# 분산 컴포넌트 기반의 소프트웨어 분석 및 설계 방법

## (An Approach to Software Analysis and Design based on Distributed Components)

최 유 희 † 염 근 혁 ††
(YouHee Choi) (Keunhyuk Yeom)

**요 약** 현재 새롭게 개발되는 소프트웨어는 50% 이상이 분산 플랫폼에서 개발되고 있다. 또한 분산 컴포넌트 기반의 소프트웨어 개발을 가능하게 하는 EJB(Enterprise Java Beans)[1], COM(Component Object Model)[2], CORBA(Common Object Request Broker Architecture)[3]와 같은 기술이 급격히 발전하고 있다. 따라서 분산 플랫폼 상에서 컴포넌트 기반의 응용 소프트웨어 개발을 위한 체계적인 개발 프로세스가 요구된다. 그러나 현재의 일반적인 컴포넌트 기반 소프트웨어 개발 방법론은 각 작업들간의 관계와 각 작업의 산출물간의 관계를 명확히 제시하지 않는다. 또한 분산 응용 소프트웨어 개발에 있어서 고려되어야 할 분산 이슈에 대해서도 체계적으로 다루지 않고 있다.

본 논문에서는 분산 컴포넌트 기반의 응용 소프트웨어 개발을 위한 분석 및 설계 방법을 제시한다. 본 논문에서 제시하는 방법에서는 UML 기반의 프로세스인 Unified process를 바탕으로 하여 체계적인 개발 지침과 산출물의 관계를 제시한다. 또한 플랫폼과 프로그래밍 언어에 독립적인 CORBA 환경을 고려하여 성능, 결함 방지, 안전성, 분산 트랜잭션의 분산 이슈를 명시적으로 다루는 지침을 제시한다.

***Abstract*** Recently, above 50 percentages of software are being developed based on distributed application platforms. And recent technologies such as EJB(Enterprise Java Beans)[1], COM(Component Object Model)[2], CORBA(Common Object Request Broker Architecture)[3] have been advanced for distributed component-based software development. Therefore, a systematic development process is necessary to develop component-based applications using distributed application platforms. However, most of component-based software development processes do not define concrete flows between tasks and relationships among artifacts of each task. Also, distribution issues are not considered explicitly in most of component-based software development.

In this paper, we present an approach to analyze and design software based on distributed components. In this approach, we propose systematic guidelines for developing a software based on Unified process and the relationships among artifacts which are produced. Also, we explicitly consider the distribution issues such as performance, fault tolerance, security, distributed transaction on CORBA environments.

## 1. 서 론

EJB[1], COM[2], CORBA[3] 등의 컴포넌트 아키텍처 기술의 발전과 인터넷의 급속한 보급으로 여러 플랫폼 상에 분산된 컴포넌트들을 설계, 구축 및 조립하는 컴포넌트 기반 개발 기술을 필요로 하게 되었다. 이에 따라 소프트웨어 개발에 있어서 컴포넌트 아키텍처를 바탕으로 분산 환경에서 수행될 수 있는 분산 컴포넌트로 구성된 소프트웨어를 지향하고 있다. 이러한 소프트웨어를 개발하기 위한 연구로 컴포넌트 기반 개발 방법에 대한 연구가 많이 이루어지고 있다. 대표적인 컴포넌트 기반 개발 방법 중 UML(Unified Modeling Language)[4]을 기반

· 이 논문은 2000년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2000-003-E00308).
† 정 회 원 : 한국전자통신연구원 컴소연 S/W공학연구부 연구원
yhchoi@etri.re.kr
†† 종신회원 : 부산대학교 컴퓨터공학과 교수
yeom@hyowon.pusan.ac.kr
논문접수 : 2000년 9월 14일
심사완료 : 2001년 10월 9일

으로 하는 Catalysis 방법[5]과 Unified software development process[6], CBD96[7,8]가 있다. 그러나 이러한 방법들에서는 소프트웨어 개발 프로세스의 각 작업에 대하여 체계적이면서 구체적인 지침들을 충분히 제시하지 않고 있다[9]. 또한 각 작업들간의 산출물이 다른 작업들에 어떤 영향을 미치는 지 등의 산출물간의 관계가 구체적으로 정의되어 있지 않다. 따라서 각 작업들의 산출물간의 관계에 대하여 일관성을 유지하기 어렵다.

또한 분산 컴포넌트 기반의 소프트웨어는 성능, 결함 방지(fault tolerance), 안전성(security), 분산 트랜잭션(distributed transaction)등에 대한 고려가 요구된다[10]. 그러나 이러한 고려 사항들이 현재 대부분의 컴포넌트 기반 소프트웨어 개발 방법론에 의해서 체계적으로 다루어지지 않고 있고 하위 레벨인 컴포넌트 아키텍쳐에서 다루어지는 고려 사항으로만 인식되고 있다. 한편, 컴포넌트 아키텍쳐는 단지 결함 방지, 안전성, 분산 트랜잭션 등에 대하여 하위 레벨에서의 서비스 제공의 측면을 다루는 것으로 실질적으로 분산 컴포넌트 기반의 소프트웨어의 개발 과정에서 각각을 설계 단계에서 어떻게 다룰 것인가에 대하여 고려하지 않는다. 따라서 설계 단계에서의 명시적인 고려가 없이는 분산 컴포넌트 기반의 소프트웨어의 실효성과 성능 향상 등의 이점을 기대할 수 없다.

본 논문에서는 분산 컴포넌트 기반의 소프트웨어를 분석 및 설계하는 방법을 제시한다. 이를 위하여 Unified software development process의 작업 구성을 바탕으로 하여 각 작업들간의 흐름과 각 작업의 산출물간의 관계를 명확히 제시한다. 또한 분산 컴포넌트 기반의 소프트웨어의 기본적인 요소인 컴포넌트 아키텍쳐를 고려하여 성능, 결함 방지, 안전성, 분산 트랜잭션 등을 명시적으로 다루는 분석 및 설계 방법을 제시한다. 다루고자 하는 컴포넌트 아키텍쳐는 현재의 대표적인 컴포넌트 아키텍쳐인 Sun의 EJB, COM, CORBA 중에서 EJB, COM에 비하여 다양한 프로그래밍 언어와 이기종 플랫폼을 지원하고 향상된 분산 하부 서비스를 지원하는 등의 많은 장점을 가지고 있는 CORBA를 기반으로 한다.

## 2. 관련 연구

### 2.1 분산 컴포넌트의 정의

분산 컴포넌트(distributed component)는 분산 객체(distributed object), 분산 소프트웨어 컴포넌트(distributed software component) 등의 용어들과 유사하게 사용되고 있다[11,12,13,14,15]. 여러 연구에서 사용되는 언급되는 분산 컴포넌트의 정의는 다음과 같다.

- ◆ 분산 컴포넌트 기반의 소프트웨어는 네트워크상의 별개의 노드에 독립적으로 분산 될 수 있는 컴포넌트들로 구성된 소프트웨어이다[11].
- ◆ 분산 소프트웨어 컴포넌트(분산 컴포넌트와 혼용하여 사용)는 공통적인 표현언어로 정의된 자신을 기술하는 인터페이스를 통해 분산 시스템에서 서비스의 집합을 제공하는 것이다[12].
- ◆ 분산 소프트웨어 컴포넌트는 제공하는 서비스를 기술하는 인터페이스를 통하여 하부의 네트워크상에서 다른 컴포넌트들과 통신하는 것이다[13].
- ◆ 분산 객체는 소프트웨어 컴포넌트의 정의를 만족하는 것으로 소프트웨어 컴포넌트는 논리적 또는 물리적으로 분산될 수 있는 것이다[14].
- ◆ 분산 객체의 구현은 네트워크에서 여러 사이트에 있는 서버들의 집합으로 서버들은 일치하는 서비스 인터페이스를 제공하여 서로 통신하고 상호작용할 수 있다[15].

이러한 정의를 바탕으로 본 논문에서는 분산 컴포넌트를 물리적으로는 네트워크 상에 독립적으로 분산될 수 있고 논리적으로는 독립적인 서비스 단위로서 서비스를 기술하는 인터페이스를 가지고 있는 컴포넌트로 정의한다.

### 2.2 컴포넌트 기반 개발 방법

#### 2.2.1 Catalysis 방법론

Catalysis는 UML을 사용하여 객체와 컴포넌트 기반 소프트웨어 개발을 지원하는 방법론으로 도메인/비즈니스 레벨, 컴포넌트 명세 레벨, 내부 설계 레벨의 세 레벨로 구성된다[5]. 도메인/비즈니스 레벨에서는 문제 도메인 용어를 설정하고, 비즈니스 프로세스를 이해한다. 컴포넌트 명세 레벨에서는 컴포넌트의 책임(responsibility), 컴포넌트와 시스템의 인터페이스를 정의하며, 컴포넌트의 오퍼레이션을 명세한다. 컴포넌트 구현 레벨에서는 내부 아키텍쳐를 정의하고, 시스템과 컴포넌트의 내부를 설계한다.

그러나 대부분의 현재의 컴포넌트 개발 방법론과 마찬가지로 추상적이고 일반적인 틀만을 제시하고 있기 때문에 실제로 소프트웨어를 개발하는 것과는 많은 차이가 있다. 또한 각 작업에 대한 입력과 출력 산출물이 대략적으로 정의되고 있어서 작업들간의 흐름이 구체적으로 정의되지 않았고 산출물간의 관계가 일치하지 않는다는 한계를 가지고 있다[16].

#### 2.2.2 Unified software development process

Unified software development process는 UML을 사용하여 컴포넌트 기반 소프트웨어를 개발하는 방법론

으로서 Inception 단계, Elaboration 단계, Construction 단계, Transition 단계의 크게 네 단계로 구성되어 있고 각 단계 내에는 Requirements, Analysis, Design, Implementation, Test의 다섯 개의 Core Workflows로 구성된다[6]. 각 Core Workflows를 수행하면서 UML diagram과 element로 구성된 모델을 생성한다.

그러나 Unified process의 한계는 5개의 core workflow를 거치면서 수행되는 여러 작업들을 통해 생성되는 각 모델을 구성하는 구성요소들에 대하여 정의는 하고 있지만 각각의 작업간의 관계와 각각의 작업에서 입력으로 제공되는 정보가 각 활동에 어떤 영향을 주고 그런 활동의 결과로 각 모델의 구성요소들이 어떻게 생성되는지에 대하여 체계적으로 제시하고 있지 않다. 이로 인하여 각 산출물들이 이전 산출물들과 일관성 없이 생성되게 되는 문제점이 있다. 또한 Unified process에서는 분산 컴포넌트와 관련된 고려 사항에 대해서 다루어야 한다고 하고 있지만 프로세스에서는 체계적으로 다루지 않고 있다.

2.2.3 CBD96

CBD96은 Sterling에서 제시한 컴포넌트 기반 개발 방법으로 Catalysis 방법을 단순화하고 체계화하고, 확장한 방법이다[7][8]. Sterling의 CBD 프로세스는 Requirements Definition, Analysis, Behavior Specification, Architecture 4단계로 이루어진다. Requirements Definition에서는 하나 이상의 use case diagram을 생성하여 개발될 어플리케이션 요구사항을 정의하고, business type diagram을 사용하여 상위 레벨 비즈니스 개체(high-level business entities)와 그들간의 관계를 표현한다. Analysis에서는 어플리케이션의 문제 영역을 분석한다. Behavior Specification에서 context modeling단계에서는 인터페이스를 인식하고, interface modeling단계에서는 인식된 인터페이스를 상세히 분석한다. 마지막으로, Component specification단계에서는 인터페이스를 컴포넌트로 할당하고 컴포넌트를 기술한다. Architecture에서는 컴포넌트와 그들간의 상호작용을 표현한다.

그러나 Sterling의 CBD 프로세스는 각 단계에서 생성되는 모델의 표현에 중점을 두고, 다른 방법들과 마찬가지로 그 모델의 구체적인 생성방법을 제시하고 있지 않고 분산 이슈를 체계적으로 고려하지 않고 있다.

### 2.3 Unified process를 이용한 연구

[17]에서는 분산 엔터프라이즈 시스템에 대하여 엔터프라이즈 소프트웨어 아키텍처를 모델링하기 위하여 UML이 어떻게 사용될 수 있는 가를 제시하였고 UML을 확장하고 정제하는 것이 필요함을 기술하였다. 아키텍처의 구조를 나타내기 위하여 사용되는 아키텍처의 형태에 대한 예로 3-계층 아키텍처를 package diagram과 component diagram, deployment diagram으로 나타낼 수 있음을 보여 주고 있고 아키텍처의 행위를 나타내기 위하여 use case model을 이용할 수 있음을 보여 주고 있다. 또한 분산과 동시성을 나타내기 위하여 UML에서 정의하는 스테레오 타입(copy, become)을 이용할 수 있지만 복잡하고 다양한 분산 이슈를 모델링하기에는 어렵다고 언급하고 있다. 결론적으로 [17]에서는 UML로 표현하는 것에 초점을 맞추었고 실제로 3-계층 아키텍처를 구성하는 컴포넌트들을 어떻게 추출할 것인가와 분산 이슈를 프로세스내에서 어떻게 다룰 것인가에 대한 아키텍처 모델링 프로세스는 제시하고 있지 않다. [18]에서는 IBM의 SCA(Supply Chain Analyzer)에 대한 클라이언트 서버, 웹이 가능한 아키텍처와 front-end 시스템인 eSCA 시스템을 개발하기 위하여 Unified process를 적용하여 추출한 eSCA use cases에 대하여 기술하고 클라이언트-서버, 웹이 가능한 아키텍처로 구성된 eSCA 아키텍처와 기능에 대하여 기술하고 있다. 그러나 eSCA use cases를 추출하기 위하여 Unified process를 부분적으로 적용하였고 eSCA application architecture를 형성하는 데에는 Unified process가 적용되지 않았다. 또한 추출된 eSCA use cases가 application architecture의 형성에 어떻게 영향을 주는지에 대하여 제시되지 않았다. [16]에서는 UML 기반의 컴포넌트 개발 방법(COMO)을 제시하기 위하여 UML과 Unified process를 컴포넌트 개발과 관련하여 확장한 것이다. 컴포넌트 개발을 위한 프로세스는 컴포넌트 기반 개발 방법의 도메인 공학 부분으로 Unified process를 기반으로 하였다. 먼저 도메인 분석을 하여 도메인 컴포넌트를 추출하고 컴포넌트 설계 단계에서는 컴포넌트의 인터페이스와 컴포넌트 명세를 정의하는 작업 등을 수행하는 컴포넌트 개발 방법을 제시하였다. 그러나 본 연구와 같은 어플리케이션 컴포넌트를 개발하는 방법은 제시하지 않았고 컴포넌트의 분산을 고려하지 않았다.

### 2.4 분산 컴포넌트 기반의 소프트웨어에 대한 연구

[12]에서는 컴포넌트 기반 소프트웨어 개발에 있어서 기존의 이질적인 컴포넌트들의 결합이 어려운 점을 해결하기 위하여 CORBA를 기반으로 분산 컴포넌트들을 쉽게 검색할 수 있도록 정보를 제공하는 분산 컴포넌트 모델과 분산 컴포넌트를 결합하기 위한 프로세스를 기술하고 있다. 그러나 분산 컴포넌트 기반 소프트웨어를 개발하기 위한 프로세스를 제시하지 않았다. [19]에서는

각 사용자 정보에 따라 다양한 언어적인 텍스트 처리 작업을 수행하기 위한 시스템에 분산 객체 기술을 적용한 case study를 기술하고 있다. 기능을 별개의 독립적인 분산 객체로 분리함으로써 분산 수행이 가능하도록 하였고 분산 컴포넌트 아키텍쳐인 CORBA를 사용하여 확장성과 scalability 문제를 해결하였다는 점을 case study를 통해 설명하였다. 그러나 각 기능을 어떻게 분산 객체로 분리하는지에 대한 개발 프로세스 상에서의 측면은 제시되지 않았고 CORBA 자체의 장점을 이용한 측면에 대한 연구였다.

## 3. 분산 컴포넌트 기반의 소프트웨어 분석 및 설계

본 논문에서 제시하는 분산 컴포넌트 기반의 소프트웨어 분석 및 설계 방법은 Unified software development process의 전체 작업 구성을 바탕으로 하여 요구 사항 추출 단계, 분석 단계, 설계 단계로 구성된다.

### 3.1 요구 사항 추출(Requirement Capture)

소프트웨어의 요구 사항을 추출하여 use case와 actor로 구성되는 기능 중심의 use case 모델을 생성한다. 요구 사항 추출 단계의 작업 구성은 그림 1과 같다. 그림 1에서 기능적 요구 사항을 중심으로 기능과 관련 있는 actor를 추출하고 기능에 해당되는 use case를 추출한다. 추출한 use case를 중심으로 actor를 분류하여 association을 형성하고 use case간의 관계를 일반화(generalization), 확장(extend), 포함(include)관계로 분류한다. 생성된 use case 모델의 각 use case에 대하여 statechart diagram, activity diagram, collaboration diagram을 이용하여 use case description을 표현한다.

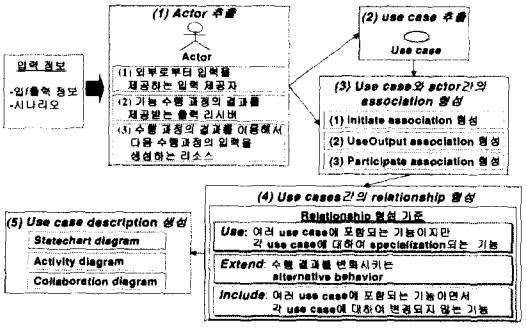


그림 1 요구 사항 추출 단계의 전체 작업 구성

3.1.1 actor 추출

Actor를 추출하기 위하여 어떠한 종류가 actor가 될 수 있는지에 대한 분류가 필요하다. 다음과 같이 actor를 세 종류로 분류할 수 있다.

① 시스템의 외부로부터 입력을 받아와서 시스템의 주요 기능을 수행하는데 사용될 수 있도록 하는 입력 제공자 또는 입력 장치

② 시스템의 주요 기능을 수행하는 과정에서의 결과 정보를 외부에서 제공받는 출력 리시버 또는 출력 장치

③ 시스템의 주요 기능 수행 과정에서의 정보를 이용해서 다음 수행 과정에서 사용될 수 있는 정보를 생성하는 리소스.

분류한 actor 정보에 따라 각 actor는 기능 수행 과정에서 외부로부터의 입력 정보 및 외부로의 출력 정보를 기술하는 입력 및 출력 정보 요구 사항으로부터 추출한다.

3.1.2 use case 추출

Use case를 추출하기 위하여 시스템의 기능을 추출하고, 추출한 기능들을 분류한다. Use case를 추출하는데 사용되는 정보는 use case의 단위를 고려하여 주요 기능, 각 actor와 관련된 기능 정보를 사용한다. 개발하고자 하는 시스템의 주요 기능은 시스템의 목적에서 추출하고 추출한 각 actor에 대해서 관련된 기능을 추출하기 위해 각 actors의 역할과 상호작용 정보로부터 추출한다.

추출한 정보를 바탕으로 use case가 될 수 있는 기능을 분류하는데 분류하는 기준으로는 추출한 기능에 대하여 적어도 하나의 actor와 관련이 있어야 하고 하나의 actor에만 관련된 여러 기능들에 대해 하나의 기능으로 통합될 수 있을 때 통합된 하나의 기능이 하나의 use case가 된다.

3.1.3 use case와 actor간의 association 형성

추출된 actor와 use case의 관계를 고려하여 각 use case에 대하여 actor를 세 종류로 분류하여 association을 형성한다. Actor는 그림 2에서처럼 세 종류의 actor로 분류할 수 있다.

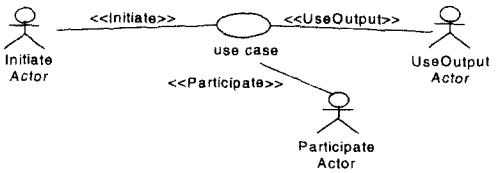


그림 2 use case와 actor간의 association

① use case의 수행을 시작하도록 하는 actor(Initiate actor): use case 기능수행과정의 처음 단계 입력을 제공하는 입력 제공자 actor가 해당되고 스테

레오타입<<Initiate>>로 association을 형성한다.

② use case의 수행에 대한 최종 결과를 사용하는 actor(UseOutput actor): use case 기능 수행과정의 마지막 단계 출력을 제공받는 출력 리시버 actor가 해당되고 스테레오타입<<UseOutput>>으로 association을 형성한다.

③ use case의 기능 수행에 참여하는 actor(Participate actor): use case 기능 수행결과를 생성하는 과정 중에 참여하는 입력 제공자, 출력 리시버, 리소스 actor가 해당되고 스테레오타입<<Participate>>로 association을 형성한다.

3.1.4 use cases간의 relationship 형성

Use cases간의 관계에는 일반화, 확장, 포함 관계가 존재한다. 각 관계를 다음과 같이 분류한다.

① 일반화 관계: 하나의 use case 수행과정상의 기능이 다른 use case 수행과정에서도 사용되지만 다르게 specialization될 수 있을 때 이러한 기능을 추상화하여 별개의 use case로 분리한다.

② 확장 관계: use case의 수행과정상의 기능 중에서 조건 만족여부에 따라 use case의 수행에 의한 기본적인 결과가 아닌 다른 결과를 actor에게 제공할 때 이러한 기능을 별개의 use case로 분리한다.

③ 포함 관계: 여러 use case의 수행과정에 동일하게 포함되는 기능으로 각 use case에 대하여 변경되지 않고 사용될 때 이러한 기능을 별개의 use case로 분리한다.

3.1.5 use case description 생성

각 use case에 대해서 기술하여야 하는 정보는 Unified software development process에서 분류하는 정보를 바탕으로 각 정보에 대하여 필요한 입력 정보를 그림 3과 같이 분류하였다.

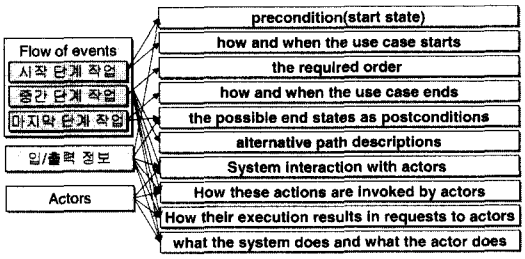


그림 3 use case description에 포함되는 정보 추출

각 정보는 그림 3에서처럼 use case에 대한 시나리오의 시작 단계 작업, 중간 단계 작업, 마지막 단계 작업과 입력, 출력 요구 사항, actors로부터 관련 정보를 추출한다. 이렇게 추출한 정보를 분류하여 statechart diagram, activity diagram, collaboration diagram을 사용하여 나타낸다. Statechart diagram은 use case 수행과정의 상태 변화 정보를 중심으로 기술한다. Activity diagram은 use case 수행과정의 활동에 대한 순서 정보를 중심으로 기술한다. Collaboration diagram은 use case와 actor간의 상호작용과 관련된 수행과정을 기술한다.

### 3.2 분석(Analysis)

요구사항으로부터 생성한 use case 모델을 이용하여 분석 모델을 생성한다. 분석 단계의 작업 구성은 그림 4와 같이 크게 3가지의 작업으로 구성된다. 그림 4에서 먼저 기능 중심의 독립적인 단위인 analysis package를 추출하고 추출된 analysis packages간의 의존(dependency) 관계를 형성한다. 또한 analysis package의 세부적인 구성요소인 analysis class를 추출한다.

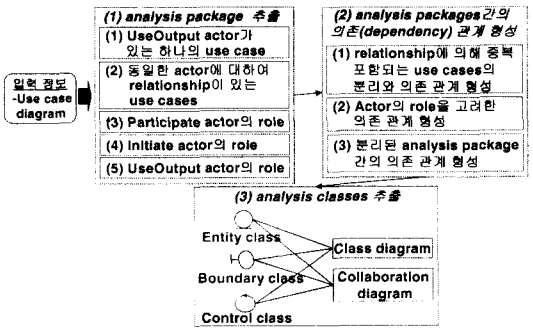


그림 4 분석 단계의 전체 작업 구성

3.2.1 analysis package 추출

Use case 모델의 use case와 actor를 바탕으로 서비스 제공과 상호독립성을 만족시키는 analysis package를 추출한다. analysis package를 추출하는 기준은 다음과 같다.

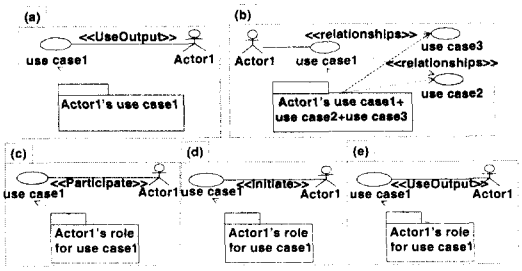


그림 5 analysis package 추출 기준

① 기본적으로 서비스 단위가 될 수 있는 것은 서비스를 수행함으로써 특정 actor가 원하는 결과를 얻고자 하는 것이다. 따라서 그림 5(a)에서처럼 use case 모델에서 하나의 use case1에 대해 수행 결과를 사용하고자 하는 Actor1이 있는 경우 Actor1에 대한 use case1이 analysis package로 추출된다.

② 특정 actor에게 서비스를 제공하기 위해서 여러 use cases가 관련이 있을 때 관련 있는 use cases가 모두 요구된다. 따라서 그림 5(b)에서처럼 Actor1에 대하여 relationship이 있는 use cases(use case1, use case2, use case3)가 analysis package로 추출된다.

③ 각 use case 수행과정상의 actor의 관점에 따라 각각이 서비스로 분류될 수 있다. 따라서 그림 5(c),(d),(e)에서처럼 Participate actor, Initiate actor, UseOutput actor 각각의 역할이 analysis package로 추출된다.

3.2.2 analysis packages간의 의존(dependency) 관계 형성

추출된 각각의 analysis package에 해당되는 use case를 바탕으로 analysis package의 관계를 형성한다. 중복되는 use case가 존재하는 경우에 따른 analysis package 분리와 analysis packages간의 의존 관계 설정은 그림 6과 같다. 그림 6에서 Actor1에 대하여 use case3과 use case5로부터 추출된 Analysis package1과 Actor2에 대하여 use case3과 use case4로부터 추출된 Analysis package2가 있을 때 use case3이 두 개의 analysis package(Analysis package1, Analysis package2)에 중복 포함되게 된다. 따라서 이러한 use case3을 Analysis package3으로 분리하고 Analysis package1과 Analysis package2에 대하여 의존 관계를 형성한다.

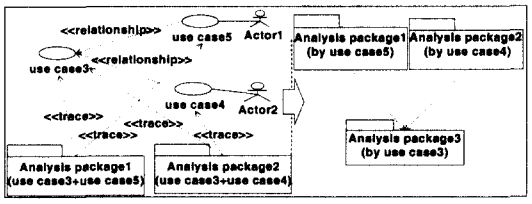


그림 6 analysis package 분리와 의존 관계 형성

또한 각 use case에 대하여 여러 actors와의 관계를 고려하여 다음과 같이 관련된 analysis packages간의 의존 관계를 설정한다.

① use case의 기능 수행에 참여하는 actor의 역할을 analysis package로 추출한 경우, actor의 역할은 use case의 수행에 사용되는 관계이므로 의 의존 관계가 설정된다. 그림 7에서처럼 Participate actor인 Actor2의 역할로부터 추출되는 Actor2's role과 use case1 for actor1간의 의존 관계가 설정된다.

② 각 use case에 대하여 수행을 시작하게 하는 actor와 수행 결과를 사용하고자 하는 actor의 역할을 analysis package로 추출한 경우, 각 actor의 역할은 use case가 제공하는 기능에 의존하는 관계이므로 의존 관계가 설정된다. 그림 7에서처럼 Initiate actor인 Actor1과 UseOutput actor인 Actor3으로부터 각각 추출된 Actor1's role과 Actor3's role이 use case1 for actor1에 대하여 의존 관계가 설정된다.

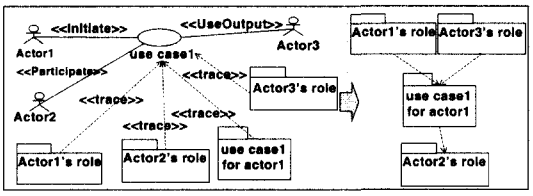


그림 7 각 actor와의 관계를 고려한 의존 관계 형성

또한 use cases간의 관계에 의하여 분리된 analysis package와 use case와 actor의 관계에 의해서 각 actor의 역할에 대하여 분리된 analysis packages간의 공통되는 부분이 존재할 수 있다. 공통되는 use case 수행 과정에 해당되는 analysis packages를 추출하고 그들간의 의존관계를 설정하여 class diagram을 형성한다.

3.2.3 analysis class 추출

분산 컴포넌트의 단위를 나누는데 기본적으로 사용될 수 있는 단위인 analysis class를 각 analysis package에 할당된 use case와 actor에 대하여 추출한다. Analysis class에는 지속적으로 유지되는 데이타를 모델링하는 entity class, 사용자와의 상호작용을 모델링하는 boundary class, 시스템의 기능을 모델링하는 control class가 있다. 각각의 analysis class를 추출하는 기준은 다음과 같다.

① entity class의 경우, use case description으로부터 각 actor와의 상호작용시 교환하는 정보와 use case를 수행하는 과정에서 내부적으로 사용되는

정보를 추출한다. 추출한 정보 중에서 여러 use cases에서 사용되는 정보와 지속적으로 유지될 필요가 있는 정보들이 entity class로 모델링된다.

② boundary class의 경우, 각 actor와의 상호작용 형태로부터 관련된 사용자 인터페이스 요소를 추출하고 추출된 요소들이 boundary class로 모델링된다.

③ control class의 경우, 추출된 boundary class와 관련하여 사용자 인터페이스와 관련된 기능을 추출한다. 또한 추출된 entity class를 이용하여 entity class에 해당되는 데이타와 관련된 기능을 추출하고, 시스템의 순수 비즈니스 로직에 해당되는 기능을 추출한다. 이렇게 추출한 기능들이 control class로 모델링된다.

Class diagram에 각 use case와 관련 있는 analysis class간의 association을 기술한다. 또한 collaboration diagram에 analysis objects간의 상호작용을 기술한다.

### 3.3 설계(Design)

기능적인 측면을 중심으로 생성한 분석 모델을 바탕으로 설계 모델을 생성하는 작업을 수행한다. 설계 단계의 작업은 그림 8과 같이 세 가지의 작업으로 구성된다.

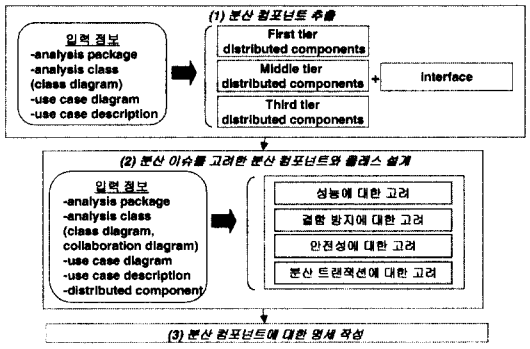


그림 8 설계 단계의 전체 작업 구성

그림 8에서 먼저 분산 컴포넌트를 생성하는 작업을 하고 분산 이슈를 고려하여 분산 컴포넌트와 클래스를 설계한다[20]. 그리고 분산 컴포넌트 명세를 작성한다.

3.3.1 분산 컴포넌트 추출

분산 컴포넌트는 독립적으로 분산될 수 있는 단위로 분산 컴포넌트의 단위는 각 분산 컴포넌트간의 상호작용이 최소화되도록 하여 의존관계를 줄일 수 있도록 형성되어야 한다. 그래서 각 analysis package의 analysis class와 대표적인 분산 구조의 형태인 3-계층 구조(3-tier architecture)를 바탕으로 분산 컴포넌트를 추출한다. 3-계층 구조의 첫 번째 계층은 사용자와의 상호작용을 다루는 계층이고, 두 번째 계층은 비즈니스 로직을 다루는 계층이며, 세 번째 계층은 지속적으로 유지되는 정보를 다루는 계층이다. 각 3-계층에 따라서 분산 컴포넌트를 추출하는 기준은 다음과 같다.

① 사용자와의 상호작용을 다루는 계층의 경우, 사용자와의 상호작용을 모델링하는 boundary class를 중심으로 분산 컴포넌트를 추출한다. Boundary class는 use case의 사용 목적 및 방식에 따라 달라진다. 가령 use case에 대하여 수행 결과를 사용하는 사용자와 입력을 제공하는 제공자가 각각 다를 경우 각 역할에 따라 사용자의 사용형태와 사용목적이 달라진다. 이에 따라 각 역할에 대하여 boundary class가 달라지고 사용자에 대한 boundary class와 입력 제공자에 대한 boundary class는 서로 직접적인 상호작용이 존재하지 않는다. 따라서 상호작용이 최소화되고 재사용될 수 있는 단위가 되도록 동일한 사용 목적 및 방식을 모델링하는 여러 boundary classes를 하나의 분산 컴포넌트로 추출한다.

② 비즈니스 로직을 다루는 계층의 경우, 비즈니스 로직을 모델링하는 control class를 중심으로 분산 컴포넌트를 추출한다. 상호작용을 최소화하고 독립적으로 분리될 수 있는 관계 측면에서 하나의 분산 컴포넌트로 추출될 수 있는 control classes간의 관계는 다음과 같다.

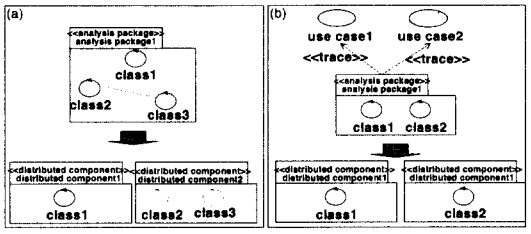


그림 9 두 번째 계층의 분산 컴포넌트 추출

◆ association관계가 있는 control class들은 상호작용이 존재하는 것이므로 association관계가 없는 control class를 분리하여 분산 컴포넌트로 추출한다. 그림 9(a)에서처럼 association 관계가 있는 class2와 class3이 하나의 분산 컴포넌트로 추출되고 class1이 하나의 분산 컴포넌트 단위로 추출된다.

◆ 동일 analysis package나 동일 use case와 관련 있는 control classes의 경우 해당 기능 수행을 위하여 서로간에 상호작용이 존재할 수 있다. 따라서 동일 analysis package 또는 동일 use case와 관련 있는 control classes를 하나의 분산 컴포넌트로 추출한다. 그림 9(b)에서처럼 use case1과 관련 있는 class1이 하나의 분산 컴포넌트로 추출되고 use case2와 관련 있는 class2가 하나의 분산 컴포넌트 단위로 추출된다.

◆ 수행 단계상에 조건 만족 여부에 따라 수행되는 control class가 다를 경우 조건 만족 여부에 따라 각각 다른 control classes의 기능이 수행되므로 서로 상호작용이 존재하지 않을 수 있다. 따라서 수행 단계상에서 조건 만족 여부에 따라 수행되는 control classes를 하나의 분산 컴포넌트로 추출한다. 그림 10(a)에서처럼 조건(condition)의 만족 여부에 따라 수행되는 class가 각각 class2와 class3으로 다를 경우 class2와 class3에 대하여 각각 분산 컴포넌트로 추출된다.

◆ control classes간의 관계가 aggregation관계에 해당되는 경우 'whole-part'관계에 해당되고 의미상으로 part에 해당되는 control classes는 각각이 독립적으로 존재할 수 있는 요소이다. 그래서 각 독립적인 부분(part)에 해당되는 control classes를 각각 분산 컴포넌트로 분리한다. 그림 10(b)에서처럼 aggregation 관계에 해당되는 class2와 class3에 대하여 class3이 하나의 분산 컴포넌트 단위로 추출된다.

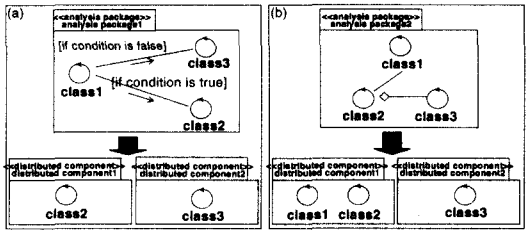


그림 10 두 번째 계층의 분산 컴포넌트 추출

③ 지속적으로 유지되는 정보를 다루는 계층의 경우, 지속적으로 유지되는 정보를 모델링하는 entity class를 중심으로 분산 컴포넌트로 추출한다. entity class와 control class의 상호작용을 고려했을 때, 동일한 control class와 관련 있는 여러 entity classes는 control class의 기능에 따라 분산될 수 있도록 하기 위하여 하나의 분산 컴포넌트로 추출한다.

추출된 분산 컴포넌트에 대하여 다른 분산 컴포넌트와 의존관계가 있는 analysis class를 각 분산 컴포넌트의 인터페이스로 형성하고 분산 컴포넌트간의 의존관계를 보여 주는 class diagram을 형성한다.

3.3.2 분산 이슈를 고려한 분산 컴포넌트와 클래스 설계

3-계층 구조를 바탕으로 추출된 분산 컴포넌트에 대해서 CORBA를 바탕으로 성능, 결함 방지, 안전성, 분산 트랜잭션 등의 분산 이슈를 고려하여 분산 컴포넌트와 클래스를 설계한다.

3.3.2.1 성능에 대한 고려

분산 컴포넌트 기반의 소프트웨어에서 성능에 영향을 미치는 것은 분산 컴포넌트간의 상호작용에 있어서 네트워크를 거치게 되는 원거리 호출에 따른 전송 지연과 마셜링 오버헤드라고 할 수 있다[21,22,23]. 따라서 성능 향상을 위해서는 원거리 호출을 줄이기 위한 설계가 요구된다. 원거리 호출을 줄이기 위한 접근 방법으로 분산 컴포넌트간의 상호작용을 고려하여 두 가지 측면으로 나눌 수 있다.

① 분산 컴포넌트를 재구조화하는 측면: 그림 11에서처럼 analysis class의 class diagram과 collaboration diagram을 이용해서 수행과정상에서 다른 분산 컴포넌트에 속하는 analysis class에 동기적인 호출 관계를 가진 원거리 호출을 추출한다. 그러한 관계의 경우 더 많은 오버헤드가 존재하기 때문에 해당되는 analysis class를 각 analysis class가 속한 분산 컴포넌트 중 하나의 분산 컴포넌트에 옮긴다. 옮기는 analysis class를 선택하기 위해서는 각각이 속해 있는 분산 컴포넌트 내부의 다른 analysis classes간의 관계에서 다른 분산 컴

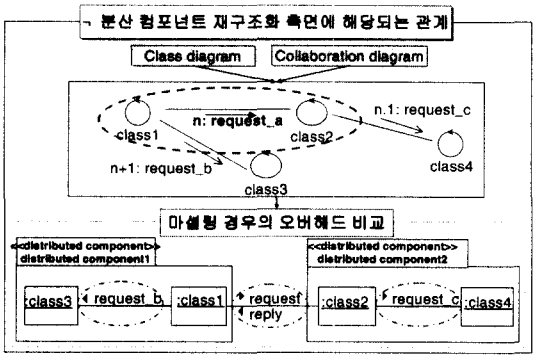


그림 11 분산 컴포넌트 재구조화 측면에 해당되는 관계

포넌트로 이동되었을 때를 가정하여 마셜링 오버헤드를 비교한다. 마셜링 오버헤드를 비교하는 방법은 관련된 오퍼레이션 호출에서 사용되는 매개변수와 결과값의 타입으로부터 CORBA 환경 상에서의 상대적인 마셜링 오버헤드[21]를 구하고 비교한다. 그림 11에서 class1과 class2의 관계, class2와 class4의 관계, class1과 class3의 관계에서의 호출에 대한 마셜링 오버헤드를 구하고 비교한다. 이를 바탕으로 마셜링 오버헤드가 상대적으로 작은 analysis class를 옮긴다.

② 분산 컴포넌트의 인터페이스를 설계하는 측면: 각 분산 컴포넌트에 존재하는 클래스들간의 호출 관계가 존재할 경우 인터페이스를 어떻게 정의하느냐에 따라서 원거리 호출의 수가 달라질 수 있다. 그림 12(a)에서처럼 다른 분산 컴포넌트에 속한 class의 여러 오퍼레이션(op1(), op2())이 하나의 기능 호출관계에 의한 것일 경우를 추출한다. 이러한 경우에 그림 12(b)에서처럼 여러 오퍼레이션(op1(),op2())을 포함할 수 있는 하나의 오퍼레이션(Wrapping_op()) 형태로 다른 분산 컴포넌트와 관계가 있는 인터페이스를 정의한다.

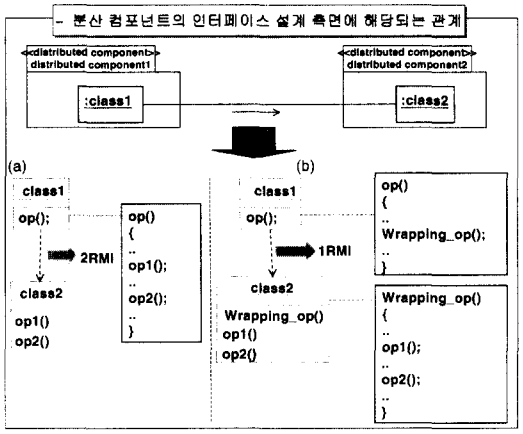


그림 12 분산 컴포넌트의 인터페이스 설계 측면

3.3.2.2 결함 방지에 대한 고려

결함 방지를 다루기 위한 기본적인 요소로는 결함 탐지(fault detection)와 결함 수정(fault correction)이 해당될 수 있다[24]. 따라서 결함 방지를 다루기 위해서는 먼저 기본적인 요소인 결함 탐지에 대한 고려가 되어야 한다. 결함 탐지에 있어서 어떤 단위에서의 결함인지에 따라서 결함의 종류와 결함 탐지 로직의 단위가 달라질 수 있다. 또한 결함 탐지에 의하여 탐지된 결함을 수정하기 위한 방법인 복제(replication)의 타입[25]도 어떤 종류의 결함인지에 따라 결정될 수 있다. 결함 탐지의 레벨을 분류하면 다음과 같이 호스트(host) 레벨과 분산 컴포넌트 레벨로 나눌 수 있다. 각 레벨에 따라 결함의 타입과 복제의 타입을 분류하면 다음과 같다.

① 호스트 레벨: 주요 결함은 호스트 자체의 결함과 네트워크를 거치는 원거리 호출에서의 결함이다. 이러한 종류의 결함은 수행을 중단시키는 결함인 fail-stop 타입[26]에 해당된다. 이러한 결함을 방지하기 위해서는 복구를 위한 적어도 하나의 복제 요소만 올바르게 동작하면 되므로 복제의 타입으로 passive 타입이 해당될 수 있다.

② 분산 컴포넌트 레벨: 주요 결함은 분산 컴포넌트에 포함되는 클래스의 오퍼레이션 자체의 의미적인 결함, 분산 컴포넌트에 포함되는 여러 클래스간의 오퍼레이션 수행상의 의미적인 결함과 분산 컴포넌트간의 오퍼레이션 수행상의 의미적인 결함이다. 이러한 종류의 결함은 수행을 중단시키지는 않으나 잘못된 수행을 하는 byzantine 타입[26]에 해당된다. 이러한 결함을 방지하기 위해서는 여러 복제 요소 중에서 대다수의 복제 요소의 수행이 동일한 값을 내는지를 비교하여야 하므로 복제의 타입으로 active 타입이 해당될 수 있다.

또한 3-계층 구조와 데이타 타입에 따라 결함 발생시 동기화가 필요한 상태가 달라질 수 있다. 따라서 동기화가 필요한 상태를 분류하면 다음과 같다.

① 첫 번째 계층과 두 번째 계층의 경우, 다른 분산 컴포넌트의 기능 수행에 영향을 주는 데이타의 변경 또는 생성 상태

② 세 번째 계층의 경우, 저장된 데이타에 대한 변경 상태

결함 방지에 대한 표현은 tagged value '{replication =' '}'를 이용하여 결함 방지가 필요한 레벨에 대하여 해당되는 복제 타입을 명시한다.

3.3.2.3 안전성에 대한 고려

안전성을 다루기 위하여 CORBA 환경에서 설계시에 고려하여야 할 사항은 어떤 정보에 대해서 안전성을 다루고 어떤 원칙을 적용할 것인지에 대한 것으로 3-계층 구조에 따라 다루는 데이타의 타입과 안전성 정책[27, 28]을 분류하면 다음과 같다.

① 첫번째 계층의 경우, 사용자의 입력과 출력 정보를 다루게 되므로 사용자의 입력과 출력의 초기값이 네트워크상에서 변경되지 않도록 하는 무결 보호

(integrity protection) 정책이 사용될 수 있다. 또한 사용자와 상호작용하기 전에 사용자가 누구인지를 증명하는 인증(authentication) 정책이 사용될 수 있다.

② 두번째 계층의 경우, 분산 컴포넌트간의 네트워크를 통한 상호작용에 대한 기밀성 보호(confidentiality protection) 정책이 사용될 수 있다. 또한 사용자가 여러 분산 컴포넌트를 이용하여 수행되는 기능을 사용할 때 사용자의 권한이 각 분산 컴포넌트에 전달될 수 있도록 하는 권한 위임(delegation of privilege) 정책이 사용될 수 있다.

③ 세번째 계층의 경우, 데이타베이스에 접근 권한을 다루는 접근 원칙(access policy)과 권한 부여(authorization) 정책이 사용될 수 있다.

분류된 안전성 정책이 적용되는 레벨에 따라서 tagged value '{security=' '}'를 이용하여 어떤 안전성 정책이 사용될 수 있는지를 명시한다.

3.3.2.4 분산 트랜잭션에 대한 고려

CORBA 환경에서 분산 트랜잭션[21,29]은 2PC protocol을 사용하므로 성능 측면에서 2PC protocol 사용에 의한 오버헤드를 줄이기 위하여 동일 분산 컴포넌트 내에서 2PC protocol이 수행되도록 한다. 따라서 3-계층 구조의 세 번째 계층에 해당되는 분산 컴포넌트에서 entity class에 대한 접근을 다루는 control class의 오퍼레이션 단위로 트랜잭션을 다루는 per-operation transaction model[20]을 이용한다.

3.3.3 분산 컴포넌트에 대한 명세

분산 컴포넌트 명세는 분산 컴포넌트에 대하여 제삼자가 구현할 수 있도록 하기 위한 것으로 요구 사항 추출 단계, 분석 단계, 설계 단계의 산출물을 중심으로 구성된다. 분산 컴포넌트의 명세의 구성요소는 그림 13과 같다. 명세는 그림 13에서처럼 요구 사항 추출 단계의 산출물인 use case 모델과 분석 단계의 산출물인 분석 모델, 설계 단계의 산출물인 설계 모델로 구성된다. 설계 모델의 class specifications는 각 class에 대한 description, attribute specification, operation specification으로 구성된다. 또한 interface specifications는 각 interface에 대한 operation specifications로 구성된다. Operation specifications의 구성요소는 operation name, return types, parameters, preconditions, postconditions이다.

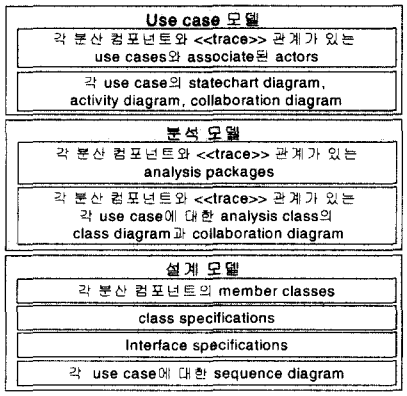


그림 13 명세의 구성 요소

## 4. 사례 연구

본 연구에서 제안한 분석 및 설계 방법을 분산 환경의 이기종 플랫폼으로 구성된 요소들과의 상호작용을 통하여 수행되는 지능형 교통 시스템(ITS: Intelligent Transportation System) 분야에 적용하였다. 지능형 교통 시스템의 첨단 교통 정보 시스템의 자가 운전자를 위한 여행전 정보 제공 시스템을 실제 적용분야로 선택하였다. 요구 문서는 그림 14와 같다.

- 여행 전 정보를 자가 운전자에게 제공하기 위한 것으로 운행 경로 정보, 교통 정보, 서비스 시설 정보 제공을 목적으로 한다.

1) 운행 경로 제공 기능

- 사용자가 지역, 출발지점, 도착지점을 지정하면 첨단 교통 관리 시스템(Advanced Traffic Management System: ATMS)으로부터 지역의 구간 운행 시간, 사고정보(원인, 예상 처리 시간, 점유 차선, 종류)를 얻어온다. 그리고 시스템 내에 존재하는 맵 데이터베이스로부터 구간 거리, 제한속도를 얻어 와서 경로를 계산한다. 그리고 예상 소요 시간, 총거리, 사고 정보를 구한다.

2) 교통 정보 제공 기능

- 사용자가 지역과 도로명을 입력하면 구간 운행 시간, 사고 정보는 첨단 교통 정보 시스템으로부터 얻어오고 차선, 제한속도, 구간 거리 정보는 시스템 내에 존재하는 맵 데이터베이스로부터 얻어온다. 날씨 정보는 기상청 시스템으로부터 얻어오고, 공사와 이벤트 정보를 외부 교통 정보 데이터베이스로부터 얻어온다.

3) 서비스 시설 정보 제공 기능

- 사용자가 지역 정보를 입력하고 숙박, 음식점, 주유소, 병원, 공원, 관공서 중에서 원하는 서비스 시설 종류를 선택하면 서비스 시설 정보 데이터베이스로부터 정확한 상호명, 주소, 홈페이지, 대표전화번호, 비고 정보를 얻어온다.

그림 14 요구 사항 문서

### 4.1 요구 사항 추출(Requirement Capture)

그림 15는 3.1과 3.2단계를 수행한 결과로 그림 15의 '(1) Actor 추출'이 3.1단계의 수행결과이고 '(2) Use case 추출'이 3.2단계의 수행결과이다.

그림 16, 그림 17은 3.1.3단계와 3.1.4 단계의 수행 예이다.

그림 17에서 'Provide traffic info.' use case 수행 과정과 'Provide routing info.' use case 수행 과정에서 교통 정보를 얻어오는 기능이 공통적으로 포함되지

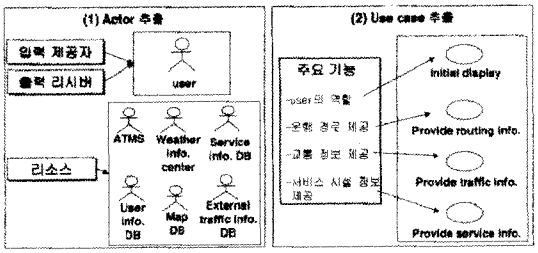


그림 15 actor와 use case 추출 작업 수행 예

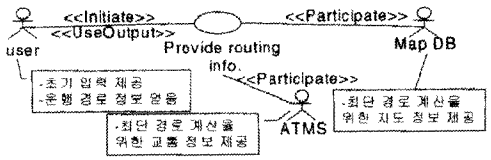


그림 16 use case와 actor간의 association 형성 예

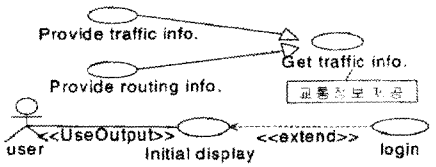


그림 17 use cases간의 relationship 형성 예

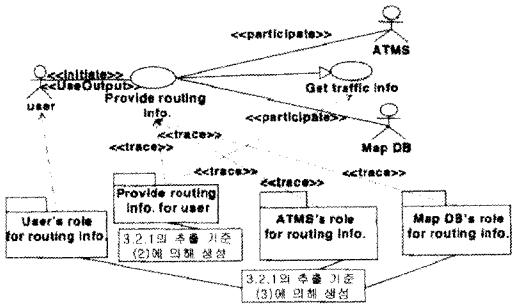


그림 18 'Provide routing info.' use case와 관련된 analysis package 추출

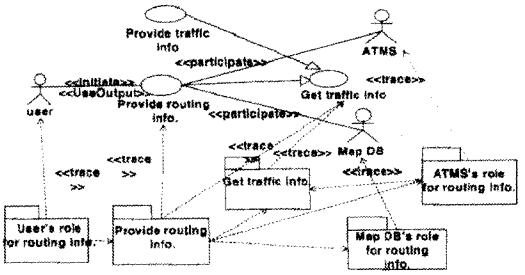


그림 19 'Provide routing info.' use case와 관련된 analysis package간의 의존 관계 형성

만 요구하는 정보가 달라질 수 있다. 따라서 'Get traffic info.' use case가 'Provide traffic info.' use case와 'Provide routing info.' use case에 대하여 일반화 관계로 형성된다. 또한 'user' actor가 초기화면에서 3가지 기능 중 하나의 기능을 선택하였을 때 로그인이 되어 있지 않다면 사용자가 기본적으로 요구한 기능이 아닌 로그인을 수행하도록 하게 하므로 login 기능이 'login' use case로 분리되고 확장 관계가 형성된다.

### 4.2 분석(Analysis)

그림 18은 3.2.1 단계의 수행 예이다. 그림 18에서 'Provide routing info.' use case를 중심으로 analysis package를 추출한 것으로 3.2.1에서 분류한 analysis package 추출 기준에 의한 수행 결과이다.

그림 19는 3.2.2 단계의 수행 예이다.

그림 19에서 중복 포함되는 'Get traffic info.' use case가 'Get traffic info.' analysis package로 분리되고 'Provide routing info.' analysis package와 'Get traffic info.' analysis package간에 의존 관계가 설정된다. 또한 3.2.1 단계의 의존관계 형성 기준에 의해 'ATMS's role for routing info.' analysis package와 'Map DB's role for routing info.' 'User's role for routing info'analysis package는 'Provide routing info.' analysis package에 대해 의존 관계가 설정된다. 그리고 'Get traffic info' analysis package와 'ATMS's role for routing info.' analysis package는 'Provide routing info.' use case에 대하여 공통되는 수행 과정상에 해당되고 'ATMS's role for routing info.' analysis package가 'Get traffic info.' analysis package에 대해서 구체화된 관계이므로 의존관계가 설정된다.

### 4.3 설계(Design)

그림 20은 3.3.1 단계의 수행 예로서 3-계층 구조에서 두 번째 계층에 해당될 수 있는 분산 컴포넌트를 추출하는 것에 대한 예이다. 그림 20(a)의 경우는 association 관계가 있는 control class들에 대하여 추출된 것으로 'Map info. for user' control class는 사용자가 출발지점과 도착지점을 입력할 때 맵 정보를 제공하기 위한 것으로 다른 analysis class와 직접적인 association 관계가 존재하지 않아 하나의 분산 컴포넌트 단위로 추출된다. 그림 20(b)의 경우는 control class간의 관계에

서 aggregation 관계에 해당되는 경우 whole-part 관계에서 'Traffic info. acquisition' control class에 대하여 part에 해당되는 'Static traffic info. acquisition'이 하나의 분산 컴포넌트의 단위로 추출된다.

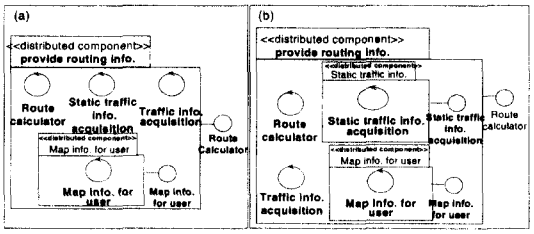


그림 20 분산 컴포넌트 추출

그림 21은 3.3.2 단계의 '3.3.2.1.성능에 대한 고려'의 수행 예로서 원거리 호출을 줄이기 위한 접근 방법 중에서 분산 컴포넌트를 재구조화하는 측면에 대한 예이다. 그림 21에서 'Routing info. request UI' boundary class에서 맵 정보를 얻어오기 위하여 'Map info. for user' control class의 기능을 사용하고 이로 인해 원거리 호출이 존재하게 된다. 그러나 'Map info. for user' control class는 다른 분산 컴포넌트에 포함된다고 해도 추가적인 오버헤드는 없다. 이런 경우에 'Map info. for user' control class를 원거리 호출 관계가 있는 'User's role for routing info' 분산 컴포넌트에 옮김으로써 원거리 호출을 줄인다.

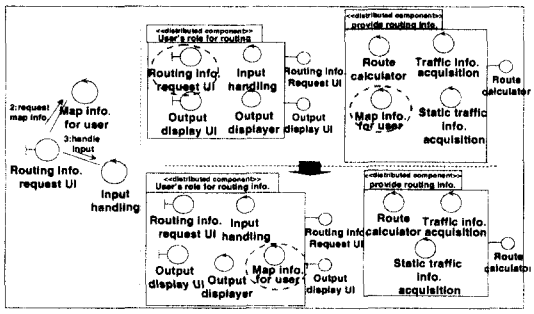


그림 21 성능을 고려한 분산 컴포넌트 재구조화

그림 22는 '3.3.2.3. 안전성(security)에 대한 고려'의 수행 예로서 3-계층 구조의 첫 번째 계층의 경우에 해당되는 예이다. 그림 22에서처럼 계층의 특징과 해당 기능에 따라 사용되는 안전성 정책을 tagged value의 형태로 나타내 준다.

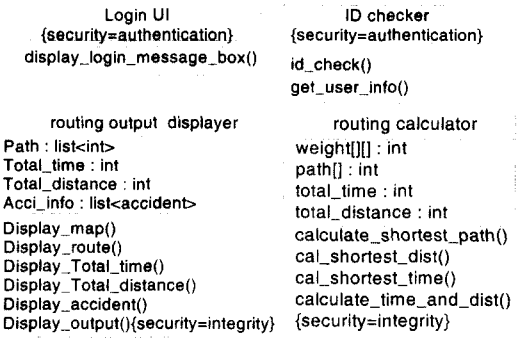


그림 22 3-계층 구조의 첫 번째 계층에서의 안전성을 고려한 클래스 설계

그림 23, 24는 '3.3.3. 분산 컴포넌트에 대한 명세'의 수행 예로서 'user's role for routing info.'분산 컴포넌트에 대한 명세의 예이다. 그림 23은 명세의 구성 요소 중 use case 모델에 대한 예이다. 그림 24는 명세의 구성요소 중 설계 모델의 일부분에 대한 예이다.

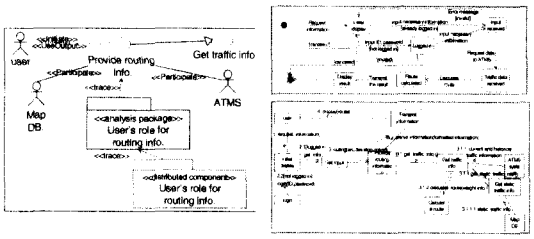


그림 23 명세구성요소 중 use case 모델

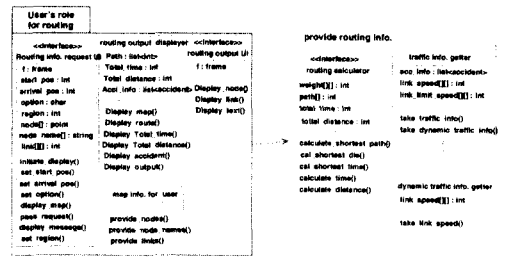


그림 24 명세의 설계 모델의 일부 예

## 5. 결론 및 향후 연구

다양한 플랫폼에 분산될 수 있는 컴포넌트로 구성된 소프트웨어를 개발하기 위해서는 체계적인 컴포넌트 기반 개발 방법이 필요하다. 그러나 기존의 Catalysis, Unified software development process, CBD96 등의 컴포넌트 기반 개발 방법론은 각 작업들간의 관계와 각 작업의 산출물간의 관계를 명확하게 제시하지 않았고 추상적인 프로세스를 제시하고 있어 실질적으로 적용하기에 어려움이 있었다. 또한 오늘날의 소프트웨어 개발에서 기본적으로 고려되어야 하는 분산 이슈를 체계적으로 다루지 않았다. 본 논문에서는 이러한 문제점들을 해결하기 위하여 분산 컴포넌트 기반의 소프트웨어 분석 및 설계 방법을 제안하였다. 본 연구에서 제안한 방법은 프로세스 측면에서는 Unified software development process의 전체 작업구성을 바탕으로 요구 사항 추출, 분석, 설계 단계로 구성되며 각 단계에 속한 작업들과 작업들의 산출물들에 대하여 체계적이고 구체적인 지침을 제시하였다. 또한 작업간의 관계에 있어서도 이전 작업들의 산출물을 바탕으로 다음 작업의 산출물을 형성하는 구체적인 지침을 제시함으로써 산출물간의 일관성(consistency)을 유지할 수 있도록 하였다. 뿐만 아니라 설계 단계에서 분산에 따른 기본적인 고려 사항들인 성능, 결합 방지, 안전성, 분산 트랜잭션 등을 명시적으로 다루는 지침을 CORBA와의 접목을 고려하여 제시하였다. 마지막으로 제안한 방법을 지능형 교통 시스템의 교통 정보 시스템 분야에 적용해 봄으로써, 제안한 방법이 분산 응용 소프트웨어 개발에 효과적으로 적용 가능함을 확인하였다.

결론적으로 본 연구를 통하여 추상적인 지침과 명확하지 않은 작업 산출물간의 관계로 인하여 발생할 수 있는 소프트웨어 개발 과정에서의 모호성을 감소시킬 수 있다. 또한 분산 컴포넌트 아키텍처인 CORBA를 고려함으로써 분산 응용 소프트웨어를 분산 환경에 보다 적합하게 분석 및 설계할 수 있다.

향후 연구 과제로는 CORBA 기반의 실질적인 구현과 분산 이슈에 대한 세부적인 연구를 통하여 분산 이슈를 다루기 위한 지침을 보완하고, 지능형 교통 시스템 뿐만 아니라 다른 도메인의 응용 시스템에 적용하여 실질적으로 활용 가능한 효과적인 소프트웨어 개발 방법을 제시하는 연구가 필요하다. 또한 CORBA 뿐만 아니라 EJB, DCOM 등을 고려하여 분산 이슈를 다루기 위한 지침을 제시하는 연구가 필요하다.

## 참 고 문 헌


[1] Sun Microsystems, "Enterprise JavaBeans Specification 1.1," http://java.sun.com/.

[2] Box, D., *Essential COM*, p.440, Addison-Wesley, January 1998.

[3] OMG, *The Component Object Request Broker: Architecture and Specification*, 1999.

[4] OMG, *OMG Unified Modeling Language Specification*, version 1.3, March 1999.

[5] D'Souza, D. and Wills, A. C., *Objects, Frameworks, and Components with UML*, Addison-Wesley, 1998.

[6] Jacobson, I., Booch, G., and Rumbaugh, J., *The Unified Software Development Process*, Addison-Wesley, January 1999.

[7] Sterling, *The CBD96 Standard Version 2.1*, Sterling, July 1998.

[8] Harmon, P., "Visual Modeling Tools, Case Vendors, and Component Methods," *Component Development Strategies*, pp. 1 - 16, June 1999.

[9] 김수동, "컴포넌트 정의 및 관련 기술 동향", *소프트웨어 공학회지*, 12권, 3호, pp. 5-18, 1999.

[10] Chelliah, M. and Ahamad, M., "System mechanisms for distributed object-based fault-tolerant computing," *Proceedings of IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, pp. 234-241, 1995.

[11] Microsoft, "Understanding How Distributed Components Affect Performance," http://msdn.microsoft.com/ library, 1999.

[12] Yau, S. S. and Bing X., "Object-oriented distributed component software development based on CORBA," *COMPSAC '98*. pp. 246 - 251, 1998.

[13] Cicalese, C. D. T. and Rotenstreich, S., "Behavioral specification of distributed software component interfaces," *IEEE Computer*, Vol. 32, No. 7, pp. 46-53, July 1999.

[14] Hofmann, H. D. and Stynes, J., "Implementation reuse and inheritance in distributed component systems," *COMPSAC '98*, pp. 496-501, 1998.

[15] Donaldson, D. I. and Magee, J. N., "Distributed system design using CORBA components," *International Conference on System Sciences*, pp. 4-13, 1997.

[16] Lee, S. D., Yang, Y. J., Cho, F. S., Kim, S. D., and Rhew, S. Y., "COMO: a UML-based component development methodology," *APSEC '99*, pp. 54-61, 1999.

[17] Kobryn, C., "Modeling enterprise software architectures using UML", *EDOC '98*, pp. 25-34, 1998.

[18] Chen, H. B., Bimber, O., Chhatre, C., Poole, E.,

and Buckley, S. J., "eSCA: a thin-client/server/Web-enabled system for distributed supply chain simulation," *Winter Simulation Conference Proceedings*, Vol. 2, pp. 1371-1377, 1999.
[19] Sanz, I. and Mazzucchelli, L., "Distributed objects in a large scale text processing system(industrial case study)", *Proceedings of the International Symposium on Distributed Objects and Applications*, pp. 224-229, 1999.
[20] 최유희, 염근혁, "분산 컴포넌트 기반의 소프트웨어 설계 방법", 한국정보과학회 추계 학술 발표 논문집, pp. 498-500, 2000.
[21] Slama, D., Grabis, J., and Russel, P., *Enterprise CORBA*, Prentice Hall PTR, March, 1999.
[22] Acton, D., Coatta, T., Phillips, P., and Sample, M., "A development framework for building fine-grained CORBA applications," *EDOC '98*, pp. 183-193, 1998.
[23] Gokhale, A. S. and Schmidt, D. C., "Measuring and optimizing CORBA latency and scalability over high-speed networks," *IEEE Transactions on Computers*, Vol. 47, No. 4, pp. 391-413, April 1998
[24] Arora, A. and Kulkarni, S. S., "Detectors and correctors: a theory of fault-tolerance components," *International Conference on Distributed Computing Systems*, pp. 436-443, 1998.
[25] OMG, *Fault Tolerant CORBA*, December, 1999.
[26] Olawsky, D., Payne, C., Sundquist, T., Apostal, D., and Fine, T., "Using composition to design secure, fault-tolerant systems," *DARPA Information Survivability Conference and Exposition*, pp. 380-390, 2000.
[27] OMG, *Security Service Specification*, December 1998.
[28] IONA Technologies, *OrbixSecurity 3.0 White Paper*, September 1999.
[29] OMG, *CORBAServices: Common Object Service Specification*, December 1998.

**최 유 희**

1999년 부산대학교 컴퓨터공학과(학사). 2001년 부산대학교 컴퓨터공학과(석사). 2001년 ~ 현재 한국전자통신연구원 컴퓨터소프트웨어 기술 연구소 S/W공학 연구부 연구원. 관심분야는 컴포넌트 기반 소프트웨어 개발 방법론, 소프트웨어 아키텍쳐, 분산 컴포넌트, 소프트웨어 재사용 등임.

**염 근 혁**

정보과학회논문지 : 소프트웨어 및 응용
제 28 권 제 10 호 참조