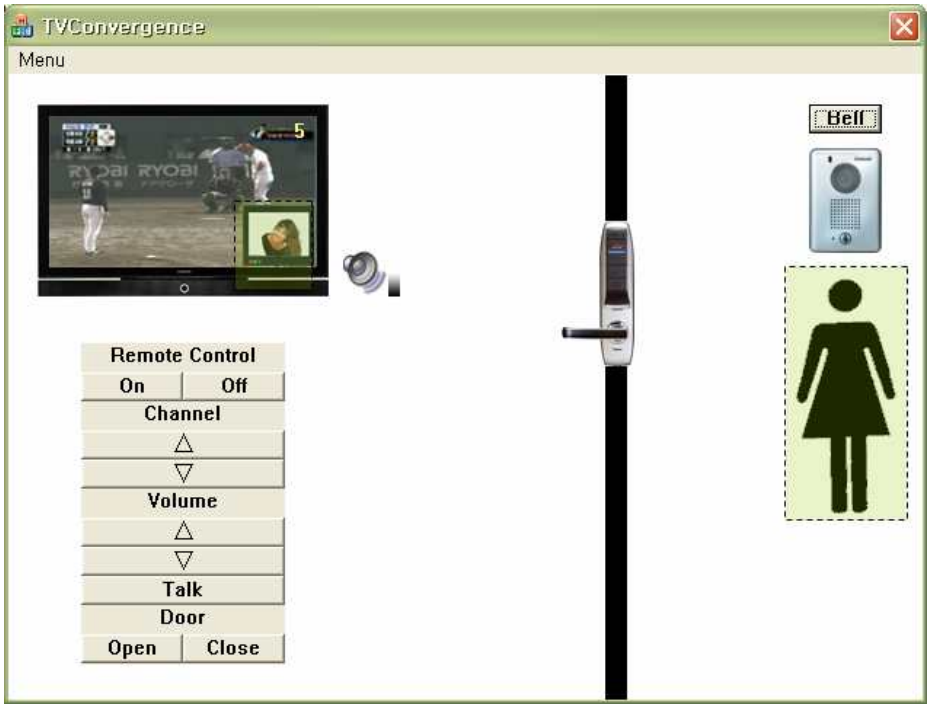
Interphone + TV

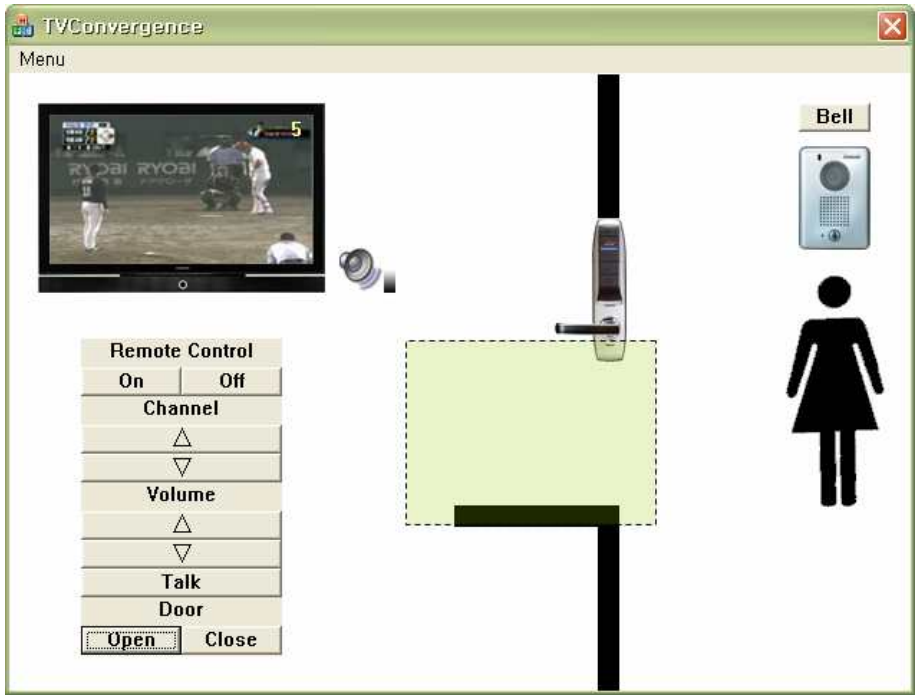
❖ 이 기종간의 네트워크로 인한 제품 기능의 확대

- 유비쿼터스 시대의 컨버전스 요구 급증
- 예) TV에서 외부인의 방문을 확인할 수 있는 기능 추가

기능 확대	단일 제품 → 이 기종 컨버전스 제품
제품의 변화	

구현


TVConvergence
Menu
Bell
Remote Control
On
Off
Channel
Volume
Talk
Door
Open
Close


TVConvergence
Menu
Bell
Remote Control
On
Off
Channel
Volume
Talk
Door
Open
Close

Lecture 8
실습

The End

Thank You !

김 영 철

홍익대학교 컴퓨터정보통신

(041)865-2477, 016-659-7518

소프트웨어공학연구실

bob@hongik.ac.kr http://selab.hongik.ac.kr