컴포넌트 기반 소프트웨어 개발을 위한 도메인 아키텍처 설계

하현주˘ 염근혁
부 산 대 학 교 컴 퓨 터 공 학 과
{hjha2, yeom}@hyowon.cc.pusan.ac.kr

Domain Architecture Design for Component-Based Software Development (CBD)

Hyun-Ju Ha∪ Keunhyuk Yeom
Dept. of Computer Engineering, Pusan National University

요 약

CBD(Component-Based Development)는 이미 존재하는 소프트웨어 컴포넌트를 조립함으로써 시스템을 개발하는 방법이다. 컴포넌트를 이용하여 시스템을 개발하는 것은 개발시간과 비용을 줄이고, 생산성을 향상시키는 등 여러 가지 장점을 가진다. 그러나 여러 벤더에 의해 개발된 컴포넌트를 조립하는 것은 쉬운 일이 아니다. 이를 위해 컴포넌트가 어떤 문맥에서 사용되는 지 이해하는 것이 필요하며 이 문맥은 아키텍처에 의해 결정된다. 따라서 본 논문에서는 컴포넌트 기반 개발을 위한 도메인 아키텍처를 제안한다.

도메인 아키텍처는 도메인을 구성하는 컴포넌트와 그들간의 관계에 대한 정보를 제공함으로써, 어플리케이션 개발을 쉽게 한다. 또한 도메인 아키텍처는 관련된 여러 시스템을 위한 아키텍처이므로, 도메인 아키텍처 상의 컴포넌트는 재사용성이 높다.

1. 서론

CBD(Component-Based Development)는 이미 존재하는 소프트웨어 컴포넌트를 조립함으로써 시스템을 개발하는 방법이다[1]. 컴포넌트를 이용하여 시스템을 개발하는 것은 소프트웨어 개발 시간과 비용을 줄이고, 생산성을 향상시키며, 이미 검증받은 컴포넌트를 사용함으로써 위험성을 줄이고, 표준 아키텍처 상에서 동작하기 때문에 사용에 있어서 일관성을 높이며, 어플리케이션 개발을 위한 여러 가지 해결책이 존재하므로 가장 좋은 해결책을 선택할 수 있다는 장점을 가진다[2].

그러나 컴포넌트는 다른 아키텍처 스타일에서 설계되고, 다른 연결 표준(interconnection standard)상에서 구현되기 때문에 여러 벤더들로부터 개발된 컴포넌트를 조립한다는 것은 쉬운 일이 아니다. 이를 위해 재사용 가능한 컴포넌트를 개발할 때에는 어떤 문맥(context)에서 사용될 것인가를 잘 이해하는 것이 필요하며, 이 문맥은 소프트웨어 아키텍처에 의해 결정된다. 따라서 컴포넌트를 기반으로 어플리케이션을 개발하는 것은 아키텍처를 기반으로 하였을 때만 가능하다[3].

본 논문에서는 컴포넌트 기반 개발을 위한 도메인 아키텍처를 제안하고, 제안한 방법에 따라 첨단 대중교통 시스템을 위한 아키텍처를 개발한다. 제안하는 도메인 아키텍처는 어플리케이션 개발 단계마다 필요한 정보들을 제공하여 도메인 아키텍처를 기반으로 어플리케이션을 개발할 수 있게 한다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 기존의 아키텍처 뷰와 첨단 대중교통 시스템에 대해 살펴본다. 3장에서는 컴포넌트 기반 개발을 위한 도메인 아키텍처를 제안하고, 첨단대중교통 시스템 도메인 아키텍처를 개발한다. 4장에서는 결론 및 향후 연구 방향을 제시한다.

2. 관련연구

2.1 Software architecture view

"4+1" view model은 그림 1과 같이 logical view, process view, development view, physical view와 scenarios를 사용하여 아키텍처를 서술한다[4].

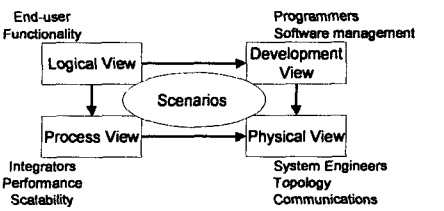

그림 1 "4 +1" view model

"4+1" view model은 다양한 stakeholder의 관심을 분리하여 설명하고 있다. 그러나 "4+1" view model은 일반적이기 때문에 컴포넌트 기반 소프트웨어 개발을 위해서는 좀 더 상세히, 그리고 컴포넌트의 특징을 충분히 포함할 수 있도록 정의될 필요가 있다.

2.2 첨단대중교통 시스템 (APTS : Advanced Public Transportation System)

기존의 교통시스템에 전자, 통신, 제어, 컴퓨터 등 첨단 기술을 접목시켜 신속, 저렴하고 안전한 교통환경을 확보하며 운영의 효율화를 기한 지능형 교통 시스템(Intelligent Transportation Systems : ITS)의 한 분야이다[5][6]. 대중교통 이용자 및 대중교통의 선택적 이용자들에

게 대중교통 수단 및 시설의 상호보완적 이용을 제공하는 시스템으로, 대중교통 운행의 신뢰성의 향상을 통하여 승용차 운전자들을 대중교통 수단으로 전환하는 결과를 도모한다[5].

3. 도메인 아키텍처 (domain architecture)

도메인 아키텍처는 도메인을 구성하는 컴포넌트와 그들간의 관계를 표현함으로써, 도메인이 가진 문제에 대한 해결책을 제시한다.

하나의 시스템은 많은 요구사항을 가진다. 모든 요구사항들이 동일한 아키텍처에 의해 지원된다면 그들을 독립적으로 만족시키는 것은 불가능하다[7]. 따라서 하나의 시스템에 대해 여러 뷰들이 표현되어야 한다. 본 논문에서는 도메인 아키텍처를 어플리케이션 개발 단계에서 필요한 정보들을 바탕으로 여러 뷰로 나눈다. 표 1은 어플리케이션 개발 단계와 각 단계마다 수행되는 일, 필요한 정보들을 나타낸다.

표 1 어플리케이션 개발 단계와 필요한 정보

	수행하는 일	필요한 정보
requirements analysis	- 요구사항 분석, 해당 도메인 찾기 - 도메인 모델을 참조, 요구사항 재정의	- 도메인 명세서 - 도메인 모델
component identification	- 요구사항를 만족하는 컴포넌트 인식	- 특정 요구사항을 만족하는 컴포넌트
component adaptation	- 컴포넌트 커스터마이즈 또는 - 어플리케이션에 맞는 컴포넌트 선택	- 커스터마이즈 지점 - 컴포넌트 선택 방법
component deployment	- 컴포넌트를 프로세스에 할당	- 실행 시 컴포넌트 간의 관계
component assembly	- 커넥터 선택 - 컴포넌트 조립	- 컴포넌트간의 연결 관계 - 커넥터 정보

표 1을 바탕으로 그림 2와 같은 도메인 아키텍처 뷰를 제안한다.

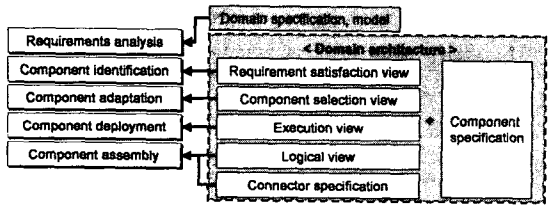

그림 2 도메인 아키텍처 뷰

3.1 가정

APTS 도메인을 위한 아키텍처를 개발하기 위해 그림 3과 같은 도메인의 목적을 정의하였다.

Transit vehicle에 대한 schedule을 감시하고 그 정보를 관리함으로써
transit user에게 vehicle의 static, dynamic 운행정보를,
vehicle driver에게 vehicle의 운행에 대한 가이드를 제공한다.

그림 3 APTS의 도메인 목적

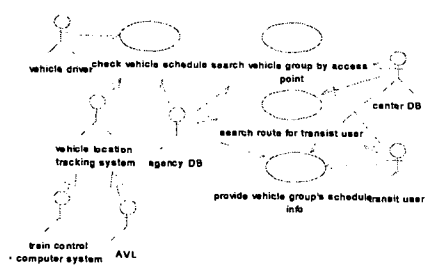

그림 4 APTS 도메인 요구사항

그림 3의 APTS 도메인 목적을 만족시키는 4가지 요구사항을 그림 4와 같이 정의하고, 이를 위한 컴포넌트를 그림 5와 같이 추출하였다.

- Route calculation
- Schedule adherence
- Display schedule info
- Handle access point info
- *Handle vehicle group info*
- Route search interface
- Schedule check interface
- Schedule providing interface
- Vehicle group searching interface
- *Driver interface*
- Transit user interface

그림 5 ATPS 컴포넌트

3.2 Requirement satisfaction view

Requirement satisfaction view는 특정 요구사항을 수행하기 위해 필요한 컴포넌트들을 표현한다. 따라서 각 요구사항마다 뷰가 생성되고, 이를 UML의 use-case diagram을 사용하여 표현한다. Requirement satisfaction view에서는 도메인 요구사항을 system으로, 컴포넌트를 use case로, 도메인에서 사용하는 다른 도메인 컴포넌트, 하드웨어 장치, 데이터베이스 등을 actor로 표현한다. 그림 6은 APTS 도메인에서의 requirement satisfaction view를 나타낸다.

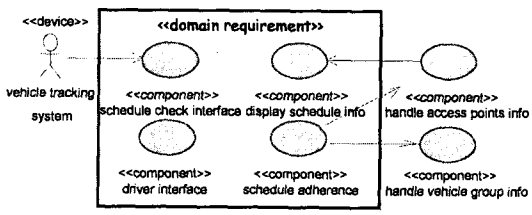

그림 6 'check vehicle schedule'의 requirement satisfaction view

3.3 Component selection view

Component selection view는 동일한 인터페이스와 기능을 제공하는 컴포넌트들이 가질 수 있는 여러 특성에 대해 표현한다. 설계 또는 구현된 각각의 컴포넌트가 가지는 비기능적인 특성, 컴포넌트 아키텍처, 컴포넌트 버전, 구현 기술, 알고리즘 등에 대해 테이블을 사용하여 표현한다.

3.4 Execution view

Execution view는 특정 요구사항을 수행하기 위해 발생하는 컴포넌트간의 상호작용을 표현한다. Execution view는 성능(performance)이나 통신(communication), 동기화(synchronization)와 관련된 문제를 다룸으로써 어플리케이션 개발 시 컴포넌트를 프로세스에 할당할 때 필요한 정보는 제공한다. Execution view는 UML의 collaboration diagram을 사용하여 그림 7과 같이 표현한다.

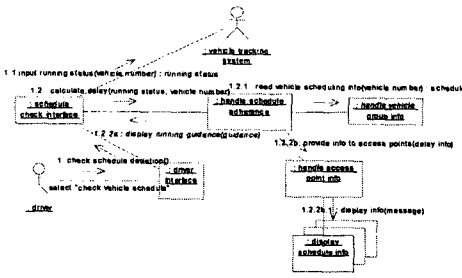

그림 7 'check vehicle schedule'의 execution view

그림 7에서 보듯이 Collaboration numbering을 통하여 동시에 실행될 수 있는 인터페이스를 표현한다. 그리고 컴포넌트간에 교환되는 메시지를 표현함으로써 컴포넌트의 인터페이스를 인식한다.

3.5 Logical view

Logical view는 도메인 전체에 대하여 컴포넌트와 커넥터, 그들간의 관계를 class diagram을 사용하여 표현한다. Logical view의 커넥터는 execution view로부터 추출한다. Execution view에서 제공하는 각 컴포넌트간에 실행 시 발생하는 관계를 기준으로 어떤 특성을 가지는 커넥터가 필요한 지 인식한다. 그림 8은 두 컴포넌트간의 실행 시 관계에서 커넥터를 추출하는 것을 나타낸다.

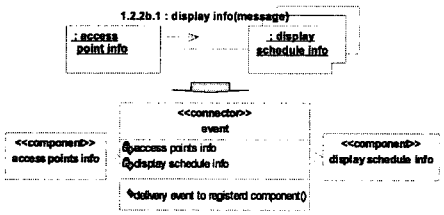

그림 8 커넥터 추출

그림 8에서 'handle access point info'컴포넌트는 여러 노드에 배치된 'display schedule info'컴포넌트의 'display info(message)'를 호출한다. 따라서 'handle access point info'가 생성한 메시지를 여러 'display schedule info' 컴포넌트에 전달하기 위한 'event' 타입의 커넥터가 필요하다. 커넥터는 UML의 interface notation을 사용하여, attribute 부분에는 커넥터가 연결시키는 컴포넌트 이름을, operation 부분에는 커넥터의 연결방법을 기술한다.

3.6 Component specification

Component specification은 각 컴포넌트가 무엇을 하는지, 그리고 그 컴포넌트를 어떻게 사용할 수 있는 지에 대한 정확한 정보를 제공한다. 컴포넌트 명세서는 컴포넌트의 목적, 컴포넌트 행동에 대한 서술, 컴포넌트 인터페이스 서술, 동일하거나 유사한 기능을 제공하는 컴포넌트, 비기능적인 특징 등으로 구성된다. 컴포넌트 인터페이스 서술은 인터페이스에 포함된 각 오퍼레이션에 대한 서술을 포함한다. 오퍼레이션 서술은 오퍼레이션의 목적, 서명(signature), 전제조건(precondition), 완료조건(postcondition), 필요한 다른 컴포넌트 등에서 대한 정보를 포함한다. Component specification은 좀 더 정확한 정보 제공을 위하여 그림 9와 같이 OCL(Object Constraint Language)을 사용하여 표현한다[8,9].

- Purpose
  - vehicle의 현재위치를 고려하여 운행상태를 확인한다.
- interface specification

  1) caiculate delay(run_status : running_status, vehicle_number : string) delay : delay_info
  - operation intent : 운행중인 vehicle의 현재 위치를 입력받아 계획된 schedule과의 차이를 계산.
  - precondition : run_status.location <> null, run_status.speed <> null,vehicle_number <> null
  - postconditiont :

    delay->notEmpty

    delay.access_point_name->first = running_status.location

    delay.access_point_name->last = schedule_info.destination

    delay.diff <> null -- 현재 위치를 기준으로 한 schedule 과의 차이

    delay.new_speed <> null -- schedule과 맞추기 위한 속력
  - component dependency
    - handle vehicle group info.read schedule info (vehicle number) : schedule info

그림 9 'schedule adherence'의 component specification

3.7 connector specification

Connector specification은 컴포넌트와 컴포넌트를 연결시켜주는 커넥터에 대해 커넥터 타입, 커넥터가 연결시켜주는 컴포넌트, 연결되는 방법 등의 요소로 서술한다[10].

4. 결론

본 논문에서는 컴포넌트 개발 방법론을 지원하기 위한 도메인 아키텍처를 제시하였다. 어플리케이션을 개발하는 과정에서 필요한 정보마다 각 뷰를 두어 간단하고 정확하게 도메인 아키텍처를 표현할 수 있게 하였다. 이러한 도메인 아키텍처 뷰는 어플리케이션 개발 시 유용한 정보를 제공함으로써 좀 더 쉽게 어플리케이션을 개발할 수 있게 한다. 각 도메인 아키텍처를 표준 모델링 언어인 UML을 사용하여 표현함으로써 보다 쉽게 이해할 수 있도록 하였으며, OCL을 사용하여 component specification을 기술함으로써 컴포넌트에 대한 정확한 정보를 전달할 수 있게 하였다. 그리고 본 논문에서 제안한 도메인 아키텍처를 첨단 대중 교통 시스템에 적용하여 제안한 방법의 적용성을 살펴보았다.

향후 연구과제로 도메인 아키텍처 개발을 위한 도메인 분석 방법, 컴포넌트 추출방법에 대한 연구가 필요하다. 또한 도메인 아키텍처에 포함된 컴포넌트 명세서를 바탕으로 컴포넌트 내부 설계 및 구현 방법도 필요하다.

5. 참고 문헌

[1] SEI in in Carnegie Mellon University, http://www.sei.cmu.edu/str/descriptions/cbsd_body.html

[2] Short, K., "Component Based Development and Object Modeling", Sterling Software, 1997

[3] Ran, A., "Software Isn't Built From Lego Blocks", Proceedings of the fifth symposium on Software reusability , 1999.

[4] Kruchten, P., "Architectural BluePrints - The "4+1" Voew Model of Software Architecture, IEEE Software, pp.42-50, November 1995.

[5] 고속도로정보통신공단, http://www.hitelecom.co.kr/sayub/yuji/index.htm.

[6] ITS Korea, http://www.itskorea.or.kr/main.html.

[7] Ran, A., "Architectural Structures and Views", Proceedings of the third international workshop on Software architecture, pp. 117 - 120, 1998.

[8] OMG, "Unified Modeling Language Specification Version 1.3", http://www.omg.org, March 1999.

[9] Bennett, S., McRobb, S., and Farmer, R., "Object-Oriented Systems Analysis and Design", McGRAW-HILL INTERNATIONAL EDITIONS.

[10] Egyed, A., Mehta, N., Medvidovic, N., "Software Connectors and Refinement in Family Architectures", USC Center for Software Engineering, http://sunset.usc.edu, 1999