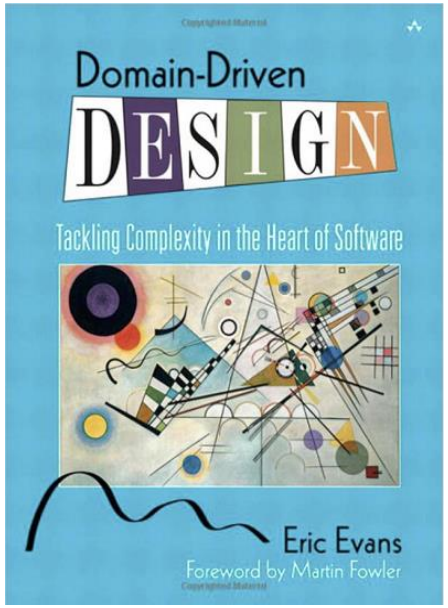
Domain Driven Design –More2 (ver 1.1)

목 차

1. 누락된 개념 찾기



1.1 데이터로 존재하는 개념들
1.2 보편 언어로 논의
1.3 정확성과 언어 확장

1.1 데이터로 존재하는 개념들

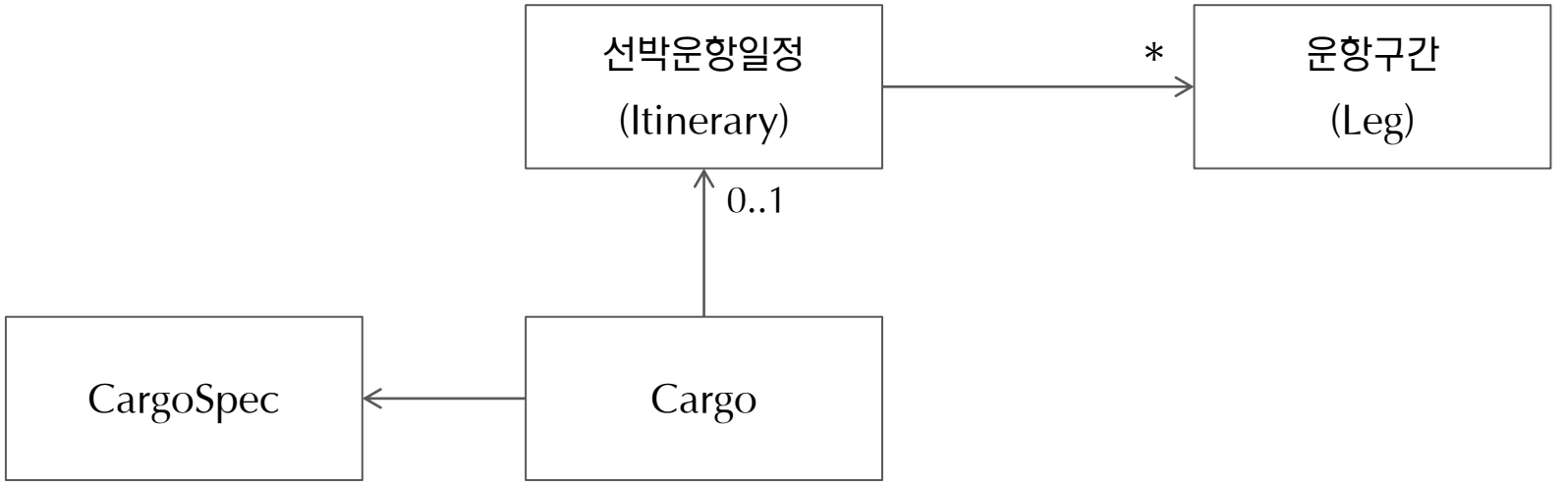
✓ 화물 예약 서비스를 개발할 때, 화물정보, 운항정보, 선적 위치, 하역 위치 정보를 포함하는 테이블을 설계했습니다.
✓ 이 테이블을 통해 원하는 정보는 얻을 수 있지만, 우리가 필요한 보편 언어는 갖지 못한 상태입니다.
✓ 보편 언어가 없다면 대화를 할 때, 우리는 보편 언어가 아닌 가장 작은 정보 조각인 데이터와 값으로 이야기를 해야 합니다.
✓ 이런 데이터를 보편 언어화 함으로써 개념을 보다 명시적으로 표현할 수 있습니다.

DB 테이블: CARGO_BOOKINGS

CargoId	VoyageId	LoadLocation	UnoadLocation
KR320-09234-123	VU-JR-MJQ	US1	UK5
…	…	…	…
…	…	…	…

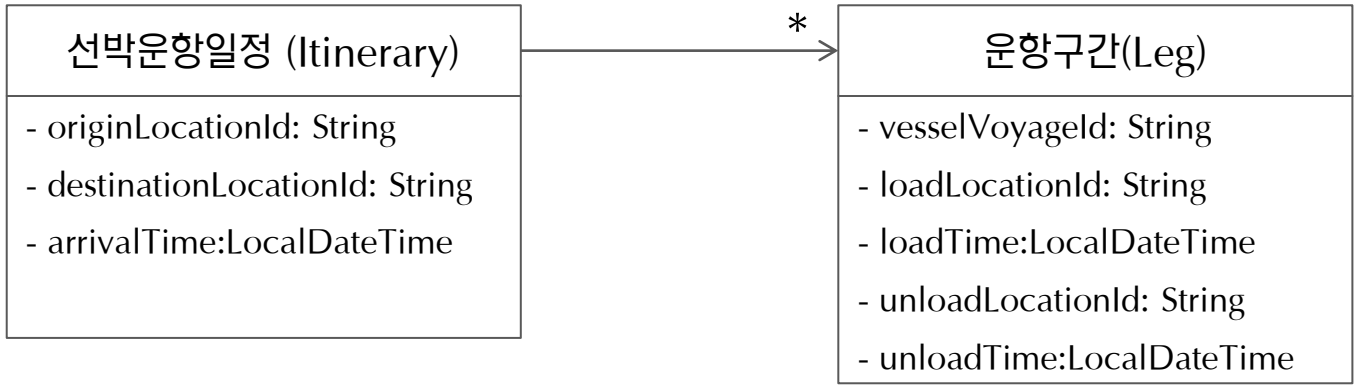
1.2 보편 언어로 표현

- ✓ 전문가는 “사람들한테 Cargo화물에 대한 전체 운항일정(Itinerary)이 필요할겁니다.”라는 식으로 이야기 합니다.
- ✓ 개발자는 관련 정보를 앞에서 보여 준 DB 테이블에 들어 있으므로, 각 DB에 들어있는 데이터 값으로 표현을 합니다.
- ✓ 전문가는 운항 구간 별로 날짜 정보를 궁금해 할 수도 있습니다. 현재는 테이블에 들어 있지 않은 상황입니다.
- ✓ 개발자는 어디서 날짜 정보를 가져와야 할 지 고민하고 있습니다.



1.3 정확성과 언어 확장

- ✓ 선박운항일정(Itinerary)이라는 개념을 보편 언어로 편입하고, 연관된 정보를 하나로 묶어줍니다.
- ✓ 운항구간(Leg)이라는 개념을 보편 언어로 편입하고, 기존 위치 정보에 시간 정보를 더함으로써 의미가 풍부한 개념을 만들어 낼 수 있습니다. 이제 개념의 확장은 추가 복잡성 없이 보편 언어 안에서 자연스럽게 이루어질 수 있습니다.
- ✓ 이제 개발자와 도메인 전문가 사이에 보다 정확한 의미를 가진 언어를 바탕으로 논의를 할 수 있게 되었습니다.



1.4 명시적인 표현(1/2)

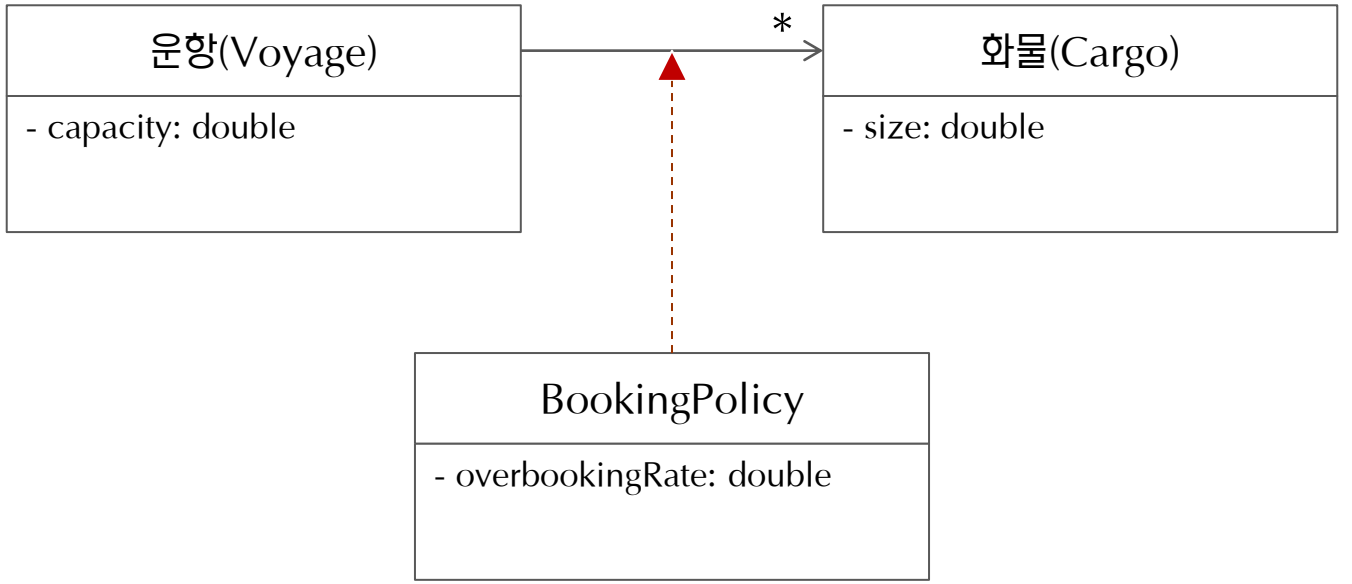
✓ 도메인 지식은 여러 가지 형태로 표현이 되고 전달되며, 특히 문장의 형태로 전달하고 사용하는 경우가 많습니다.
✓ 아래와 같은 예약 규칙을 개발자에게 전달하면, 개발자는 해당 내용을 프로그램 로직에 포함합니다.
✓ 다음과 같은 코드를 작성할 수 있습니다. → if(voyage.capacity > sum(cargo.size) * 1.1
✓ 여기서 1.1이 무엇을 의미하는 지 명시적으로 표현되어 있지 않습니다. 아래 문장을 봐야 알 수 있습니다.

운항 시 정규 적재량보다 10% 정도 초과하여 예약을 받는 것이 해운 업계의 관행입니다...

1.4 명시적인 표현(2/2)

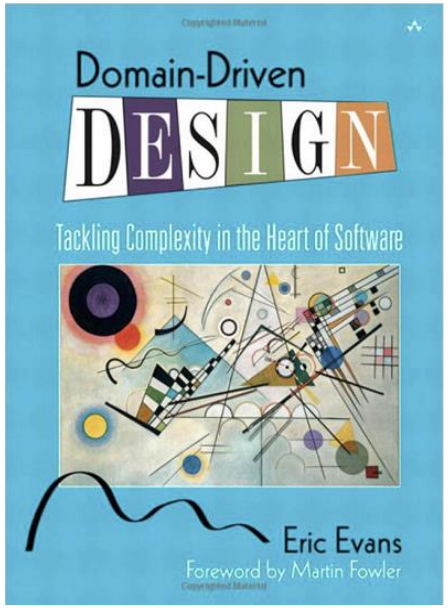
- ✓ 코드 속에 묻혀 있던 의미가 보편 언어화 되어 또렷하게 살아남으로써 의사소통을 명확하게 할 수 있습니다.
- ✓ 절차 지향 프로그램에서 부족한 부분 중에 하나가 바로 개념이 코드 속으로 녹아 들어가버리는 것입니다.
- ✓ BookingPolicy라는 언어를 이용하여 의사전달을 명확하게 할 수 있을 뿐만 아니라 향후 확장 가능성까지 갖게 되었습니다.
- ✓ 때로는 새로운 언어(개념)는 개별적으로 존재하는 여러 속성을 하나의 의미로 묶어 줍니다.

운항 시 정규 적재량보다 10% 정도 초과하여 예약을 받는 것이 해운 업계의 관행입니다...



2. Specification 패턴

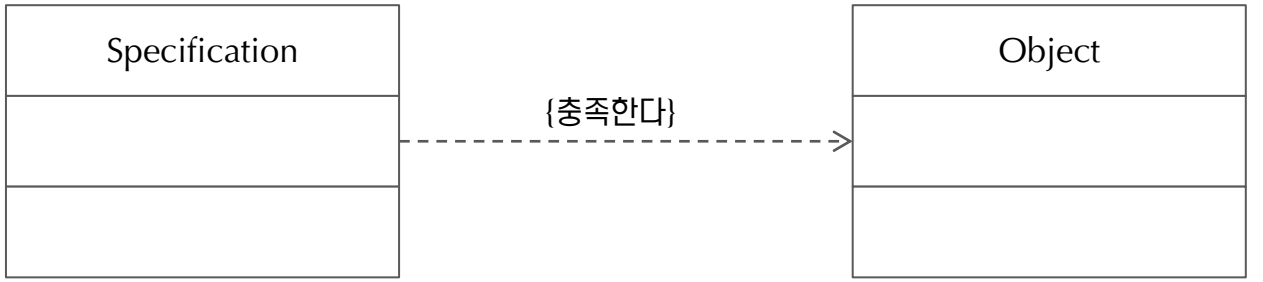
https://martinfowler.com/apsupp/spec.pdf



2.1 Specification의 필요성
2.2 Specification과 도메인 모델
2.3 Specification의 적용
2.4 논리 연산을 이용한 조합

2.1 Specification 필요성

- ✓ 모든 종류의 애플리케이션은 간단한 규칙을 검사하는 메소드가 존재합니다.
- ✓ 규칙이 단순하다면 hasNext()나 isOverdue() 같은 메소드를 사용해서 규칙을 처리할 수 있습니다.
- ✓ 대부분의 비즈니스 애플리케이션에서 처음에는 단순했던 규칙이 시간이 지나면서 점점 복잡해 집니다.
- ✓ 규칙이 복잡해지면 명확했던 도메인 객체가 비즈니스 규칙을 평가하는 코드에 묻혀 그 명확함이 사라져 버리게 됩니다.

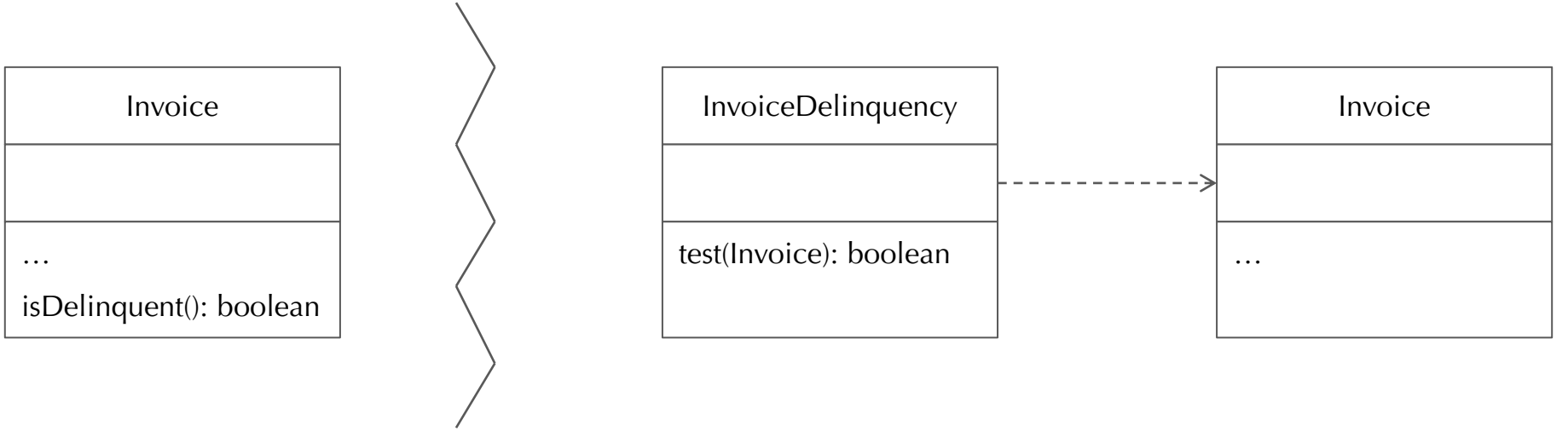


업무 규칙이 ENTITY나 VALUE OBJECT가 맡는 책임에 맞지 않고 규칙의 다양성과 조합이 도메인 객체의 기본 의미를 압도하는 경우가 있습니다.

업무 규칙을 도메인 계층으로부터 분리한다면 도메인 코드가 더는 모델을 표현할 수 없는 상황이 됩니다.

2.2 Specification과 도메인 모델

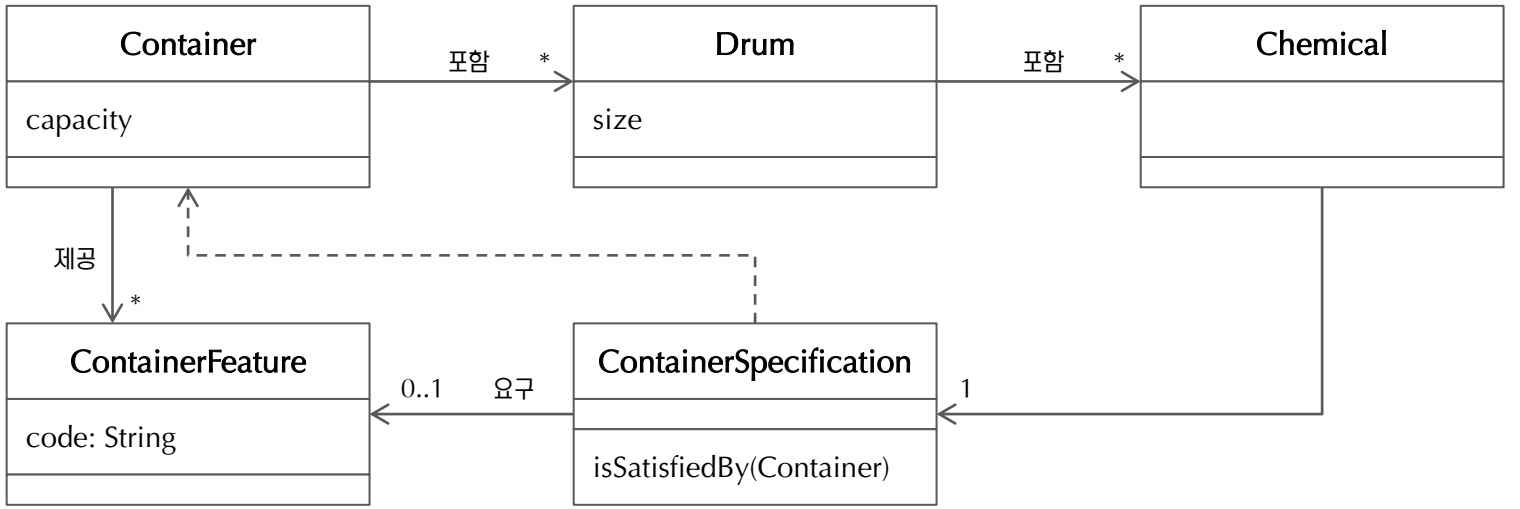
- ✓ 논리 프로그래밍에서 "규칙"은 true/false로 평가 되는 술어(Predicate)로 표현할 수 있습니다.
- ✓ 객체지향에서는 술어의 개념을 차용하여 Boolean 결과를 반환하는 특별한 객체를 만들 수 있습니다.
- ✓ 특별한 객체는 "명세"에 해당하며 다른 객체에 대한 제약조건을 기술하고 대상 객체가 명세를 만족하는지 검사합니다.
- ✓ 규칙이 복잡한 경우 논리 연산자를 사용해 술어를 결합(AND, OR, NOT)할 수 있습니다.
- ✓ Specification을 이용하면 명확한 도메인 객체를 유지하면서 비즈니스 규칙을 도메인 계층에서 유지할 수 있습니다.
- ✓ 또한 객체를 이용하여 규칙을 표현하므로, 설계에 도메인을 정확하게 반영할 수 있습니다.



2.3 Specification의 적용

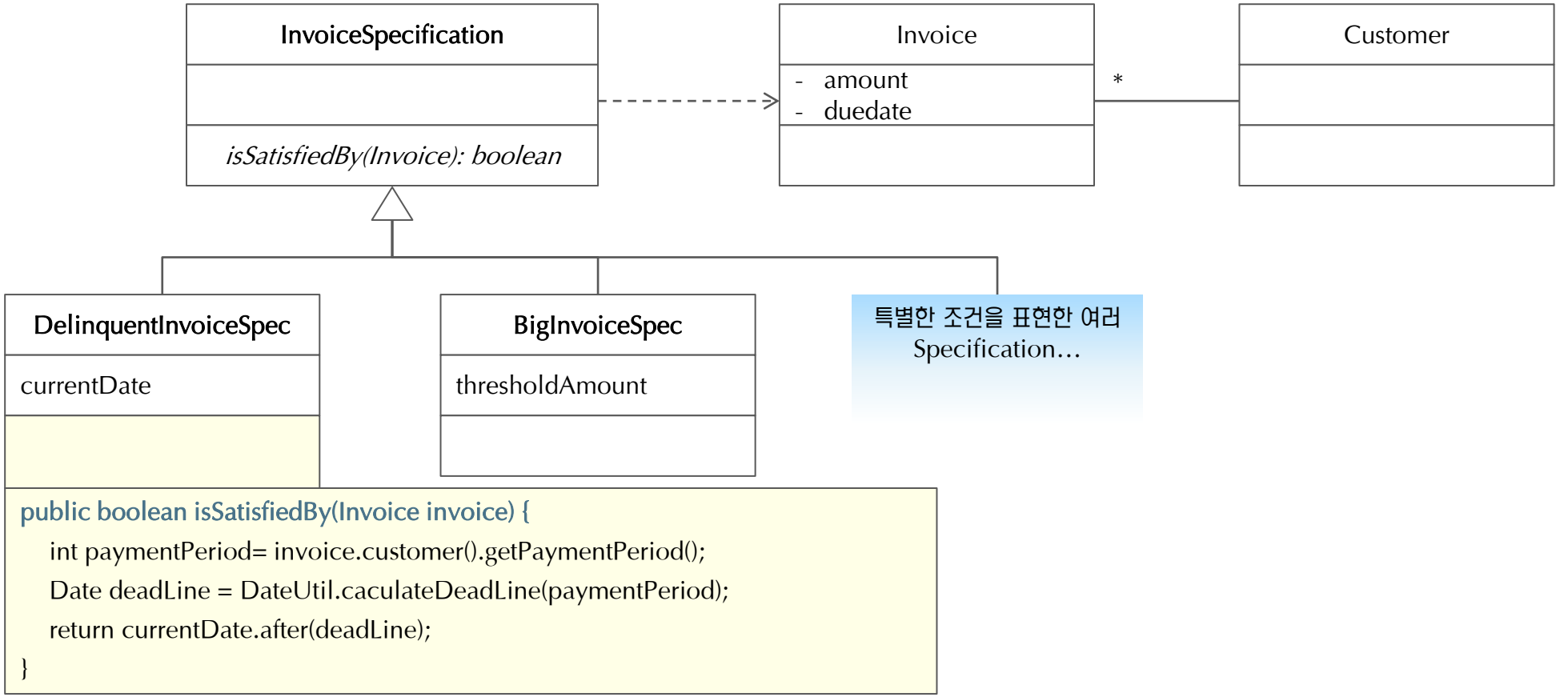
- ✓ Specification은 일반적으로 서로 다르다고 생각하는 애플리케이션의 여러 기능을 하나로 묶어줍니다.
- ✓ 대표적인 적용 범위는 다음과 같습니다. → 검증, 선택, 생성 기준 제시
- ✓ 이런 방식의 처리는 조건을 검증하기 위한 다양한 조건처리 코드를 객체로 캡슐화하여 줍니다.
- ✓ 뿐만 아니라 새로운 검증 조건을 처리해야 할 때도 기존 코드를 건드리지 않고 확장할 수 있습니다. → OCP

검증	객체가 어떤 요건을 충족하거나 특정 목적으로 사용할 수 있는지 판단할 목적으로 객체를 검증합니다.
선택	컬렉션 내에서 요건을 충족하는 객체를 선택(기한이 만료된 송장 목록)합니다.
객체 생성 기준 제시	특정한 요구사항을 만족하는 새로운 객체의 생성 조건을 명세합니다.



2.3 Specification의 적용(1/4) – 검증 1

✓ Specification의 가장 단순한 용도로 개념을 가장 잘 설명해 주는 방식입니다.
✓ DelinquentInvoiceSpec과 BigInvoiceSpec은 각각의 검증 목적을 가지는 명세입니다.
✓ 기존 코드에 영향을 주지 않고 새로운 유형의 명세서를 추가하는 방식으로 확장할 수 있습니다.



2.3 Specification의 적용(2/4) – 검증 2

- ✓ isSatisfiedBy 메소드는 컨테이너가 필요로 하는 Feature를 만족하는지 확인하도록 구현해야 합니다.
- ✓ 예를들어 폭발성 화학물질의 Specification은 "강화(ARMORED)"특성이 포함되어 있어야 합니다.
- ✓ Container 객체의 isSafelyPacked()는 보관된 Chemical의 모든 Feature가 Container에 포함되어 있는지를 확인합니다.

컨테이너의 특성을 검증하는 Specification

```
public class ContainerSpecification {

    private ContainerFeature requiredFeature;

    public ContainerSpecification(ContainerFeature required) {
        requiredFeature = required;
    }

    boolean isStisfiedBy(Container container) {
        return container.getFeatures().contains(requiredFeature);
    }

}
```

컨테이너애 보관된 Chemical의 모든 특성을 포함하는지 확인

```
boolean isSafelyPacked() {

    Iterator<Drum> it = contents.iterator();
    while (it.hasNext()) {
        Drum drum = it.next();
        if (!drum.containerSpecification().isSatisfiedBy(this)) {
            return false;
        }
    }
    return true;

}
```

폭발성 화학물질을 구성하는 클라이언트 코드

```
Chemical tnt = new Chemical();

ContainerSpecification armoredSpec =
                        new ContainerSpecification(ARMORED);
tnt.setContainerSpecification(armoredSpec);
```

재고에서 위험한 컨테이너를 확인

```
Iterator<Container> it = containers.iterator();
while (it.next()) {
    Container container = it.next();
    if (!container.isSafelyPacked()) {
        unsafeContainers.add(container);
    }
}
```

2.3 Specification의 적용(3/4) – 선택 1

- ✓ 특정한 조건을 기반으로 객체 컬렉션의 일부를 선택하는 것으로 다양한 구현 방식이 있습니다.
- ✓ 모든 Invoice가 메모리에 존재하는 경우와 데이터베이스에 존재하는 경우 동일 개념이 상이하게 구현됩니다.
- ✓ 메모리에 존재하는 경우 Specification을 이용하면 아래와 같이 구현할 수 있습니다.

InvoiceSpecification을 만족하는 Invoice 선택

```
public class InvoiceRepository {
    …
    public Set selectSatisfying(InvoiceSpecification spec) {
        Set<Invoice> results = new HashSeth<Invoice>();
        Iterator<Invoice> it = invoices.iterator();
        while (it.hasNext()) {
            Invoice candidate = it.next();
            if (spec.isSatisfiedBy(candidate)) {
                results.add(candidate);
            }
        }
        return results;
    }
}
```

InvoiceSpecification을 이용하는 클라이언트

```
Set delinquentInvoices = invoiceRepository.selectSatisfying(
                        new DelinquentInvoiceSpecification(currentDate));
```

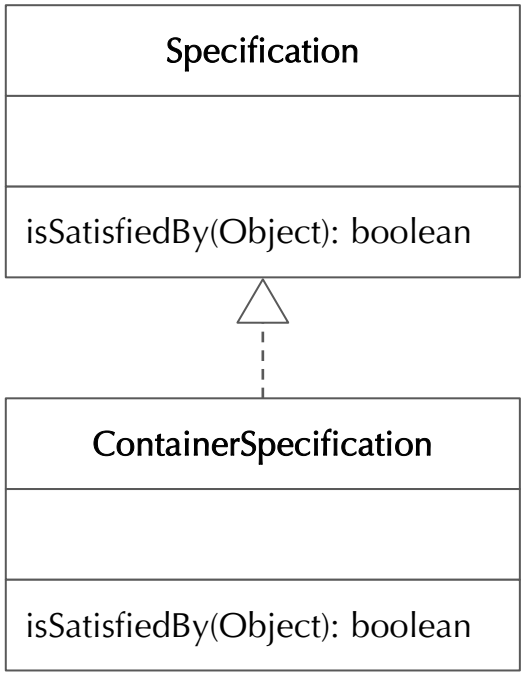
2.3 Specification의 적용(4/4) – 생성요청

✓ 앞서 본 검증, 선택과 달리 아직 존재하지 않는 객체를 생성하거나 재구성을 위한 기준을 명시하는데 사용합니다.
✓ Specification을 사용하지 않고 객체 생성을 위한 절차나 일련의 명령이 암시적으로 규정된 생성기를 작성할 수 있습니다.
✓ 생성 요청은 새로운 객체를 생성하는 것과 이미 존재하는 객체가 명세를 충족하도록 구성하는 것일 수 있습니다.

- ✓ 생성할 결과물에 대한 요구사항은 선언하지만 결과물을 생성하는 방법은 정의하지 않는다.
- ✓ 인터페이스는 생성 규칙을 명시하므로 연산의 세부적인 사항을 알지 않고도 결과물을 예상할 수 있다.
- ✓ 명세에 포함된 조건에 따라 객체를 생성하지만 요청은 클라이언트에 존재하므로 유연하게 개선할 수 있다.
- ✓ 생성기에 대한 입력을 정의하는 명시적인 방법이 모델에 표현되므로 테스트하기에 더 수월하다.

2.4 논리연산을 이용한 조합 (1/2)

- ✓ 사용중인 Specification 조합은 비즈니스 규칙을 손쉽게 확장할 수 있는 기회를 제공합니다.
- ✓ Specification은 상당히 일반화된 기능을 가지므로 확장할 수 있도록 추상 클래스나 인터페이스로 만드는 것이 유용합니다.
- ✓ 앞서 살펴본 ContainerSpecification은 다음과 같이 변경됩니다.




```
public interface Specification<T> {
    boolean isSatisfiedBy(T candidate):
}
```

```
public class ContainerSpecification implements Specification<Container> {
    private ContainerFeature requiredFeature;
    public ContainerSpecification(ContainerFeature required) {
        requiredFeature = required;
    }
    public boolean isSatisfiedBy(Container candidate) {
        return candidate.getFeatures().contains(requiredFeature);
    }
}
```

2.4 논리연산을 이용한 조합 (2/2)

✓ 컨테이너 예제에서 휘발성인 동시에 폭발성이 강한 화학 물질인 경우 두 가지 Spec이 모두 필요합니다.

✓ 새로 정의한 메소드(and, or, not)를 이용하면 이를 간단하게 조합할 수 있습니다.

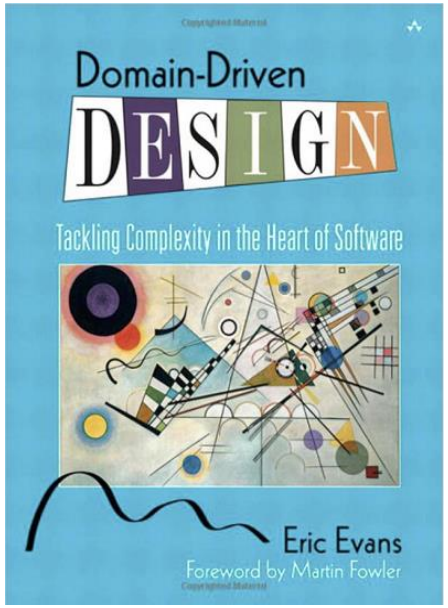
```
public abstract class AbstractSpecification implements Specification {
    public Specification and(Specification other) {
        return new AndSpecification(this, other);
    }
    public Specification or(Specification other) {
        return new OrSpecification(this, other);
    }
    public Specification not() {
        return newNotSpecification(this);
    }
}
```

```
public class AndSpecification<T> implements Specification<T> {
    Specification one, other;
    public AndSpecification(Specification x, Specification y) {
        one = x;
        other = y;
    }
    public boolean isSatisfiedBy(T candidate) {
        return one. isSatisfiedBy(candidate) &&
                other. isSatisfiedBy(candidate);
    }
}
```

```
public class OrSpecification<T> implements Specification<T> {
    Specification one, other;
    public OrSpecification(Specification x, Specification y) {
        one = x;
        other = y;
    }
    public boolean isSatisfiedBy(T candidate) {
        return one. isSatisfiedBy(candidate) ||
                other. isSatisfiedBy(candidate);
    }
}
```

```
public class NotSpecification<T> implements Specification<T> {
    Specification wrapped;
    public NotSpecification(Specification x) {
        wrapped = x;
    }
    public boolean isSatisfiedBy(T candidate) {
        return !wrapped. isSatisfiedBy(candidate);
    }
}
```

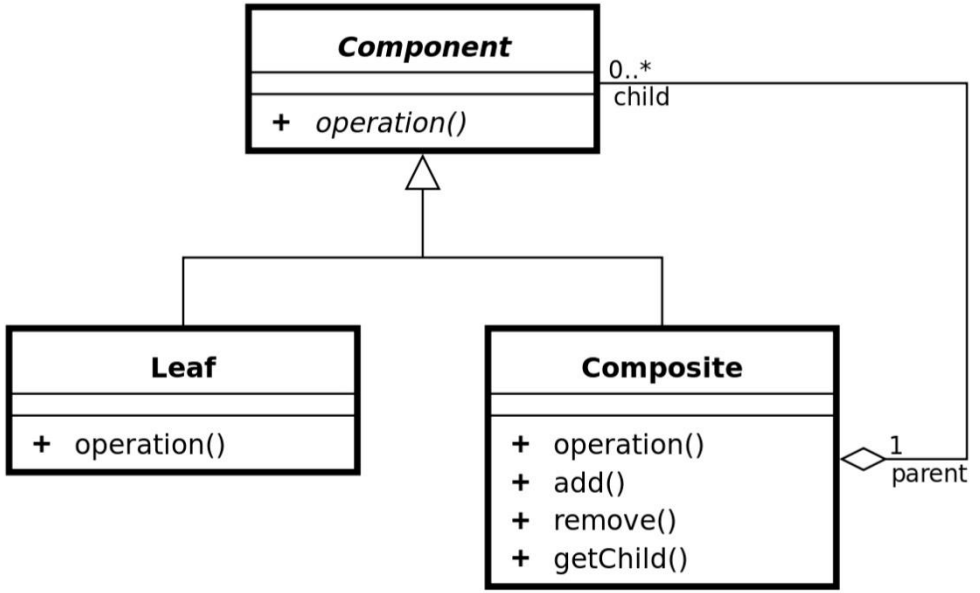
3. 모델과 디자인 패턴



3.1 Composite 패턴 개요
3.2 화물운송구간(Leg)
3.3 Route와 Leg
3.4 도메인 인식의 차이
3.5 도메인 모델

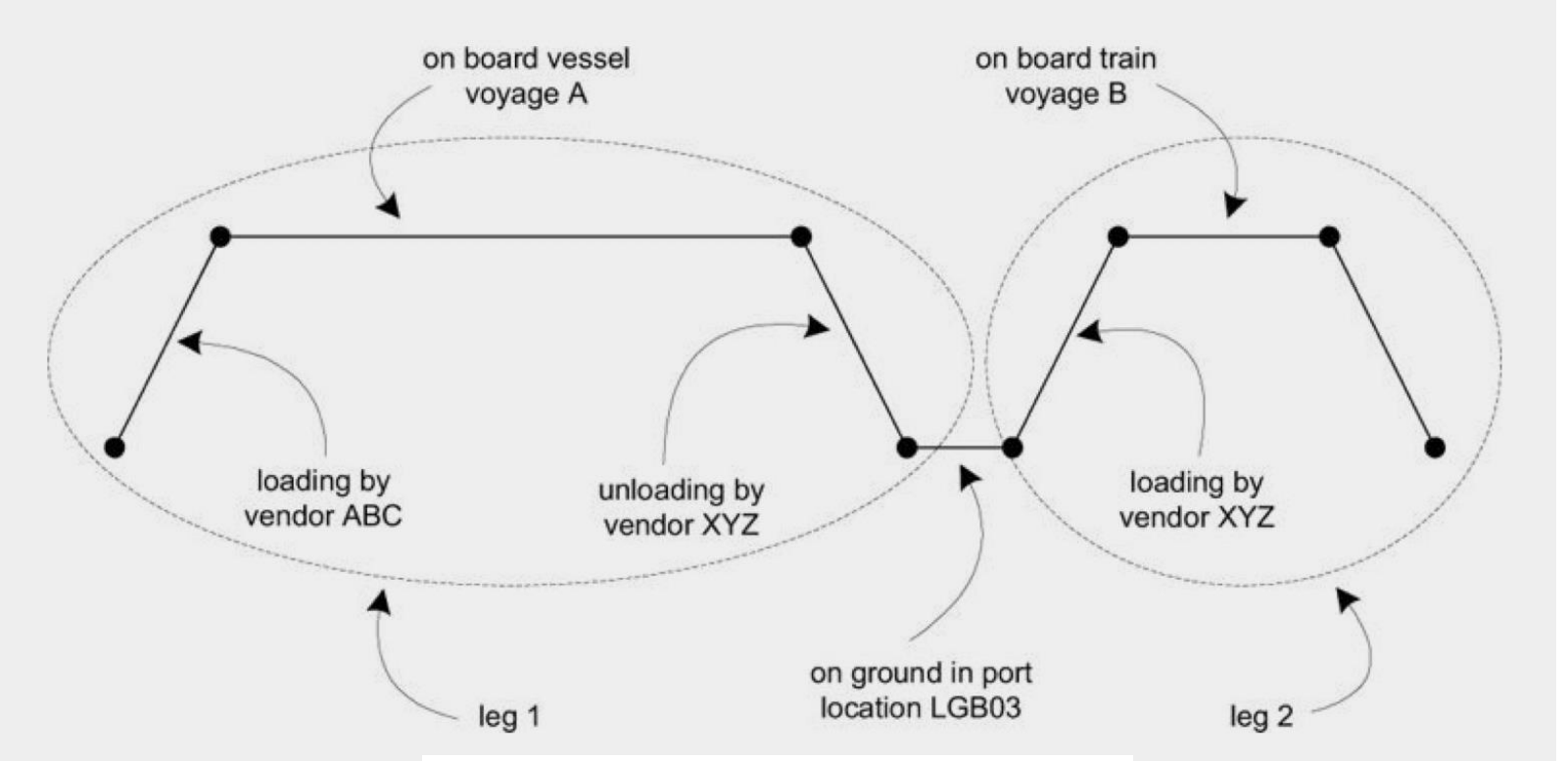
3.1Composite 패턴 – 개요

✓ 부분과 전체의 계층을 표현할 때, Composite 패턴을 사용합니다.
✓ 클라이언트는 부분과 전체를 구분하지 않고 추상화된 단일 일터페이스를 이용할 수 있습니다.
✓ 부분(leaf)에서 확장이 자유롭습니다. 이러한 특징 때문에 가장 많이 사용하는 패턴 중에 하나입니다.



3.2 화물 운송 구간(Leg)

- ✓ 화물수송 항로(route)는 매우 복잡합니다. 컨테이너를 철도에 싣기 위해 트럭으로 운반하고, 철도를 통해 항구로 옮기고, 항구에서 선박을 이용하여 다른 항구로 운송하고, 이런 절차를 반복하면서 최종 목적지에 도착합니다.
- ✓ 하나의 Leg는 loading → voyage → unloading의 연속으로 이루어집니다.

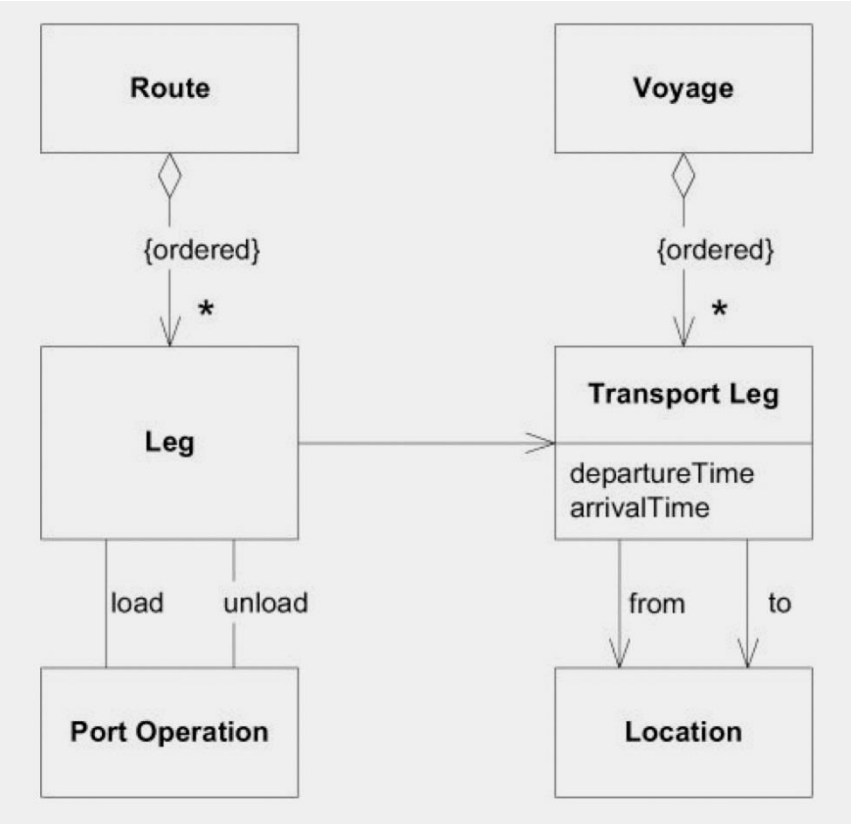


[이미지출처] Domain Driven Design, Eric Evans

3.3 Route와 Leg 모델

✓ 항로는 Route는 일련의 여러 운송 구간(Leg)로 구성됩니다.

✓ 각 운송 구간은 화물 싣기 → 운항(구간 이동) → 화물 내리기로 구성됩니다.

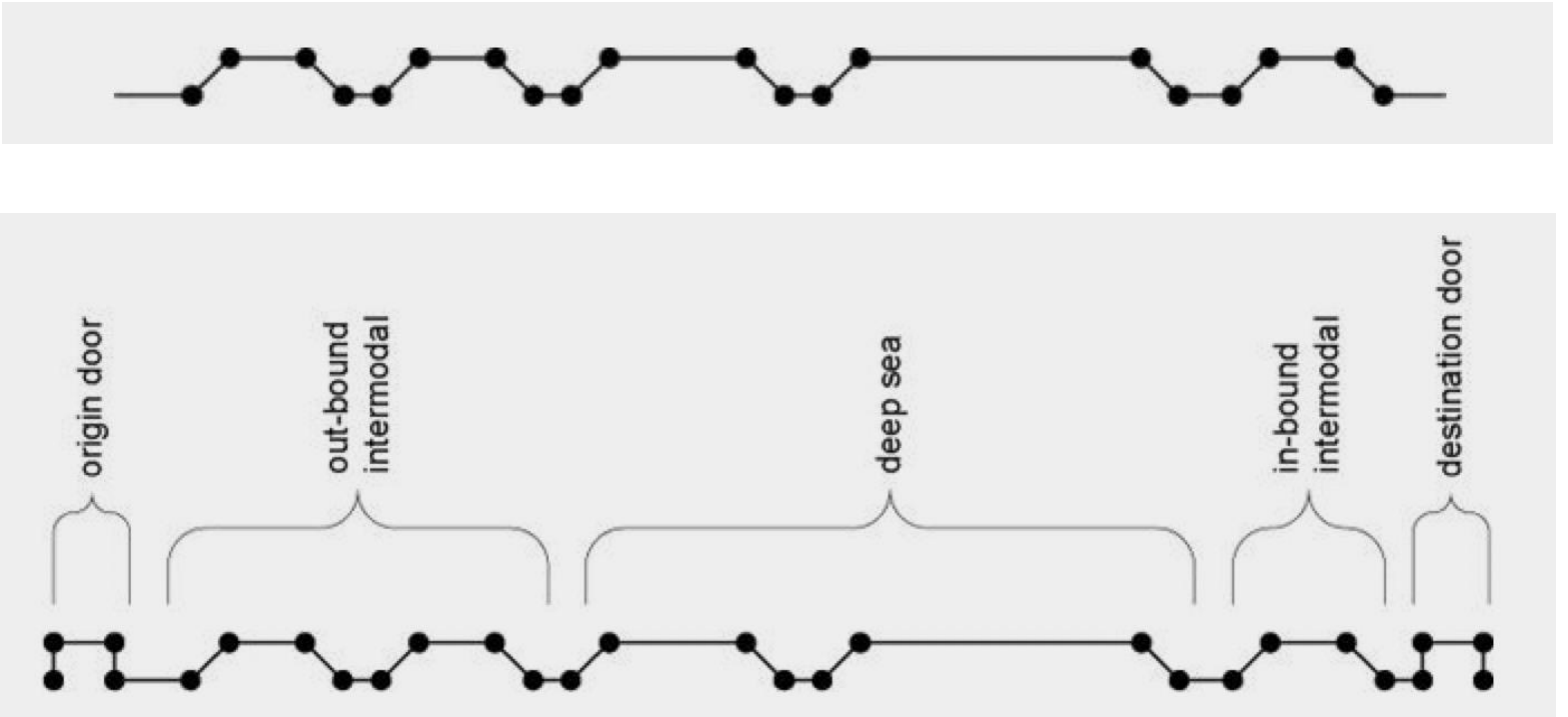


[이미지출처] Domain Driven Design, Eric Evans

3.4 도메인 인식의 차이

✓ 개발자는 항로를 동일한 유형의 Route가 반복적으로 진행되는 것으로 인식을 하였습니다.

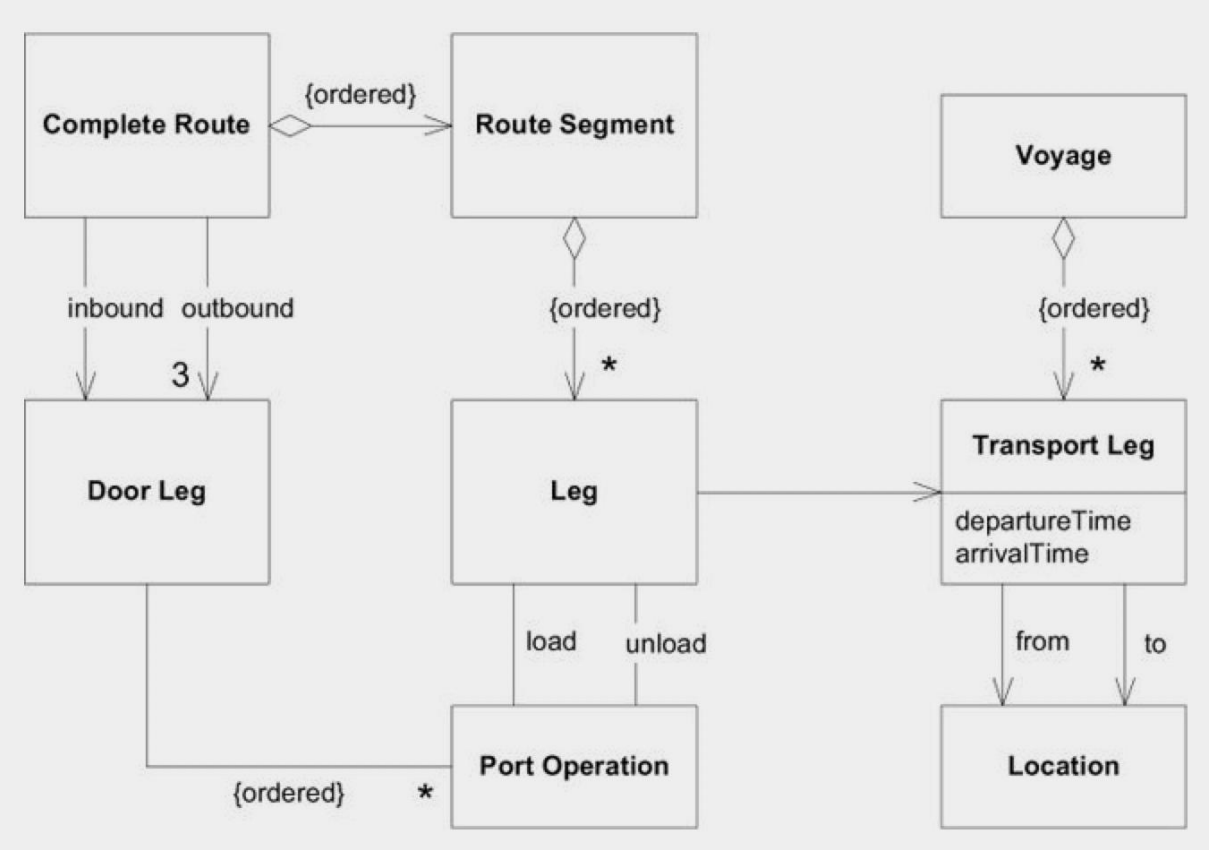
✓ 하지만, 도메인 전문가는 특수한 Route인 입하와 출하, 그리고 여러 개의 Route 조각들로 이루어진 것으로 보았습니다.



[이미지출처] Domain Driven Design, Eric Evans

3.5 도메인 모델(1/3) – 패턴 없음

✓ 전체 루트는 출하 → 여러 조각 루트 → 입하로 구성되어 있습니다.
✓ 입하(루트)와 출하(루트) 에서 운항구간을 특별히 DoorLeg라고 합니다.
✓ 현재 다이어그램에서는 서로 다른 유형의 루트(운항구간)을 별도로 처리하여야 하는 번거러움이 있습니다.

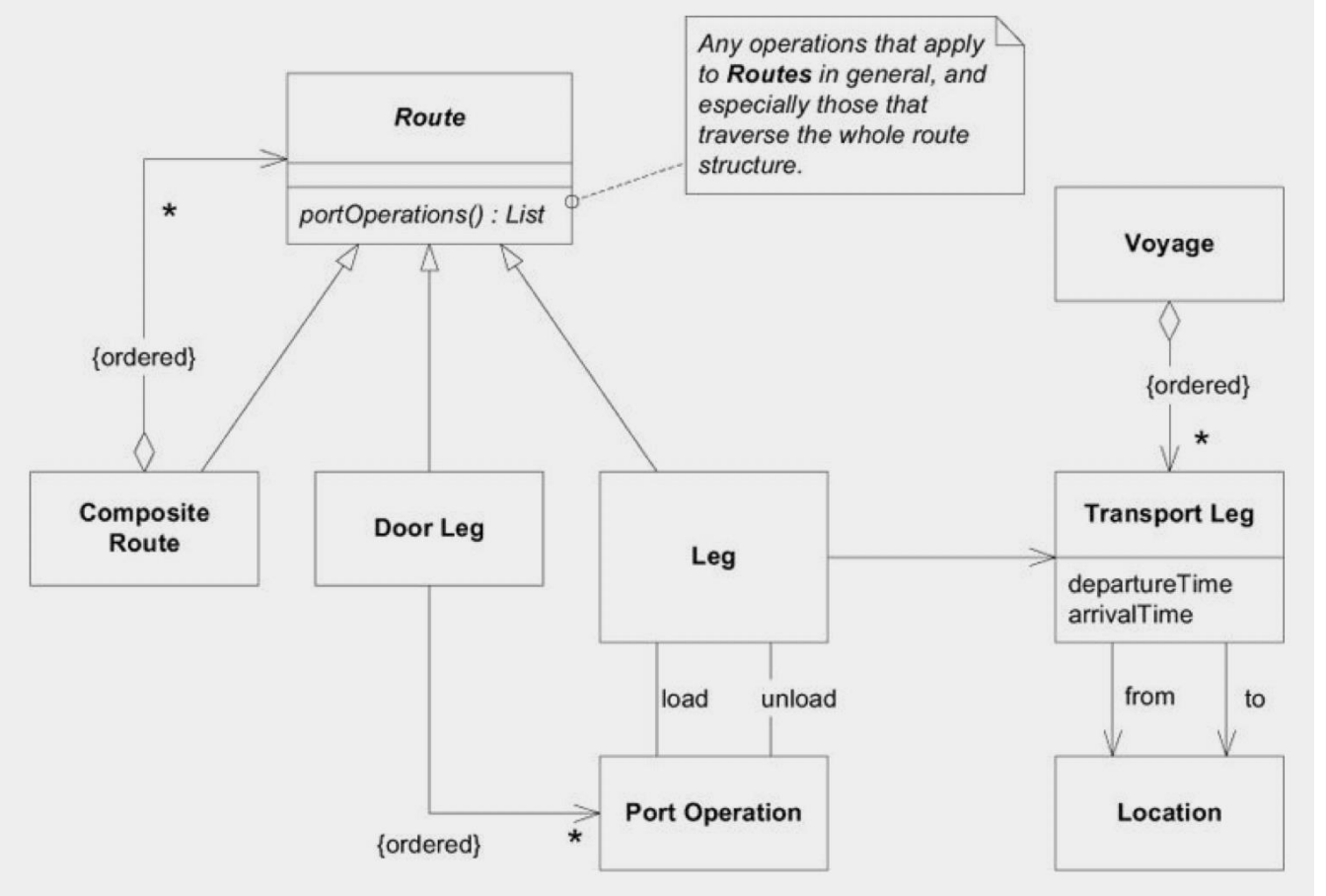


[이미지출처] Domain Driven Design, Eric Evans

3.5 도메인 모델(2/3) – Composite 패턴

✓ 복합루트는 여러 개의 서로 다른 유형의 루트(DoorLeg, Leg)을 갖고 있습니다.

✓ CompositeRoute는 시작과 끝에 DoorLeg 하나씩, 가운데 여러 개의 복합Route를 가질 수 있습니다.

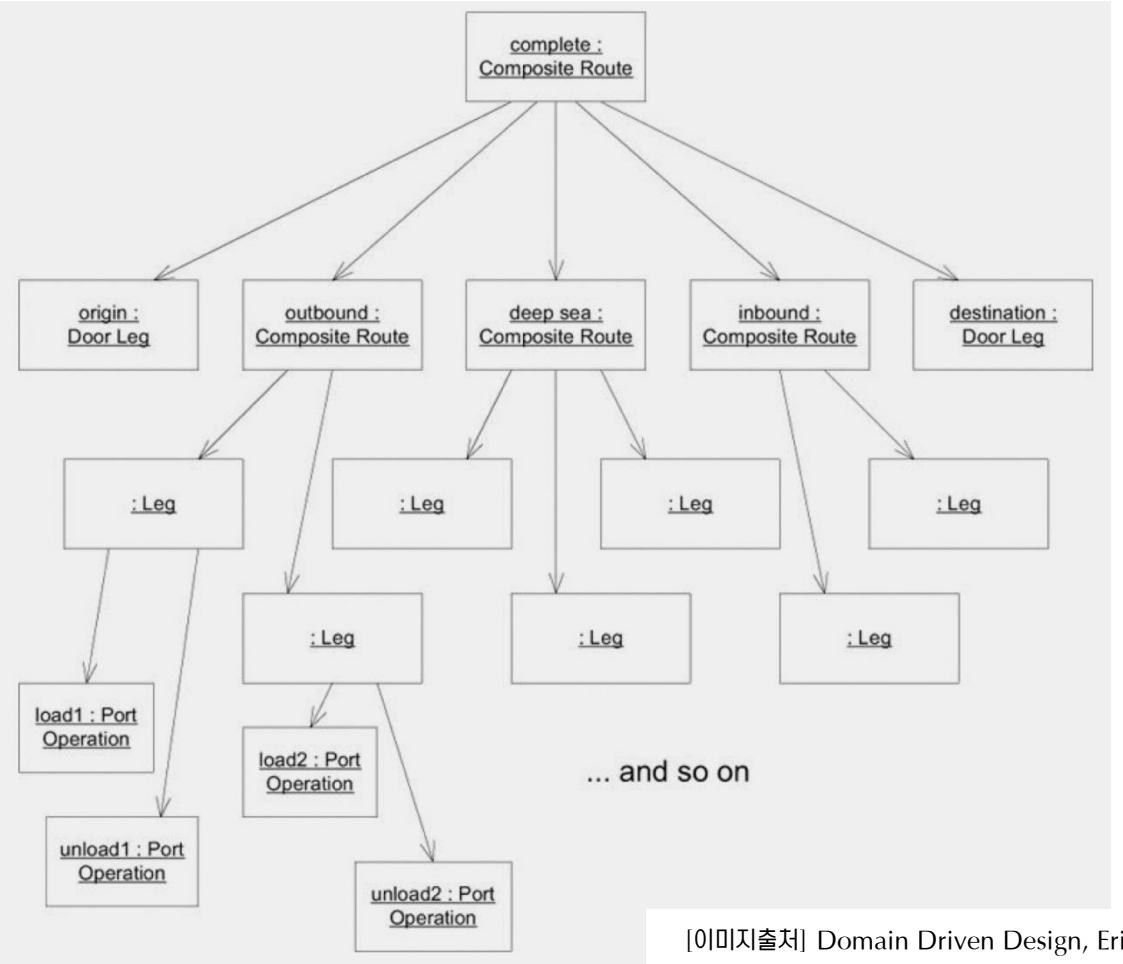


[이미지출처] Domain Driven Design, Eric Evans

3.5 도메인 모델(3/3) – 객체 다이어그램

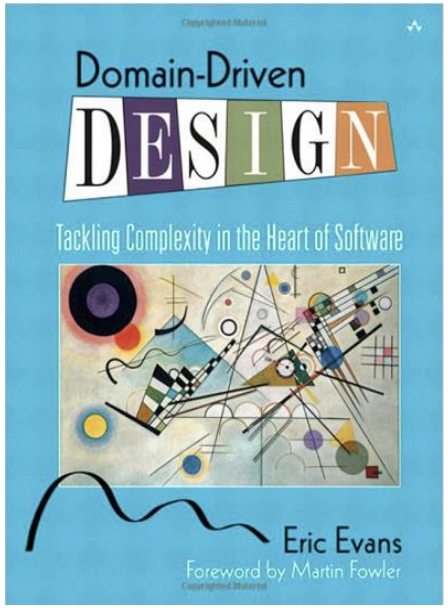
✓ 객체 다이어그램으로 앞의 클래스 다이어그램을 표현해보면 내포 관계의 Route를 이해할 수 있습니다.

✓ 복합루트가 여러 개의 복합 루트를 갖고 있음을 알 수 있습니다. 이것은 마치 폴더 아래 폴더 있는 것과 같습니다.



[이미지출처] Domain Driven Design, Eric Evans

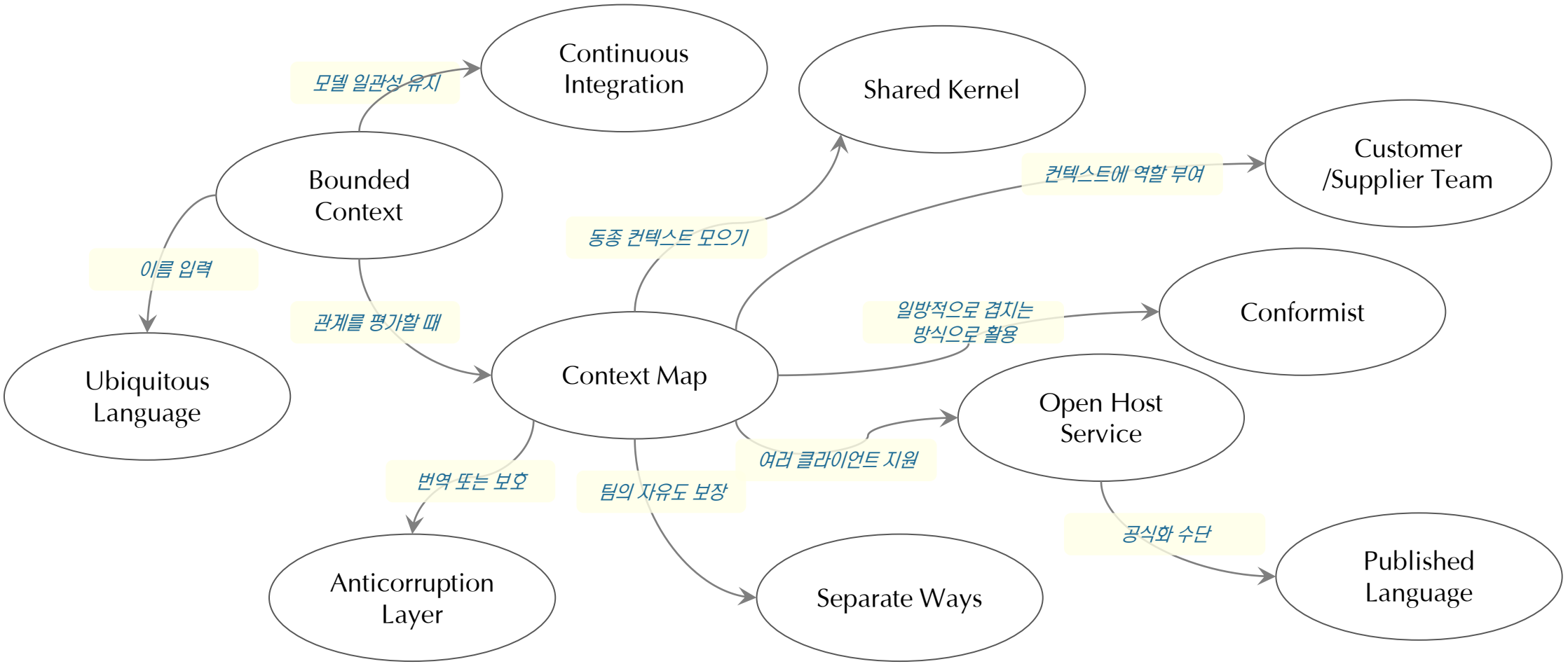
4. 전략적 설계



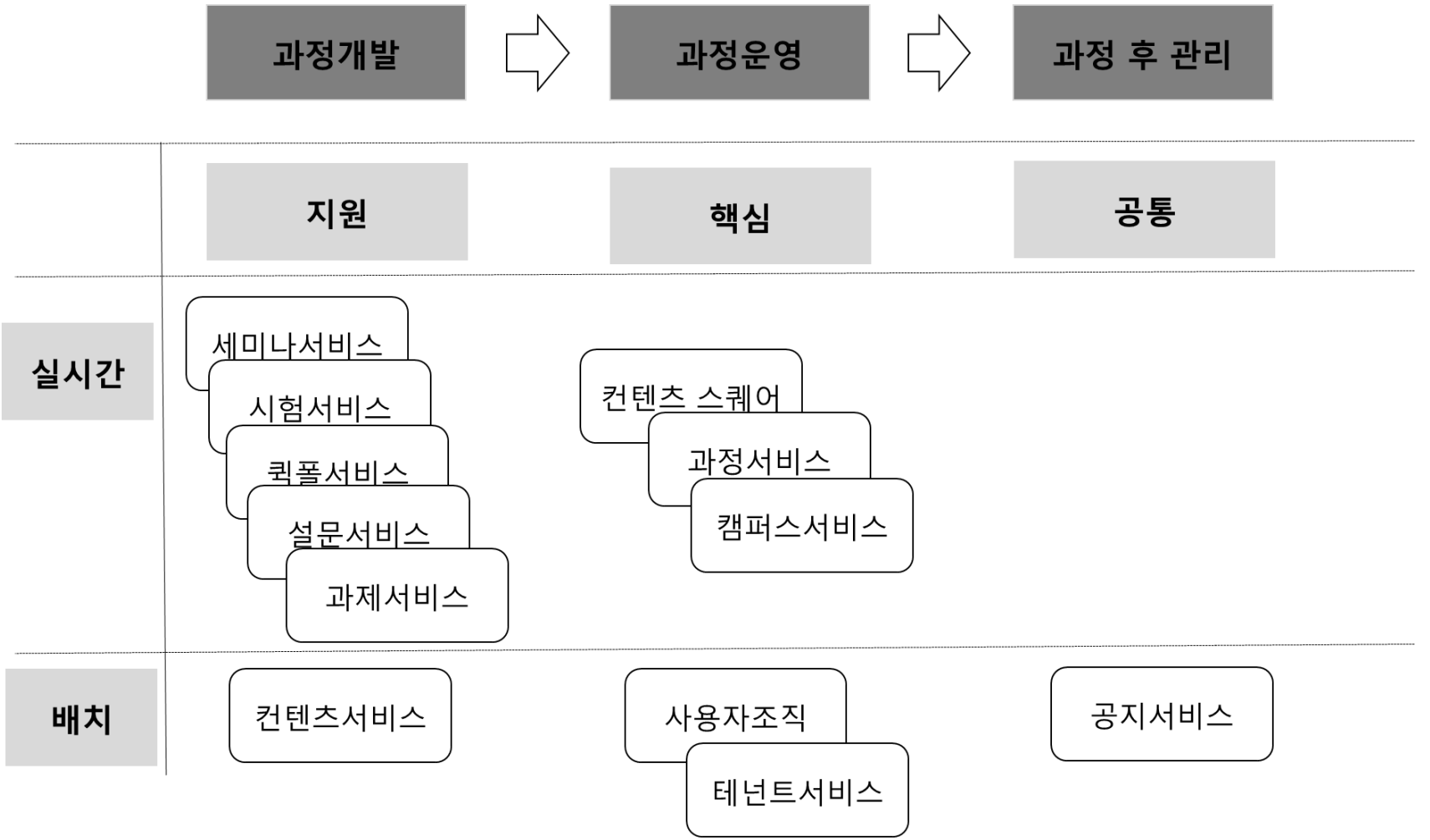
4.1 모델 무결성 패턴

4.1모델 무결성 패턴

✓ 아래 다이어그램은 모델 무결성 유지를 위한 패턴들의 맵을 보여줍니다.



4.2 사례연구 – 역량 개발 도메인: 개요



4.2 사례연구 – 역량 개발 도메인: 브레인스토밍

<서비스>

과정기획
강좌(플로우, 수강신청, 오픈, 종결)
강사의 강의지원(자료 공유, 과제, 개발환경 공유...)
강의분석
온라인 강좌
온라인 강좌 스토어
세미나
스터디클럽
직무로드맵/마이로드맵/역량진단

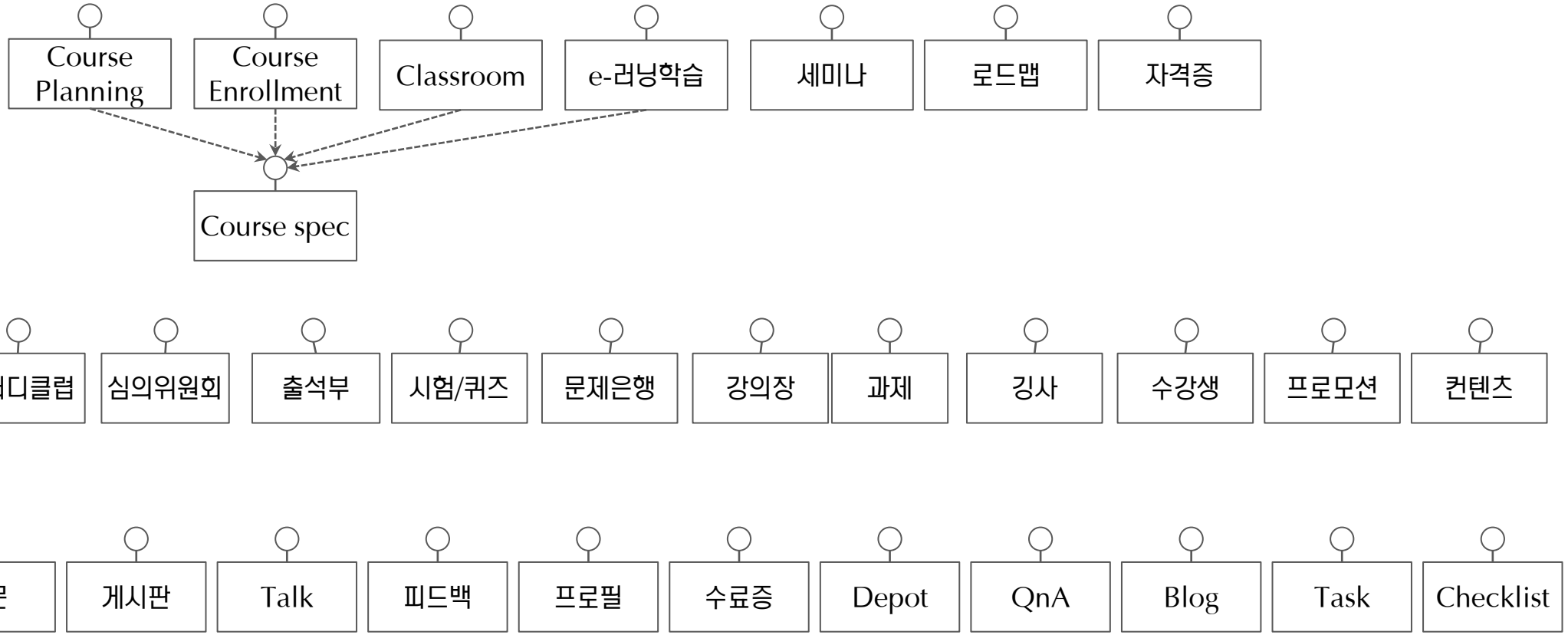
<지원 서비스>

플로우
심의위원회
출석부
설문
시험/문제은행
퀴즈
강의장예약
기숙사/시설 예약 (외부 연동)
강사
수강생
홍보
컨텐츠
결재
캠퍼스안내
자격정보

<공유서비스>

게시판
톡
피드백
프로필
수료증
파일/폴더
Q&A(with 보상)
Blog
Checklist
Task
Album

4.2 사례연구 – 역량 개발 도메인: 서비스



토론

✓Q&A

✓토론

감사합니다...

❖ 송태국 (tsong@nextree.co.kr)

❖ 넥스트리컨설팅(주) 대표 컨설턴트

❖ www.nextree.co.kr