CSE053 Software Engineering
Lecture: UML

Software engineering laboratory
Yeungnam university

영남대학교 Yeungnam University

YEUNGNAM UNIVERSITY 1947 영남대학교

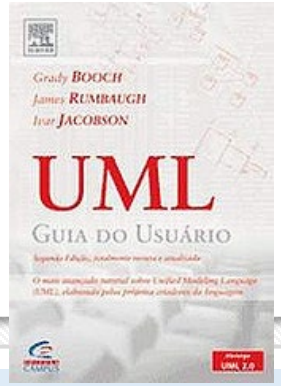
UML의 개요

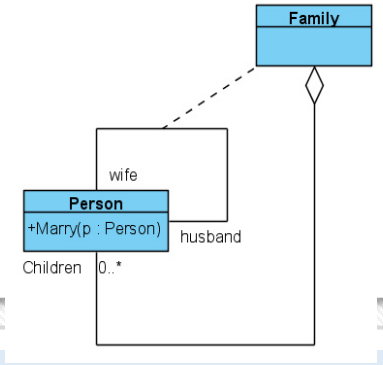
▪ UML의 개념

- UML: Unified Modeling Language
- UML은 1994년 소프트웨어 방법론의 선구자인 Grady Booch, James Rumbaugh, Ivar Jacobson에 의해 연구
- 1997년 객체관리그룹(OMG, Object Management Group)에서 여러 표기법을 통합하여 UML 발표
- UML은 객체지향 시스템 개발 분야에서 가장 우수한 모델링 언어로 인식되고 있음

→ 구현에 앞서 표준화되고 이해되기 쉬운 방법으로 소프트웨어를 설계하여 효율적으로 의사소통 할 수 있는 메커니즘을 제공

UNIFIED MODELING LANGUAGE™





UML의 개요

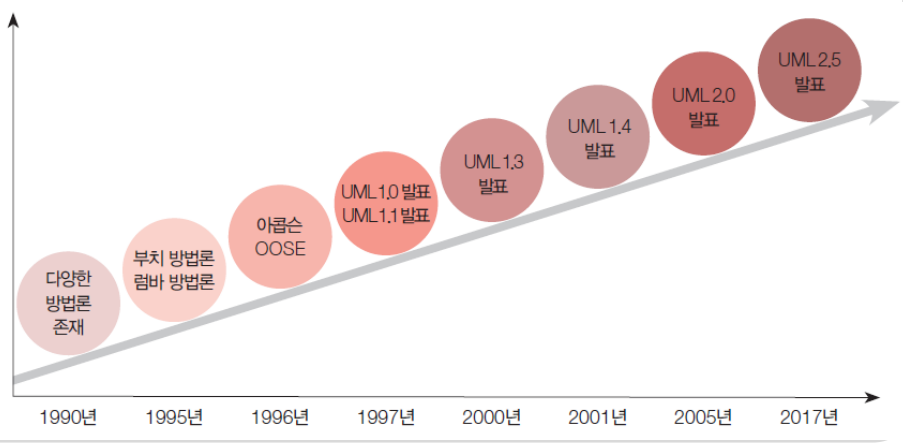
▪ UML의 역사

- 1990년대에는 OOD/Booch, OMT, OOAD, OOSD 등과 같은 많은 방법론들은 실제 시스템을 구축하는 데 있어서 각각의 객체지향 기술들이 갖는 방법과 심벌이 서로 다름

  → 호환성 측면에서 장애

- UML은 많은 객체지향 방법론들 때문에 발생하는 문제점을 인식하고, 이의 해결을 위해 통합된 객체지향 모델링 언어를 정의하기 위한 노력의 결과로 탄생

시기	내용
1980년대 말~1990년대 초	50여 가지의 다양한 표현 기법과 방법론 존재
1995년	부치 방법론과 럼바의 OMT 방법론 통합 (OOPSLA'95에 발표)
1996년	야콥슨의 OOSE 방법론 추가 UML 0.9 정의
1997년	1월: UML 1.0 발표(Microsoft, Oracle, IBM, HP 참여) 9월: UML 1.1 발표(OMG에 표준화안 상정) 11월: OMG 표준 인증
1999년	UML 1.3 발표
2001년	UML 1.4 발표
2003년	UML 2.0 승인



UML의 개요

▪ UML의 용도

- 시스템을 만들기 전에 모델을 만드는 것은 건물을 짓기 위한 설계도처럼 아주 중요한 역할
- 시스템을 만드는 데도 어휘와 규칙을 마련하여 시스템을 개념적, 물리적으로 표현하는 모델이 필요
- 성공적으로 시스템을 만들기 위해서는 객체지향적인 분석과 설계를 위한 표준으로 인정받는 모델링 언어인 UML이 필요
- 개발방법론에 관계없이 적용 가능

[표] UML이 모델링 언어로 사용될 수 있는 대표적인 시스템 유형

시스템 종류	설명	시스템 예
정보 시스템	사용자로부터 정보를 입수, 가공, 저장하고 적절한 양식으로 사용자에게 출력하는 작업을 한다.	각종 공공기관, 금융기관, 회사에서 해당 기관 고유의 업무 지원
기술적 시스템	기계 등을 제어하는 시스템이다.	통신 장비, 군 장비, 공장의 기계
내장 시스템	하드웨어에 내장되어 수행되는 시스템이다.	휴대전화기, 세탁기, TV, 전기밥솥

UML의 개요

▪ UML의 특징

- UML은 가시화 언어이다

  → UML은 소프트웨어의 개념 모델을 시각적인 그래픽 형태로 작성

  → 표기법에 있어서는 각 symbol에 명확한 정의가 존재

  → 개발자들 사이에 오류 없는 원활한 의사소통이 가능

- UML은 구축 언어이다

  → UML은 Java, C++, Visual Basic과 같은 다양한 프로그래밍 언어로 표현

  → UML로 명세화된 설계 모델은 프로그램 소스 코드로 변환하여 구축 가능

  → 구축되어 있는 소스를 UML로 역변환하여 분석하는 역공학이 가능

Modeling의 개념

- **Modeling**
  - 시스템을 구축할 때 개발자가 고민하고 결정하는 모든 활동

    → 어떤 기능을 제공? GUI는 어떻게? 어떤 클래스들이 필요한지?
- **Model**
  - Modeling 활동의 결과

    → 분석 모델, 설계 모델 등
- **Modeling Language**
  - Model을 표현할 때 사용되는 언어

    → UML은 바로 이러한 목적으로 사용될 수 있게 정의된 모델링 언어 중 하나

	Modeling	Programming
목적	구축할 시스템의 모습 정의	시스템의 실제 구현
세부 수행 활동	요구사항 정의, 분석 (Analysis), 설계 (Design)	소스코드 편집, 컴파일 (Compile), 시험 (Testing), 디버깅 (Debugging)
결과물	모델	소스 코드를 포함한 구현된 시스템
표기법	모델링 언어(UML, ERD, DFD 등)	프로그래밍 언어(C, C++, Java 등)

Modeling의 개념

```c
#include <stdio.h>
int main() {
      int a, b;
      scanf("%d %d", &a, &b);
      printf("a + b = %d", a + b);
 }
```

(a) C로 구현한 덧셈 프로그램

```java
import java.util.Scanner;
public class AddDemo {
      public static void main(String[] args) {
            Scanner in = new Scanner(System.in );
            int a = in.nextInt();
            int b = in.nextInt();
            System.out .printf("%d + %d = %d", a, b, a+b);
    }
}
```

(b) 자바로 구현한 덧셈 프로그램

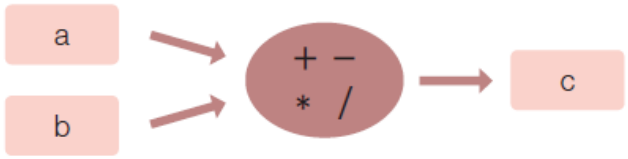
```python
def add():
      num = int(input("num1:"))
      num2 = int(input("num2:"))
      result = num + num2
      print("두수의 합은 " result)
```

(c) 파이썬으로 구현한 덧셈 프로그램

그림 1-3 다양한 언어로 구현한 덧셈 프로그램

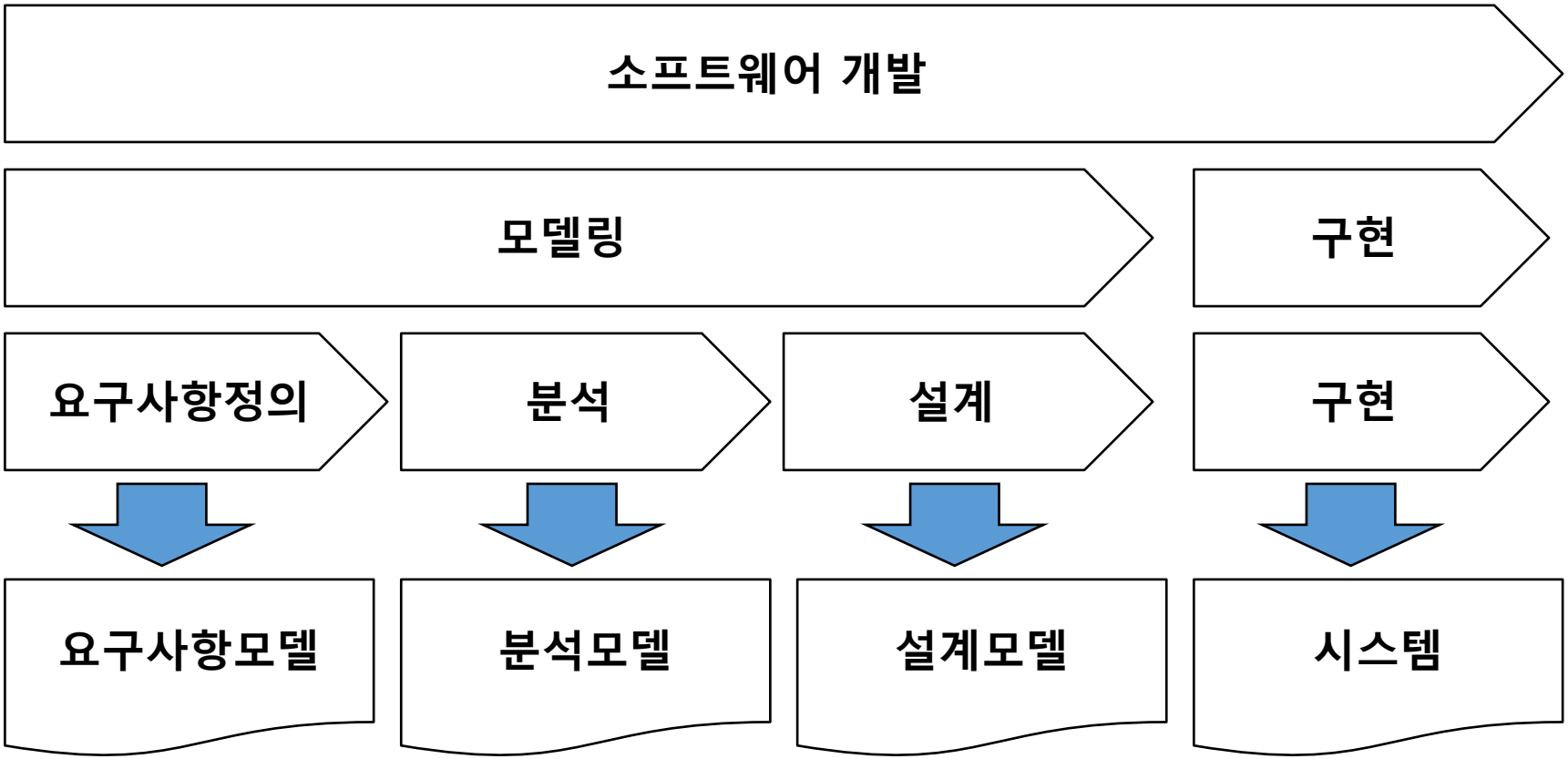
개발하고자 하는 프로그램을 시각적으로 표현하는 것이며,
이때 의뢰자의 요구에 맞게 쉽게 수정해서
결과적으로 유지보수 기간을 줄여 생산성을 높일 수 있음

(a) 덧셈 연산 프로그램의 시각적 표현

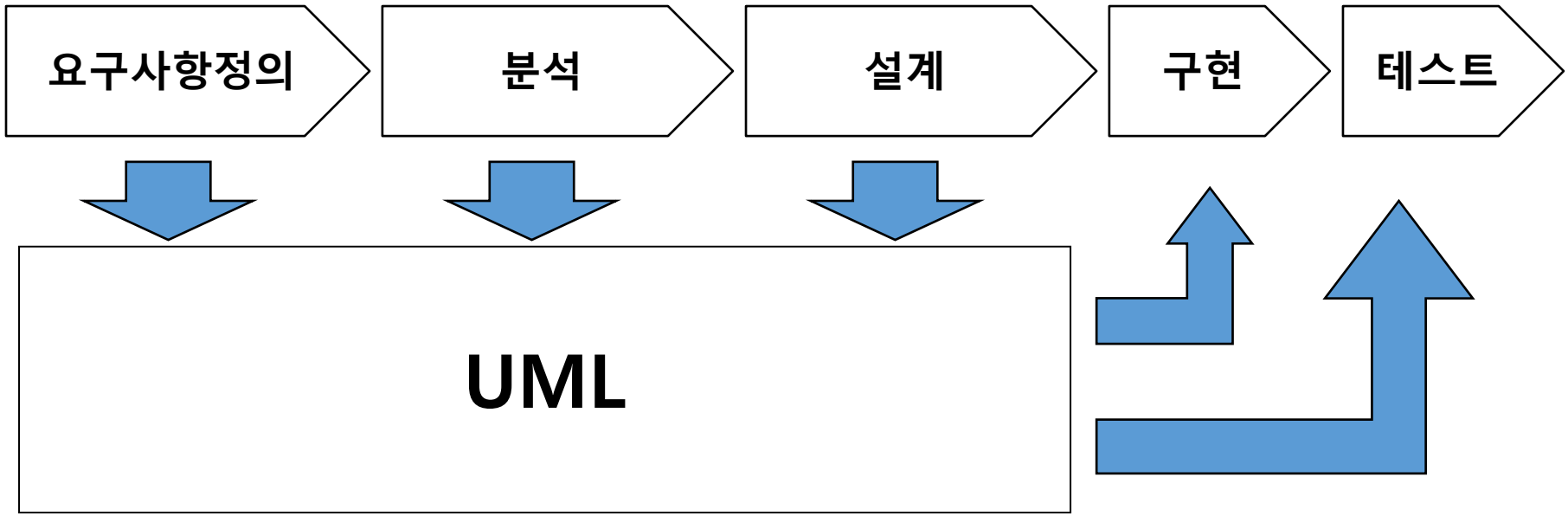


(b) 사칙연산 프로그램의 시각적 표현

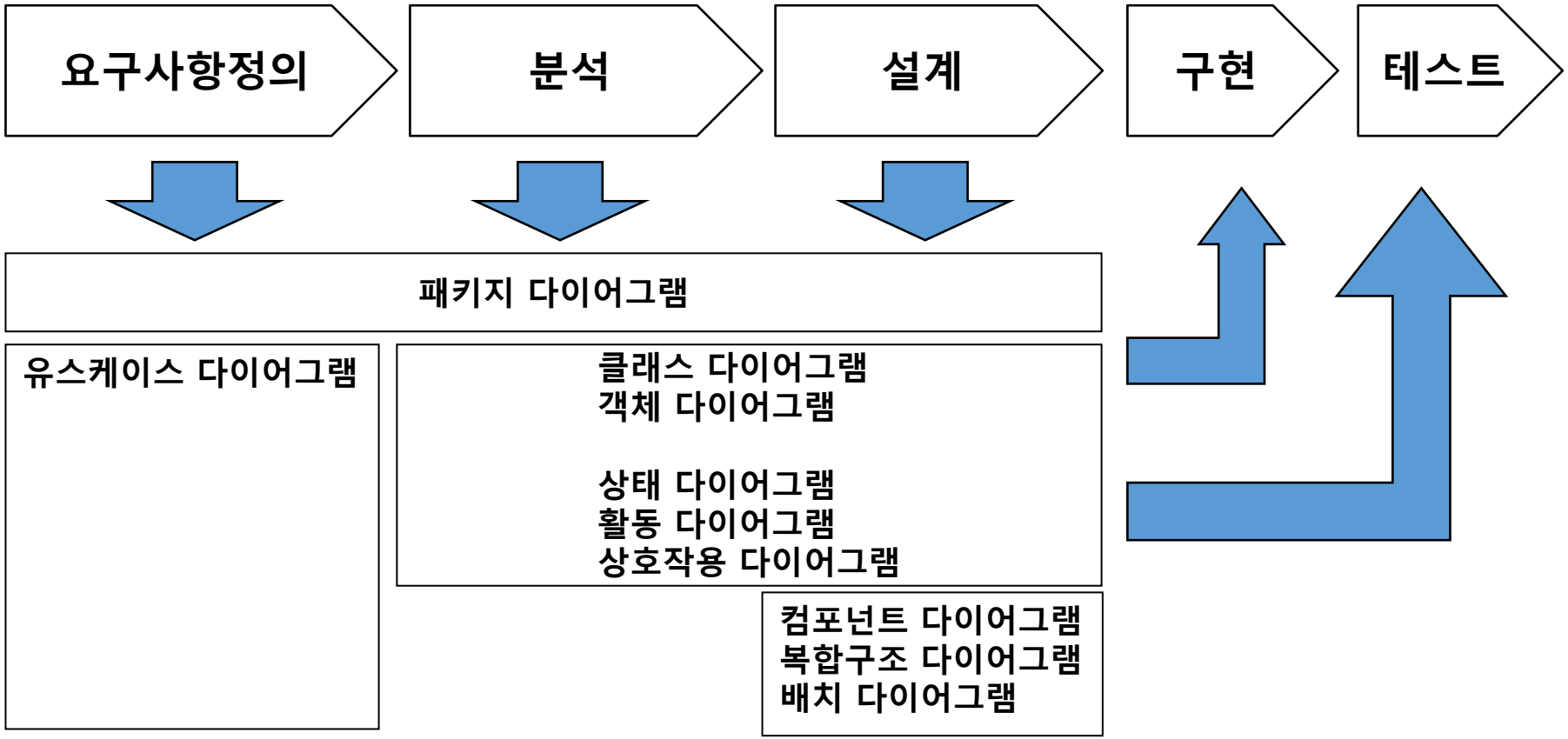
Modeling의 개념



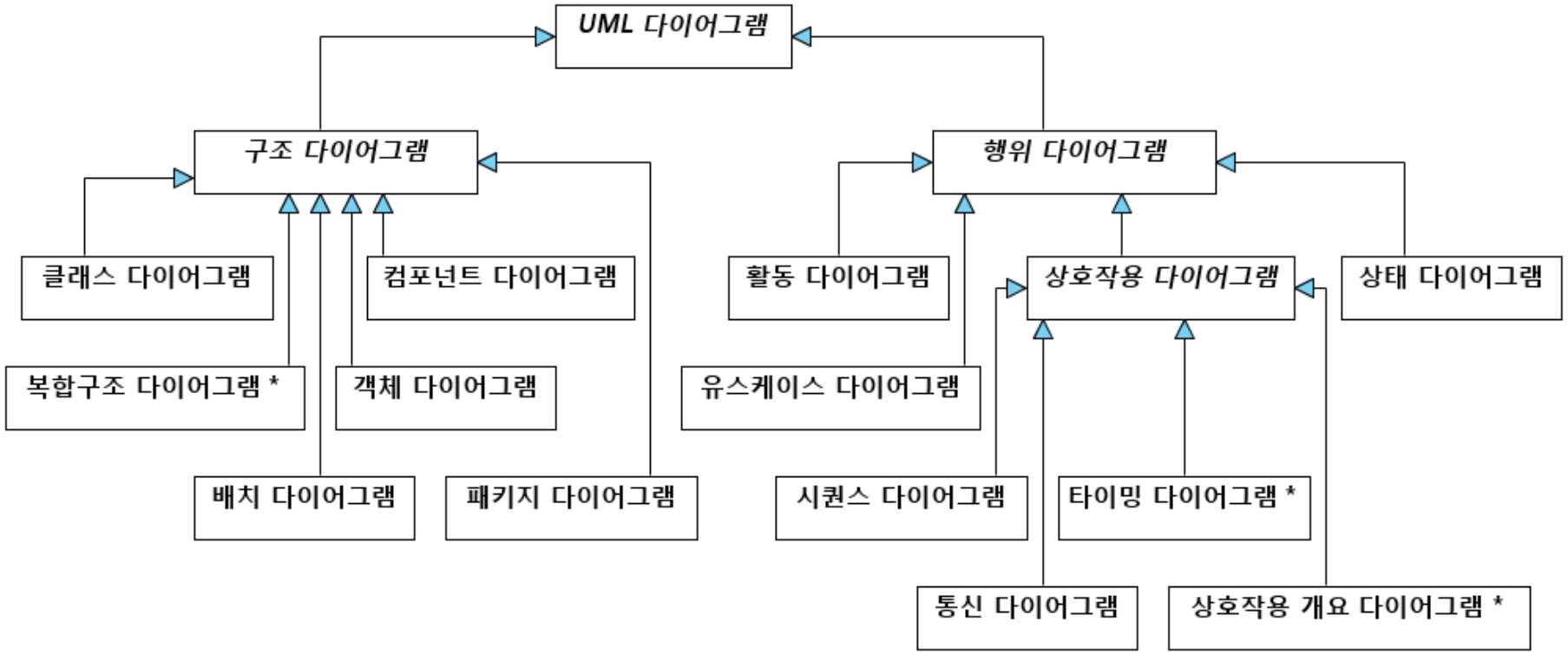
- 개발 전 과정에서 사용됨



▪ 개발 단계별로 사용되는 다이어그램 존재



UML Diagram Texonomy (분류체계)



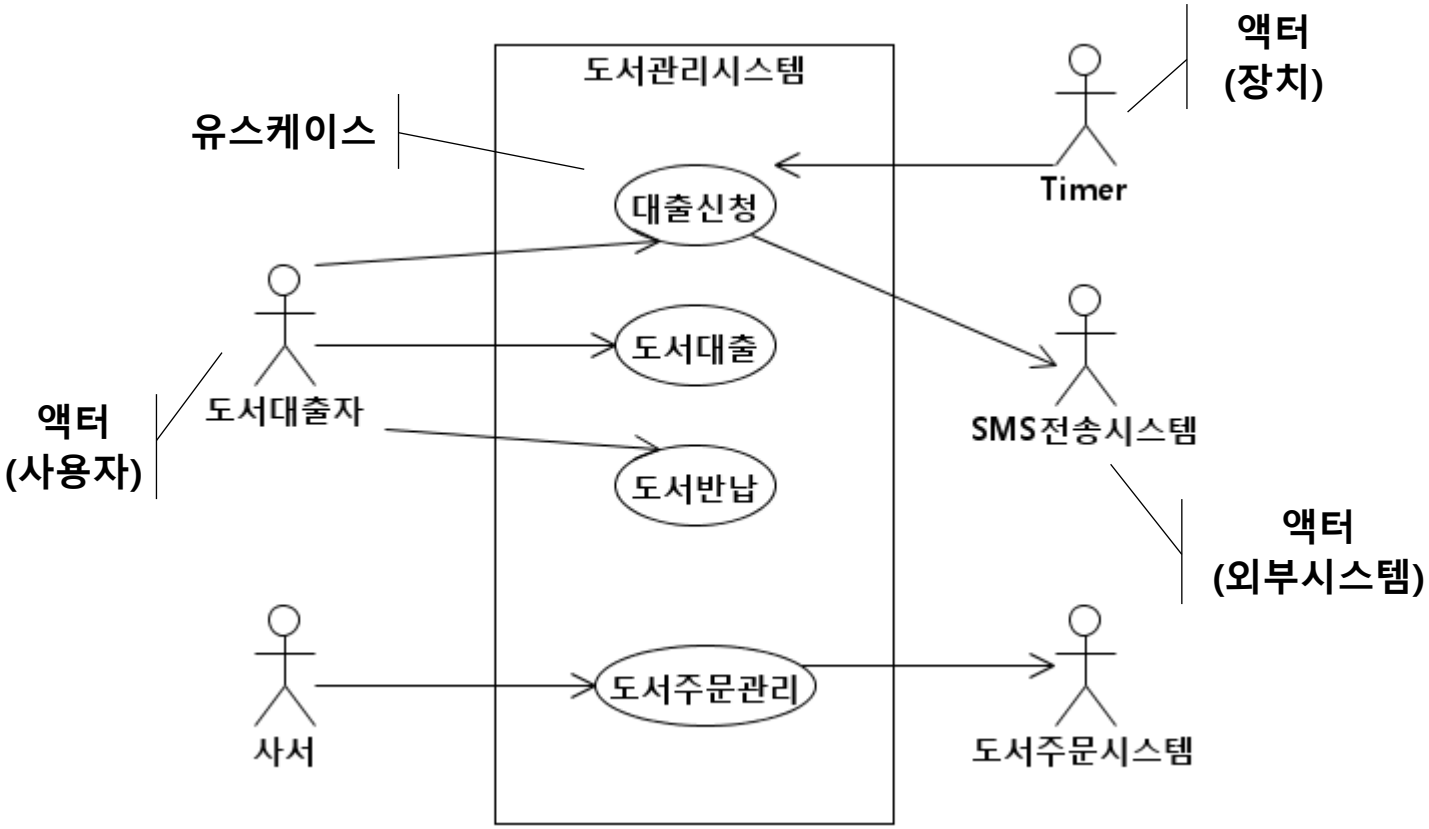
구조 다이어그램(Structural diagram)

명칭	내용	비고
Class diagram	시스템을 구성하는 **클래스** 표현	논리적 수준
Package diagram	많은 수의 모델 요소(예, 클래스, 논리적 컴포넌트)들을 **패키지**를 이용하여 조직화함	
Component diagram	시스템을 구성하는 **논리적 컴포넌트** 표현	
Composite Structure diagram	**논리적 컴포넌트**의 내부를 **파트(part)**와 **연결자(connector)**로 표현	
Deployment diagram	시스템을 구성하는 **노드**와 **통신 경로**, 배치되는 **물리적 컴포넌트**를 표현	물리적 수준

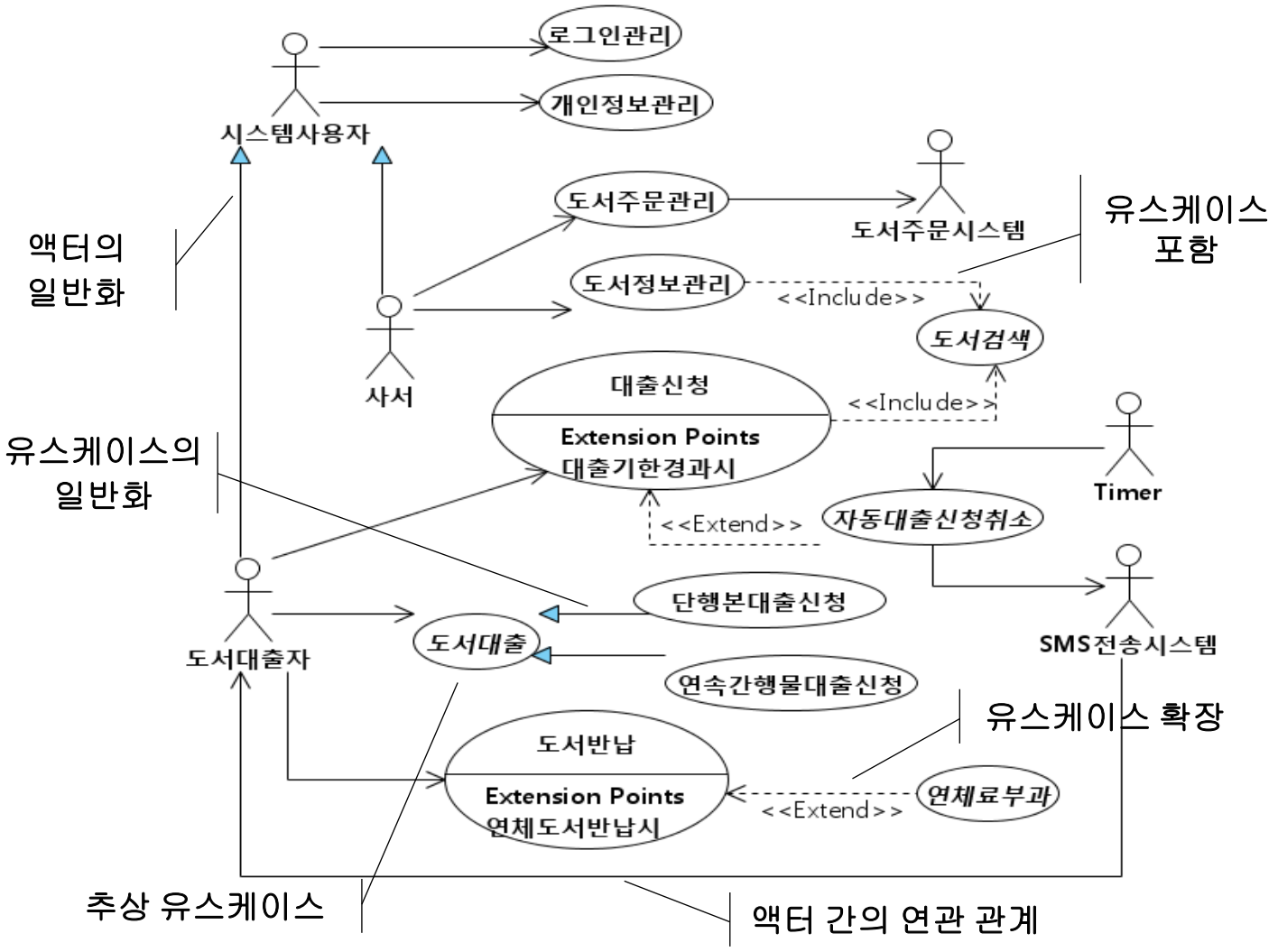
행위 다이어그램(Behavior diagram)

명칭	내용		비고
Use case diagram	시스템의 외부 요소와 기능적 요구사항을 **액터**와 **유스케이스**로 표현		시스템의 행위
State machine diagram	개별 대상의 동적 행위를 **상태**와 **전이**로 표현		개별 구성 요소의 행위
Activity diagram	개별 대상의 동적 행위를 **활동**으로 표현		
Interaction diagram	Sequence diagram	상호작용을 구성 요소간의 **시간적 순서에 따른 메시지 전달**로 표현	구성 요소간의 상호작용
	Communication diagram	상호작용을 **구성 요소간의 관계를 바탕을 둔 메시지 전달**을 표현	
	Interaction overview diagram	**여러 상호작용의 관계**를 상위 수준에서 표현	
	Timing diagram	구성 요소의 **상태 변화**의 구체적인 **시간**으로 표현	

Use case diagram

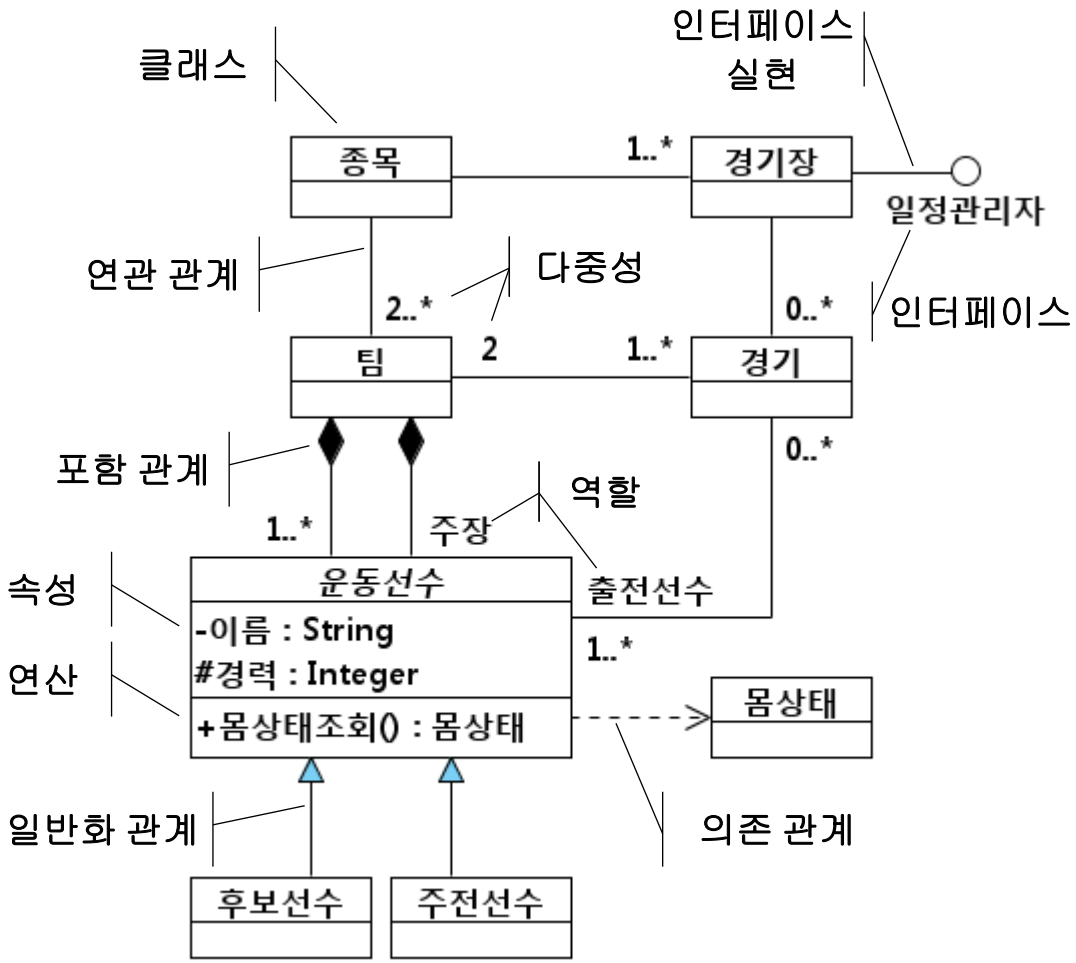


Use case diagram



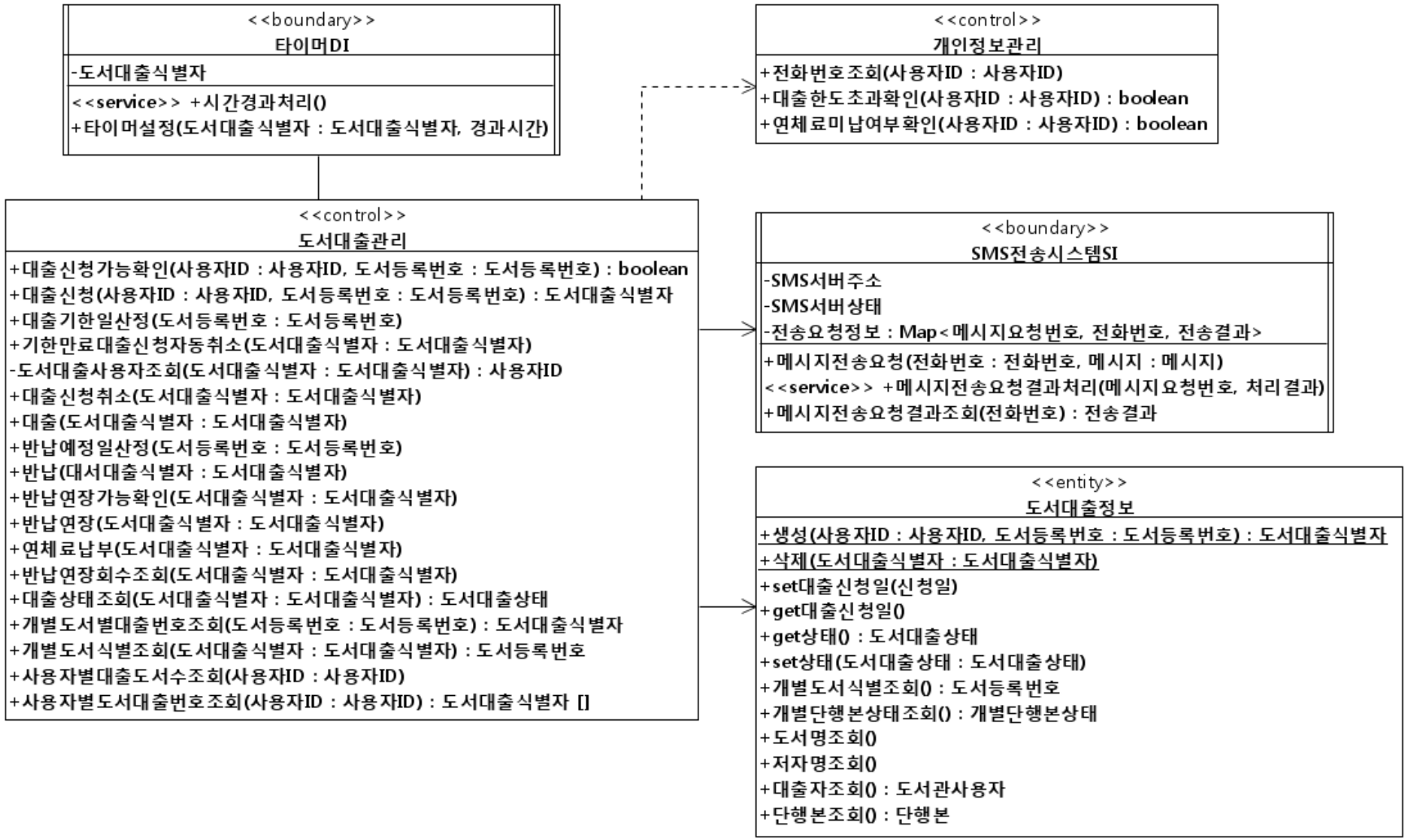
Class diagram

- 시스템을 구성하는 Class들, Class간의 관계 표현



Class diagram

- 시스템을 구성하는 Class들, Class간의 관계 표현

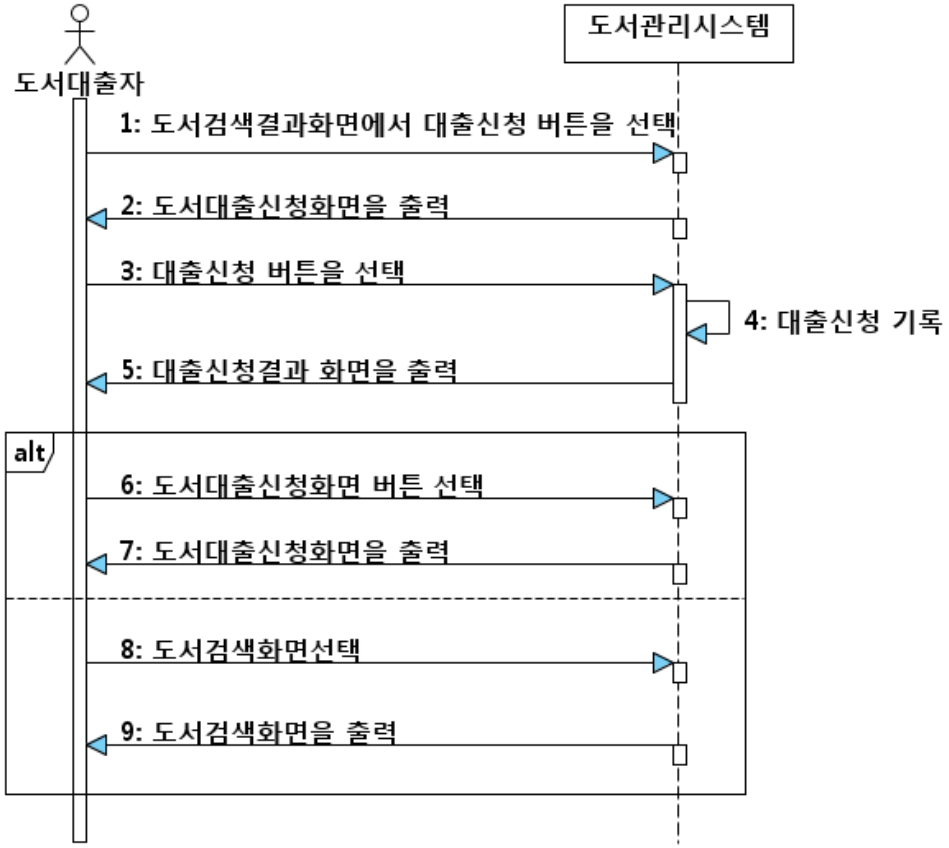


Sequence diagram

- **Interaction diagram**
  - **Sequence diagram**
  - Communication diagram
  - Interaction overview diagram
  - Timing diagram
- **참여자간의 시간적 순서에 따른 상호작용을 표현**
- **객체들간의 cowork를 설명**
  - **객체들간의 동작을 표시**

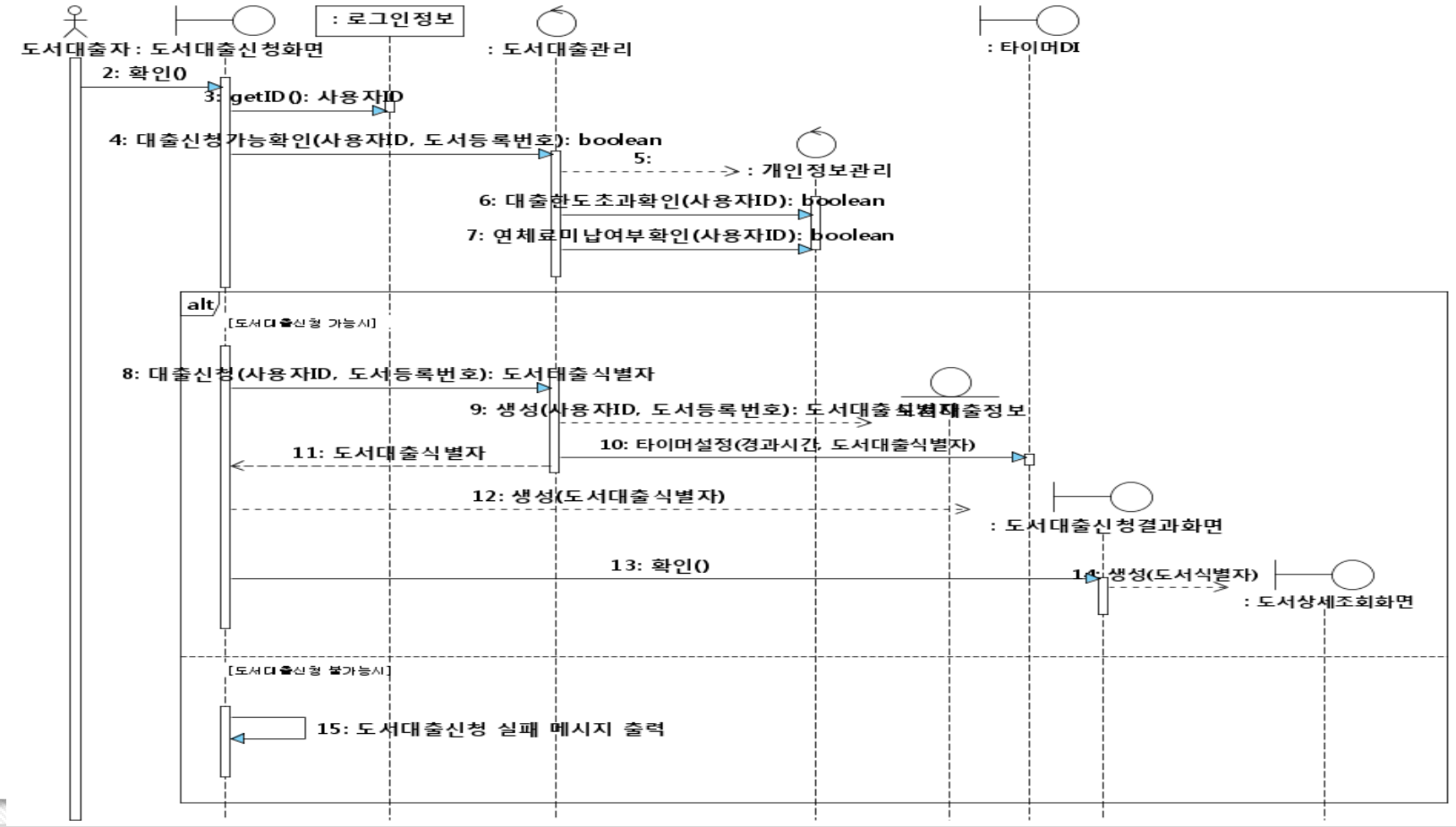
Sequence diagram

- **Use case 시나리오 표현: “도서대출신청” use case**
  - **각 Use case별 sequence diagram 필요**



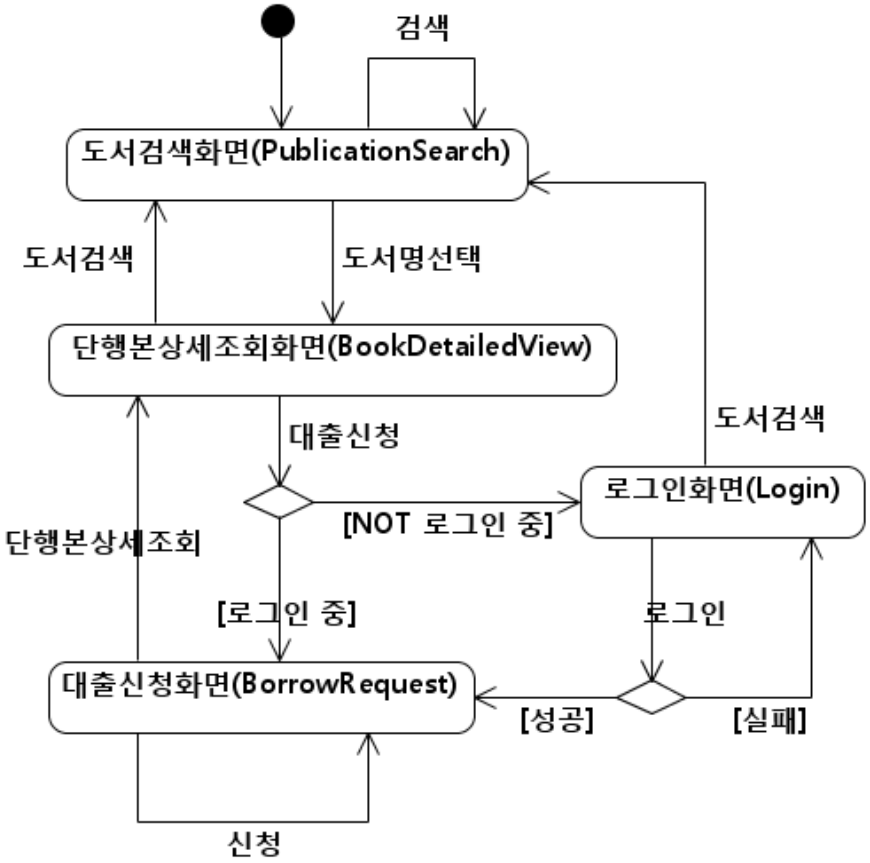
Sequence diagram

- Use case 시나리오 표현: “도서대출신청” use case
  - 각 Use case별 sequence diagram 필요



State machine diagram

- **SW 시스템이 특정 시점에서 가지는 상태(state) 표현**
  - **Event에 의해 상태 이동**

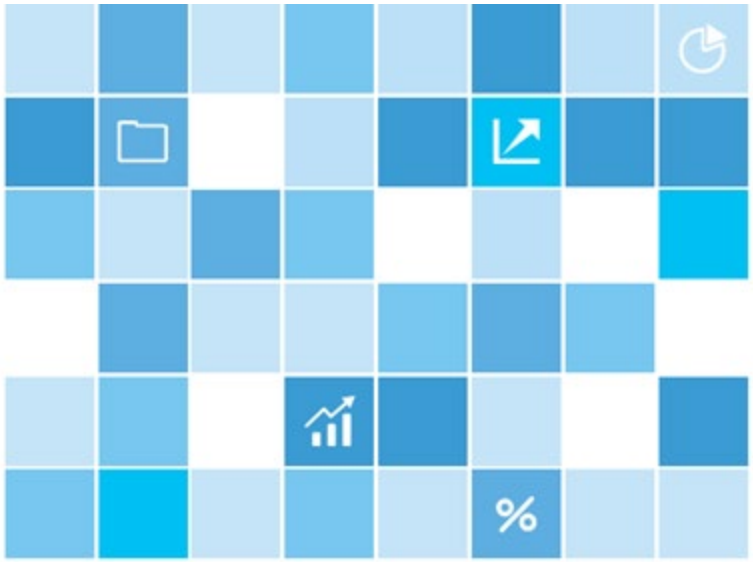


Summary

유형	다이어그램	핵심 개념	수준	주요 활용 단계		
				요구사항 정의	분석	설계
구조	클래스 다이어그램	클래스	논리적		√	√
	객체 다이어그램	객체			√	√
	패키지 다이어그램	패키지		√	√	√
	컴포넌트 다이어그램	논리적 컴포넌트				√
	복합구조 다이어그램	파트 연결자				√
	배치 다이어그램	노드 물리적 컴포넌트	물리적			√

Summary

유형	다이어그램	핵심 개념	수준	주요 활용 단계		
				요구사항 정의	분석	설계
행위	유스케이스 다이어그램	액터 유스케이스	논리적	√		
	상태 다이어그램	상태 전이			√	√
	활동 다이어그램	활동			√	√
	시퀀스 다이어그램	생명선 메시지 복합적 부분			√	√
	통신 다이어그램	생명선 링크 메시지			√	√
	타이밍 다이어그램	생명선 시간적 상태 변화			√	√
	상호작용 개요 다이어그램	상호작용 간의 관계			√	√



Thank You