컴포넌트 기반 소프트웨어 개발을 지원하는 소프트웨어 아키텍처 뷰 모델

(The View Model of Software Architecture for Component Based Software Development)

박 준 석 † 문 미 경 † 염 근 혁 ††
(Joonseok Park) (Mikyeong Moon) (Keunhyuk Yeom)

요 약 컴포넌트 기반 소프트웨어 개발 방법은 새로운 패러다임으로 인식되고, 활발한 연구가 진행되고 있다. 그러나 컴포넌트 기반 개발에서 컴포넌트의 재사용성과 효율적인 소프트웨어 개발을 보장하기 위해서는 소프트웨어 아키텍처를 기반으로 한 개발이 필요하다.

본 논문에서는 컴포넌트 기반 소프트웨어 개발을 지원하기 위해, Kruchten이 제시한 4+1 뷰 모델을 기반으로 재 정의한 컴포넌트 기반 4+1 소프트웨어 아키텍처 뷰 모델을 제안한다. 또한 뷰 모델의 요소와 UML을 이용한 뷰 모델 요소의 표현을 제시한다. 이 아키텍처는 컴포넌트의 사용문맥을 각 뷰에 반영함으로써 소프트웨어에 대한 이해와, 컴포넌트의 상호 작용에 대한 정보 등 컴포넌트 기반 소프트웨어 구성에 대한 틀을 구성한다.

키워드 : 소프트웨어 아키텍쳐, 4+1 뷰 모델, 컴포넌트 기반 소프트웨어 개발

Abstract Component Based Software Development has been recognized as a new software development paradigm, and received much attention among researchers. However, it requires software architecture based development to assure component reusability and efficient software development.

This paper proposes the Component Based 4+1 View Model of software architecture to support component based software development. It is redefined on the basis of the existing 4+1 view model of software architecture developed by Kruchten. Also, we describe the elements of the view model in detail with UML. This architecture constructs the foundation of component based software such as increasing the understanding of software and providing the information about how the components interact with each other. It can be done by exposing the context for the use of software components to each views.

Key words : Software Architecture, 4+1 View Model, Component Based Software Development

1. 서 론

컴포넌트 기반 개발(Component Based Development)은 컴포넌트의 생성, 선택, 평가와 통합으로 구성되는 소프트웨어 개발 방법의 새로운 패러다임이다[1]. 이러한 CBD 방법을 사용함으로써 소프트웨어 개발 시간의 단축, 비용 절감, 유지보수의 편이성 등의 장점을 얻을 수 있다[2]. 그러나 CBD 방법만으로 소프트웨어를 개발하는 것은 컴포넌트가 어떤 문맥에서 사용될 것인지에 대한 정보를 제공해주지 못하고, 단지 한 소프트웨어 개발에 종속적이 되어 컴포넌트의 재사용 이점을 얻을 수 없다[3].

따라서 CBD 방법만의 소프트웨어 개발의 단점을 보완하고, 소프트웨어가 가지는 주요한 요구사항을 만족시키며, 소프트웨어가 시간이 흐름에 따라 변경되더라도 소프트웨어의 무결성을 보증하기 위해서는, 개발되는 소프트웨어의 다양한 관점을 반영해 주어 개발의 청사진 역할을 하는 소프트웨어 아키텍처 뷰 모델이 필요하다.

본 연구는 한국과학재단 목적기초연구(R05-2000-000-00276-0)지원으로 수행되었음.

† 비 회 원 : 부산대학교 컴퓨터공학과
pjs50@pusan.ac.kr
mkmoon@pusan.ac.kr
†† 종신회원 : 부산대학교 컴퓨터공학과 교수
yeom@pusan.ac.kr

논문접수 : 2002년 5월 24일
심사완료 : 2003년 3월 10일

소프트웨어 아키텍처 뷰 모델 중의 하나로 Kruchten이 제시한 4+1 뷰 모델이 존재한다. 이 모델은 다양한 Stakeholder들의 관점을 반영하고 있다. 그러나 컴포넌트에 기반한 소프트웨어 개발을 지원하기 위해서는 컴포넌트의 특징, 속성이 반영되어야 하며, 일반적인 모델인 4+1 뷰 모델이 컴포넌트 기반에 맞도록 재 정의될 필요가 있다. 또한 각 뷰의 관점을 반영하기 위한 명확한 뷰의 요소와 표현에 대한 연구가 필요하다.

본 논문에서는 Kruchten이 제시한 4+1 뷰 모델을 바탕으로 컴포넌트 기반 소프트웨어 개발을 지원하는 소프트웨어 아키텍처 뷰 모델을 제시한다. 또한 제시된 뷰 모델을 바탕으로 뷰의 요소를 정의하고, 현재 널리 쓰이는 모델링 언어인 UML(Unified Modeling Language)[4]을 도입하여 뷰 모델 요소의 표현에 대해 제시한다.

2. 관련연구

2.1 소프트웨어 아키텍처 정의

소프트웨어 아키텍처에 대해서는 보편화된 정의가 없으며 다양한 정의가 존재한다.

Perry와 Wolf[5]는 {요소, 형태, 이론적 근거}로 결정되는 특정한 형태를 가지는 아키텍처 요소의 집합으로 정의한다. Garlan과 Perry[6]는 프로그램/시스템을 구성하는 컴포넌트의 구조, 상호관계, 그리고 설계와 변경을 결정하는 원칙과 지침으로 정의한다. Bass, Clements와 Kazman[7]은 소프트웨어 컴포넌트로 구성되는 시스템의 구조, 이러한 컴포넌트의 외부적으로 볼 수 있는 속성과 컴포넌트 사이의 관계로 정의한다. IEEE Std 1471-2000[8]에서는 컴포넌트로 구체화되는 시스템의 기본적인 구성, 컴포넌트 서로간의 관계와 설계, 변경을 통제하는 원칙이라고 정의한다.

2.2 소프트웨어 아키텍처 뷰 모델

하나의 시스템은 다양한 Stakeholder의 요구사항들을 가지고 있기 때문에, 아키텍처는 이렇게 다양한 Stakeholder의 관점을 반영해야 한다. 즉, 시스템의 복잡성을 관리하기 위해, 완전한 아키텍처의 기술은 다수의 뷰로 나누어져야 한다[9]. 여기서 뷰는 연관된 관점의 집합으로 전체적인 시스템의 표현을 의미하며, 시스템 Stakeholder의 하나 혹은 그 이상의 관점을 나타낸다[8]. 이러한 아키텍처 뷰로 구성된 소프트웨어 아키텍처 뷰 모델로 Kruchten이 제시한 4+1 뷰 모델[10], Hofmeister 등이 제시한 뷰 모델[11], Davis와 Williams가 제시한 뷰 모델[12]등이 존재한다.

Kruchten이 제시한 4+1 뷰 모델은 그림 1과 같이 논리 뷰(logical view), 개발 뷰(development view), 처리 뷰(process view), 물리 뷰(physical view)와 시나리오(scenario)의 5가지 뷰로 소프트웨어 아키텍처를 나타낸다.

논리 뷰는 객체 지향 방법이 사용된다면 객체의 형태로 실수요자에게 제공되는 시스템의 기능적 사항을 서술한다. 개발 뷰는 개발 환경에서 소프트웨어의 정적인 구조를 서술한다.

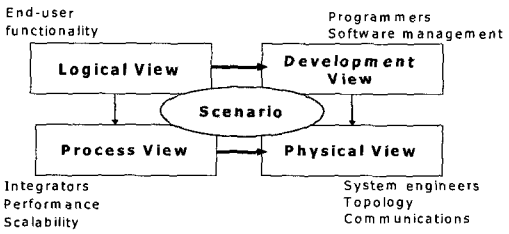


그림 1 4+1 뷰 모델 소프트웨어 아키텍처

처리 뷰는 설계의 동시성과 동기화 측면을 서술한다. 물리 뷰는 소프트웨어의 하드웨어 할당과 분포 측면을 서술한다. 그리고 시나리오는 전체 뷰들을 통합하는 역

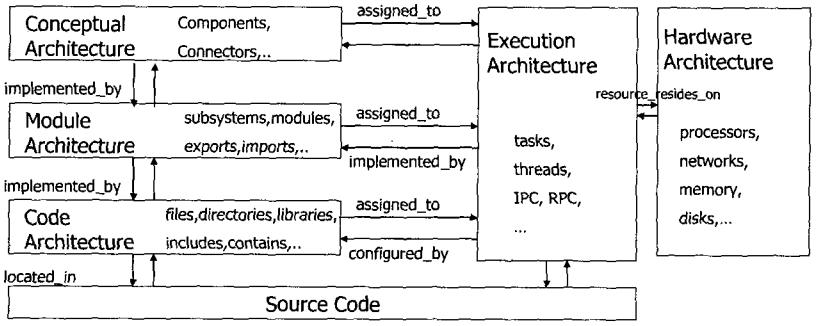


그림 2 Hofmesiter등이 제시한 뷰 모델

할을 한다. 4+1 뷰 모델은 다양한 Stakeholder의 관점을 분리하여 반영하고, 시나리오라는 개념의 사용, 일반적인 모델이라는 특징이 있다. 그러나 컴포넌트에 기반한 소프트웨어 개발을 지원하기 위해서는 컴포넌트의 특징, 속성 등이 반영되어야 하며 일반적인 뷰 모델의 재 정의가 필요하다.

Hofmeister등이 제시한 뷰 모델은 그림 2와 같이 개념 아키텍처(conceptual architecture), 모듈 상호연결 아키텍처(module interconnection architecture), 실행 아키텍처(execution architecture), 코드 아키텍처(code architecture)의 4가지 뷰로 소프트웨어 아키텍처를 나타낸다.

개념 아키텍처는 주된 설계 요소와 설계 요소들의 관계 측면에서 시스템을 서술한다. 모듈 상호연결 아키텍처는 기능적인 분해와 계층으로 구성된 두 개의 직교 구조를 포함한다. 실행 아키텍처는 시스템의 동적인 구조를 서술한다. 코드 아키텍처는 개발 환경에서 소스 코드, 이진 코드, 그리고 함수가 어떻게 조직화되어 있는지 서술한다. 제시된 소프트웨어 아키텍처는 이미지와 신호처리 시스템, 실시간 운영체제 시스템 등 여러 가지 시스템에 대한 조사를 바탕으로 만들어졌다는 점, 뷰를 이용한 관점의 분리, 개념 → 모듈 → 코드로 점점 정제되어간다는 특징을 가지고 있다. 그러나 구조적인 측면이 너무 부각되어 있고, 뷰에 관점을 가지고 있는 Stakeholder에 대한 부분은 언급되어 있지 않다. 또한 CBD에 기반하여 구성된 모델이 아니다.

Davis와 Williams가 제시한 뷰 모델은 그림 3과 같이 도메인 뷰(domain view), 컴포넌트 뷰(component view), 플랫폼 뷰(platform view), 인터페이스 뷰(interface view)와 문맥 뷰(context view)의 5가지 뷰로 소프트웨어 아키텍처를 나타낸다.

도메인 뷰는 아키텍처 모델을 구성하는 시스템 군에 의해 제공되는 기능성과 특성을 나타낸다. 컴포넌트 뷰는 어플리케이션과 도메인에 특정한 기능성을 제공하는 하드웨어와 소프트웨어 부분에 걸친 로드 맵이다.

플랫폼 뷰는 만약 COTS가 사용된다면, 운영 체제, 네트워크 요소와 하드웨어 요소 측면에서 무엇이 컴퓨팅 플랫폼(computing platform)을 구성하는지 서술한다.

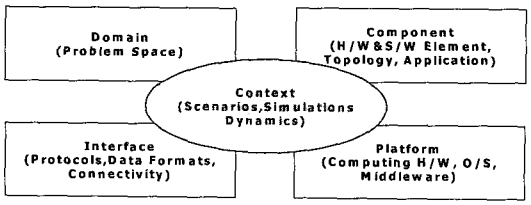


그림 3 Davis와 Williams가 제시한 뷰 모델

문맥 뷰는 용량, 타당한 행동, 신뢰성과 유지보수성 등과 같은 동적인 시스템 속성을 보여준다. 제시한 소프트웨어 아키텍처는 Kruchten이 제시한 4+1 뷰 모델에서 영향을 받았으며, 공통성과 다양성을 반영한 도메인 뷰의 존재, COTS 기반의 아키텍처라는 특징이 있다. 그러나 실제 적용을 위해서는 공통성과 다양성을 반영하여 도메인 뷰를 구성하는 것이 어려우며, 뷰에 관점을 가지고 있는 Stakeholder에 대한 부분은 언급되어 있지 않다.

2.3 컴포넌트의 특징과 속성, 컴포넌트 기반 개발 과정

본 논문에서는 컴포넌트의 특징과 속성을 다음과 같이 정의한다[3,13]. 첫째, 컴포넌트는 독립적으로 개발, 인도, 배치될 수 있으며 제 3자에 의해 조립 가능하다. 둘째, 컴포넌트는 제공하는 서비스에 대한 인터페이스와 요구하는 서비스에 대한 인터페이스를 가진다. 셋째, 컴포넌트는 코드의 변경 없이 커스터마이즈되고 다른 컴포넌트와 조립될 수 있다. 여기서 커스터마이즈된다는 것은 인터페이스의 변경 없이 새로운 기능의 추가가 가능하다는 것을 의미한다. 넷째, 컴포넌트는 대체 가능한 단위다. 즉, 동일한 인터페이스를 가진다면 다른 특성을 가진 컴포넌트로 대체 가능함을 의미한다.

컴포넌트 기반의 개발 과정은 기존의 요구분석→설계→구현이라는 개발 방식과는 다른 그림 4와 같은 활동을 가진다[14].

Find는 저장소의 컴포넌트를 어떻게 서술하고, 생성하는지를 정의한다. Select는 컴포넌트 기반의 소프트웨어 개발에 사용하기 위해서 저장소에서 컴포넌트를 선택한다. Adapt는 컴포넌트가 사용되는 새로운 문맥상에

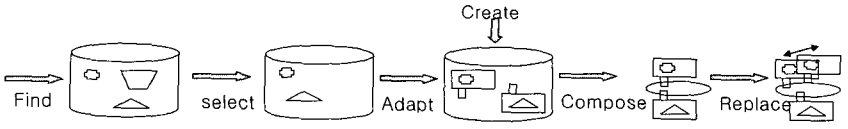


그림 4 컴포넌트 기반 개발에서의 활동

서 사용자의 요구사항을 만족하도록 선택된 컴포넌트를 커스터마이징하는 과정이다. Create는 변경 후에도 선택된 컴포넌트가 요구사항을 만족하지 않는 경우 새로운 컴포넌트를 개발하고 생성한다. Compose는 조립하고 통합하는 과정이다. Replace는 새로운 기능이 추가되면 오래된 버전의 컴포넌트는 새로운 수정된 버전의 컴포넌트로 교체하는 등 산출물 관리와 연관이 있는 과정이다.

3. 컴포넌트 기반 4+1 뷰 모델

3.1 컴포넌트 기반 4+1 뷰 모델 뷰

본 논문에서는 컴포넌트 기반 4+1 뷰 모델을 제시하기에 앞서 소프트웨어 아키텍처를 "다양한 Stakeholder의 관점을 반영하여 컴포넌트로 구성된 전체적인 시스템의 구조를 보여주며, 개발 프로세스에 사용될 뿐만 아니라, 설계자간에 기록되고 전달될 수 있는 상위 수준 시스템 표현" 이라고 정의한다[15].

컴포넌트 기반 4+1 뷰 모델은 위와 같은 아키텍처에 대한 정의를 바탕으로, Kruchten이 제시한 4+1 뷰 모델의 관점을 수용하여 (1)컴포넌트가 가지는 특징과 속성, (2)아키텍처 무결성을 유지하는데 중요한 설계 결정 사항, (3)CBD의 개념을 지원하고 사용한다는 측면 등을 반영한 모델이다.

컴포넌트 기반의 소프트웨어 개발을 지원하기 위해 문맥 뷰, 논리 뷰, 조립 뷰, 처리 뷰, 물리 뷰로 구성된 그림 5와 같은 컴포넌트 기반 4+1 뷰 모델을 제시한다.

컴포넌트 기반 소프트웨어 개발에서 핵심은 컴포넌트를 구성하는 것이다. Kruchten이 제시한 시나리오라는 뷰를 컴포넌트 기반 시스템 구성의 문맥을 제공한다는 측면에서 문맥 뷰로 재 정의하고, Primary Stakeholder로 컴포넌트 개발자를 정의한다. 문맥 뷰는 다른 4가지 뷰에 기초 정보를 제공해주며, 관점은 컴포넌트 기반 소프트웨어 개발에 대한 전체적인 문맥, 행동, 컴포넌트 구성정보, 아키텍처에 대한 이론적 근거를 나타낸다.

논리 뷰의 Primary Stakeholder는 Kruchten이 제시한 논리 뷰의 Stakeholder인 실수요자이며, 논리 뷰에 대한 관점은 시스템의 논리적인 구성, 컴포넌트의 논리적인 배치와 컴포넌트의 행동을 나타낸다.

Kruchten이 제시한 4+1 뷰 모델에서는 개발 뷰가 정의되어 있지만 컴포넌트 기반 소프트웨어 개발에서는 이미 구성되어 있는 컴포넌트를 조립하고 통합하는 관점이 필요하다. 따라서 개발 뷰를 조립 뷰로 재 정의하고, Primary Stakeholder로 컴포넌트 조립자를 정의한다. 조립 뷰에 대한 관점은 컴포넌트 기반의 시스템 통합·유지와 관련된 정보를 나타낸다.

처리 뷰의 Primary Stakeholder는 Kruchten이 제시한 4+1 뷰 모델의 Stakeholder인 시스템 통합자이며, 기존의 4+1 뷰에 대한 관점을 수용해서 성능, 규모 등을 고려한 동기화, 병행성 정보를 나타낸다.

물리 뷰의 Primary Stakeholder는 Kruchten이 제시한 4+1 뷰 모델의 Stakeholder인 시스템 엔지니어이며, 기존의 4+1 뷰에 대한 관점을 수용해서 컴포넌트의 하드웨어 매핑·물리적인 배치와 연관된 플랫폼 장치에 대한 정보를 나타낸다.

본 논문에서 제시하는 컴포넌트 기반 4+1 뷰 모델의 Stakeholder, 뷰, 뷰 관점을 도식화하면 그림 6과 같다.

3.2 컴포넌트 기반 4+1 뷰 모델 요소와 표현

컴포넌트 기반 4+1 뷰 모델의 관점을 반영하기 위해 정의한 뷰 모델 요소와 UML을 이용한 요소 표현을 도식화하면 그림 7과 같다.

그림 7에 제시된 것처럼 뷰 내부는 모델들로 구성되며, 모델들은 문서화 항목과 UML을 이용하여 표현한 다이어그램으로 구성된다.

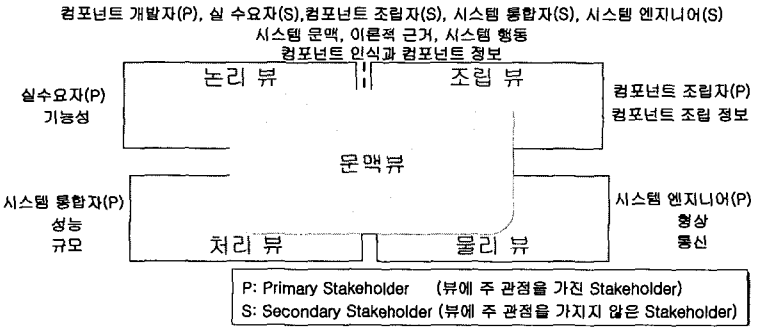


그림 5 컴포넌트 기반 4+1 뷰 모델

3.2.1 문맥 뷰

문맥 뷰는 시스템 모델, 유즈케이스 모델, 컴포넌트 구성 모델을 정의한다.

시스템 모델은 문맥 뷰에서 시스템의 전체적인 문맥, 아키텍처에 대한 이론적 근거, 결정사항을 제공하기 위한 요소로 시스템 문맥 다이어그램과 윤곽으로 구성된다. 시스템 문맥 다이어그램은 시스템과 상호작용하는 시스템, 문맥을 이해하기 위해 필요한 다른 요소, 그리고 이들의 연관관계를 어트리뷰트와 오퍼레이션을 제외한 UML의 클래스 다이어그램으로 표현하여 시스템과 연관된 전체적인 문맥을 나타낸다. 윤곽은 소프트웨어 아키텍처에 대한 이론적 근거와 결정사항에 대한 항목을 기술하는 것으로 그림 8과 같은 항목으로 구성한다.

유즈케이스 모델은 UML의 유즈케이스 다이어그램과 유즈케이스 서술로 구성된다. 유즈케이스 모델은 시스템의 행동과 시스템이 제공하는 중요한 기능성에 대한 정보를 제공한다. 본 논문에서는 유즈케이스 서술[3,17]에 대한 항목을 바탕으로 그림 9와 같은 유즈케이스 서술 항목을 정의한다.

컴포넌트 구성 모델은 시스템의 기본 요소인 컴포넌트 구성 정보를 제공한다. 컴포넌트 구성 모델은 컴포넌트 집합, 외부적으로 볼 수 있는 속성들, 인터페이스 명세 다이어그램으로 구성된다. 컴포넌트 집합은 컴포넌트 저장소로부터 요구사항에 맞는 컴포넌트를 선택하거나 필요한 컴포넌트를 추출해서 구성한다. 컴포넌트를 표현하는데는 UML 표기법 중 어트리뷰트와 오퍼레이션 부분을 없앤 클래스를 사용하여 나타낸다. 상용 벤더 혹은

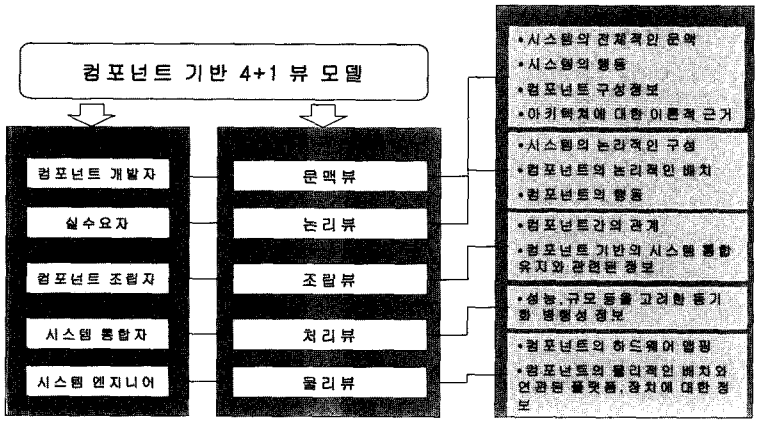


그림 6 컴포넌트 기반 4+1 뷰 모델 - Stakeholder, 뷰, 뷰 관점

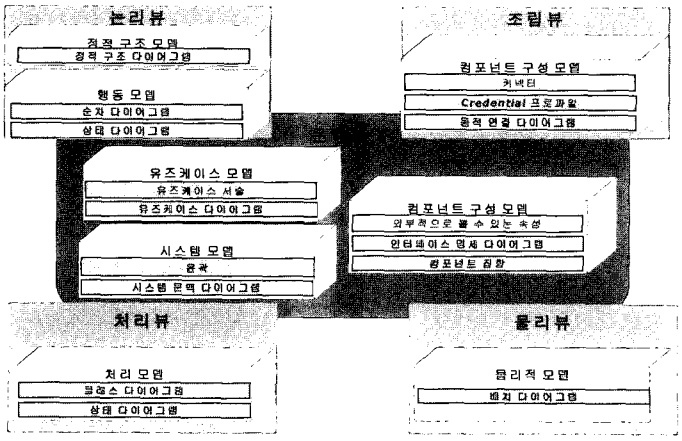


그림 7 컴포넌트 기반 4+1 뷰 모델 요소

공용 저장소 등 어떤 외부의 컴포넌트 저장소로부터 선택한 컴포넌트는 <<COTS>>라는 스테레오 타입을 선언해서 요구사항으로부터 직접 추출해서 구성한 컴포넌트와 차별성을 둔다. 또한 컴포넌트 분류 정보가 있을 경우, 예로 인터페이스 컴포넌트가 있다면 외부 저장소로부터 가지고 온 경우는 <<COTS:Interface>>형식의 표현을, 추출한 경우는 <<Interface>>의 형식으로 표현하여 나타낸다. 또한 컴포넌트를 가지고 있는 집합 개념을 표현하기 위해 UML 표기법 중 패키지를 함께 사용하여 나타낸다. 패키지는 요소들을 그룹으로 편성하기 위한 다목적 메카니즘이다[18]. 그림 10은 클래스와 패키지를 사용해서 표현한 컴포넌트 집합을 나타낸다.

외부적으로 볼 수 있는 속성은 다른 컴포넌트도 컴포넌트로 구성될 수 있다는 가정을 지칭하는 것으로 소프트웨어 아키텍처 정의 중 Bass, Clements와 Kazman이 언급한 개념이다. 본 논문에서는 [7]에서 외부적으로 볼 수 있는 언급되어진 속성과 몇가지 요소를 더 추가해서 그림 11과 같은 외부적으로 볼 수 있는 속성 항목을 정의한다.

인터페이스 명세 다이어그램은 그림 11에서 정의된 항목 중 공유 자원 사용과 인터페이스의 템플릿에 정의된 제공하는(provided) 서비스, 요청하는(required) 서비스 등 인터페이스에 대한 내용을 반영한다. 컴포넌트에 대한 인터페이스를 그룹화 하기 위해 UML의 패키지를 이용하여 표현한다. 인터페이스는 UML의 클래스를 사용하여 <<operation>>라는 스테레오 타입을 이용하여 나타낸다. 공유 자원에 대한 부분은 <<shared resource>>라는 스테레오 타입을 사용한 클래스로 나타내며, UML의 노트 표기를 사용하여 요청하는 인터페이스를 제공하는 컴포넌트를 나타낸다. 그림 12는 인터페이스 명세 다이어그램을 제시한다.

3.2.2 논리 뷰

논리 뷰는 정적 구조 모델과 행동 모델을 정의한다.

정적 구조 모델은 시스템의 논리적인 구성과 컴포넌트의 논리적인 배치를 제시하는 정적 구조 다이어그램으로 구성된다. 정적 구조 다이어그램은 그림 13과 같이

1) 시스템 분류: 시스템이 속하는 도메인과 시스템 이름을 표현한다.
2) 아키텍처 스타일(패턴): 시스템에 대한 근본적인 구조적 구성 스키마를 표현한다[16].
3) 아키텍처 제약사항: 컴포넌트의 행동 또는 상호통신을 제어하는 중요한 규칙 혹은 원칙을 나타낸다.
4) 용어: 용어와 단어의 의미를 제공한다.

그림 8 윤곽 항목

1) 유즈케이스 명: 유즈케이스 이름을 기술한다.
2) 범위: 설계시에 블랙 박스로 사용되는 요소를 기술한다.
3) 사용 문맥(선택 사항): 필요하다면 어떠한 경우에 유즈케이스가 쓰이는지 기술한다.
4) 목적: 유즈케이스의 목적을 기술한다.
5) 전제조건: 유즈케이스가 기술되기 위한 선행 요건을 기술한다.
6) 관여자: 유즈케이스에 참여하는 관여자(Actor)를 기술한다.
7) 활성화조건: 유즈케이스가 시작되는 시점을 기술한다.
8) 사건의 흐름(주 흐름, 대체 흐름): 유즈케이스가 수행되는 단계, 흐름을 기술한다.
9) 사후조건(성공, 실패): 유즈케이스의 종료조건을 기술한다.
10) 다양성(선택사항): 기술적 측면 등 다양성의 경우를 기술한다.
11) 비기능성(선택사항): 유즈케이스의 비기능성을 기술한다.

그림 9 유즈케이스 서술항목

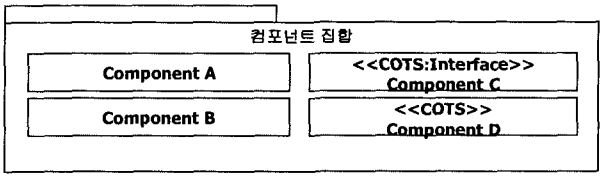


그림 10 컴포넌트 집합

UML의 패키지를 사용하여 시스템, 시스템과 관련된 서브시스템, 실수요자에게 제공되는 기능성등을 나타내며, 컴포넌트의 배치 정보를 커넥터를 포함한 UML의 협동 다이어그램으로 표현된다.

행동 모델은 컴포넌트의 외부 행동과 내부 행동을 나타내기 위해 정의한 요소로, 외부 행동은 UML의 순차 다이어그램으로 내부 행동은 UML의 상태 다이어그램으로 나타낸다. 그림 14는 행동 모델을 나타낸다.

1) 컴포넌트 명: 기초적인 정보로 외부적인 속성을 기술하기 위한 컴포넌트의 이름을 나타낸다. 외부 저장소로부터 가지고 온 컴포넌트라면 컴포넌트 명 앞에 <<COTS>>라는 스테레오 타입을 지정한다.
2) 프로그래밍 언어 또는 컴포넌트 타입: 컴포넌트가 COM/CORBA/WEB 컴포넌트인지에 대한 사항 혹은 컴포넌트를 작성한 프로그래밍 언어가 무엇인지에 대한 사항을 나타낸다.
3) 팀 혹은 벤더: 컴포넌트를 작성한 벤더나 팀 명, 혹은 추출된 컴포넌트일 경우는 개발해야 하는 팀, 벤더에 대한 사항을 나타낸다.
4) 비용: 외부 저장소에서 구입한 컴포넌트에 대한 비용 또는 구성하는데 필요한 비용을 나타낸다.
5) 비기능적 사항: 컴포넌트가 가질 수 있는 성능 특징 등 컴포넌트가 제공하는 비기능적 사항을 나타낸다.
6) 공유자원 사용: 컴포넌트가 사용되는데 필요한 공유되는 요소에 대한 정보를 나타낸다.
7) 인터페이스 템플릿: 컴포넌트에 대한 인터페이스 정보를 나타낸다. 인터페이스 템플릿은 인터페이스에 대한 [19,20]을 바탕으로 정의한 요소로 다음과 같은 항목을 가진다.
   - 인터페이스 명 : 인터페이스의 명을 기술한다.
   - 인터페이스 분류: 인터페이스가 제공하는 서비스, 요청하는 서비스에 대한 사항을 나타낸다
   - 기술 부분: 각 오퍼레이션, 시멘틱에 대한 설명이나 인터페이스에 대한 부수적인 설명이 필요한 경우 기술한다.
   - 시그너쳐: 인터페이스가 가지는 어트리뷰트, 이벤트, 리턴 타입 등을 기술한다.
   - 제약 사항: 인터페이스의 오퍼레이션간의 호출순서, 의존관계, 상호작용에서의 제약 사항등을 나타낸다.
   - 품질: 인터페이스가 제공하는 비기능적 사항을 기술한다.

그림 11 외부적으로 볼 수 있는 속성

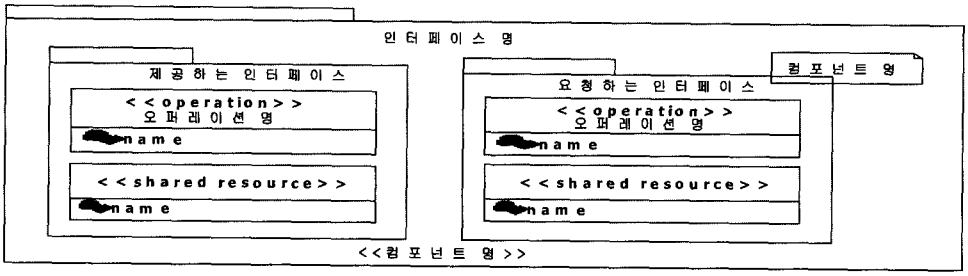


그림 12 인터페이스 명세 다이어그램

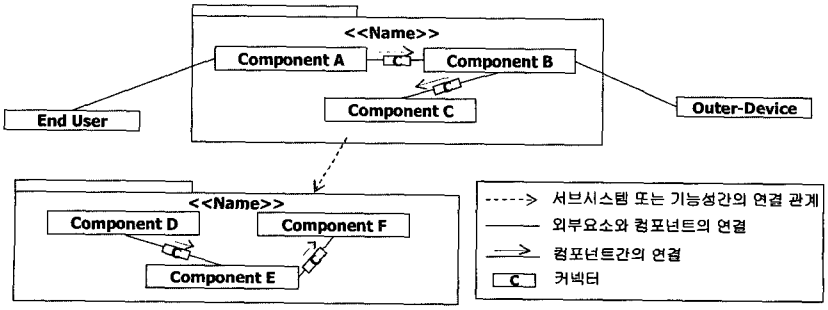


그림 13 정적 구조 다이어그램

3.2.3 조립 뷰

조립 뷰는 조립 모델을 정의한다.

조립 모델은 컴포넌트간의 상호연결 관계와 주고받는 정보를 표현하기 위해 그림 15와 같이 UML의 배치 다이어그램과, 협동 다이어그램을 사용해 조립 단위가 되는 상호 연결되는 컴포넌트를 표현한 동적 연결 다이어그램과, Credential[21,22]의 개념을 적용한 Credential 프로파일, 특정한 기능성을 제공하지는 않지만 컴포넌트 사이의 능률화된 상호 작용을 가능하게 하는 요소로 커넥터[23]에 대해 제시된 내용을 바탕으로 커넥터 항목을 정의한다. 또한 각 컴포넌트와 커넥터마다 번호를 통한 레이블을 주어 연결 및 조립에 있어서의 관계를 명확히 제시한다.

3.2.4 처리 뷰

처리 뷰는 처리 모델을 정의한다.

처리 모델은 컴포넌트로 구성된 전체적인 시스템을 프로세스와 쓰레드로 표현된 태스크로 나타낸다. 어트리뷰트와 오퍼레이션을 제외한 <<process>>, <<thread>>라는 스테레오 타입을 가진 UML의 클래스 다이어그램을 이용하여 정적인 구조를 제시하고, 상태 다이어그램으로 시스템의 병행성·동기화 정보를 표현한다.

3.2.5 물리 뷰

물리 뷰는 물리적 모델을 정의한다.

물리적 모델은 그림 16과 같이 외부는 컴포넌트를 배치하고 있는 하드웨어 장치·플랫폼등을 나타내며, 내부는 컴포넌트를 나타낸다. 또한 UML의 노트 표기를 이용해 컴포넌트를 배치하고 있는 하드웨어 장치·플랫폼 등의 정보가 그림 16과 같이 첨부한 배치 다이어그램으로 제시된다.

3.3 컴포넌트 기반 4+1 뷰 모델의 매핑 관계

컴포넌트 기반 4+1 뷰 모델에서 문맥 뷰는 다른 4가지 뷰의 기초 정보를 제공하는 뷰로서의 역할을 하며, 논리 뷰, 조립 뷰, 처리 뷰, 물리 뷰 사이에는 아키텍처 모델의 일관성을 유지시켜주기 위해 그림 17과 같은 매핑 관계를 정의한다.

논리 뷰의 정적 구조 다이어그램에서 제시된 컴포넌트의 논리적인 배치 및 연결 관계는 컴포넌트간의 상호연결 관계와 주고받는 정보를 표현한 조립 뷰의 동적 연결 다이어그램에 반영된다. 또한 논리 뷰의 정적 구조 다이어그램에서 컴포넌트가 배치된 정보는 처리 뷰의

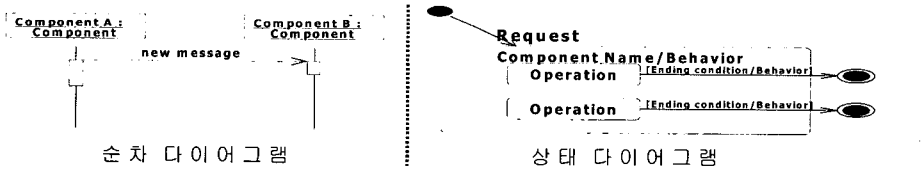


그림 14 순차/상태 다이어그램으로 표현되는 행동모델

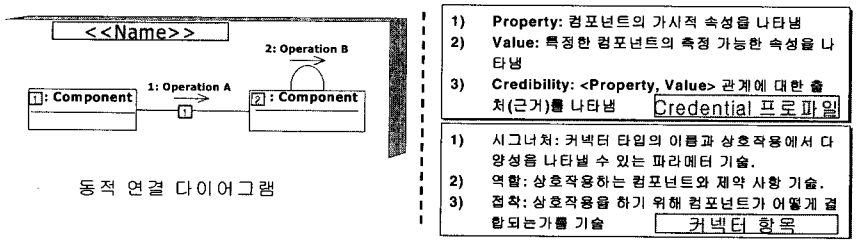


그림 15 조립 모델

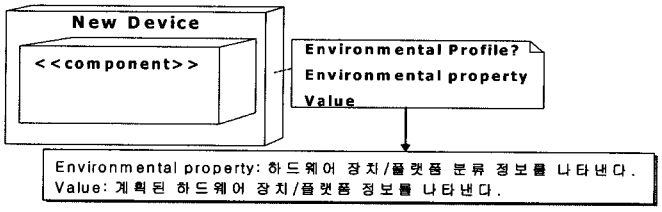


그림 16 배치 다이어그램

클래스 다이어그램/상태 다이어그램으로 구성된 처리 모델에서 하나의 프로세스나 쓰레드 단위의 정보로 반영된다. 그리고 처리 모델에서 반영되는 프로세스나 쓰레드로 표현되는 웹 서버, 어플리케이션 서버 등의 정보는 물리 뷰의 컴포넌트를 배치하고 있는 하드웨어 장치/플랫폼 정보에 반영된다.

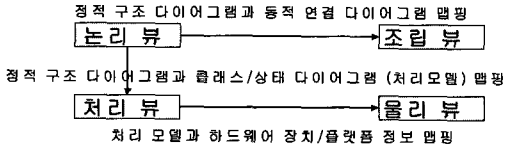


그림 17 매핑 관계

4. 사례연구

본 논문에서는 컴포넌트 기반 4+1 뷰 모델을 두께가 6㎜이상의 강판이며, 선박건조와 교량 및 각종 산업기계 등에 널리 쓰이는 후판의 저장, 이적, 출하 업무를 수행하는 아래와 같은 요구사항을 가진 후판 창고 관리시스템에 적용하여 표현하였다.

◆ 후판 창고 관리시스템 요구사항

후판 창고 관리시스템은 한 회사내의 전체 ERP 시스템 중 자재 관리의 하위 시스템으로 하나의 후판 창고에서 일어나는 전반적인 흐름을 관리하는 시스템이다. 후판 창고 관리 업무를 운영하기 위한 업무처리 내역은 다음과 같다.

- 후판 창고 관리시스템은 저장, 출하, 이적의 대상이 되는 후판 관리를 목적으로 후판에 대한 치수, 전담동 등 후판에 대한 기본적인 정보와 저장계획, 출하일정, 이적사항에 대한 정보를 관리한다.
- 운전실에서는 후판의 저장, 출하, 이적에 관한 모든 사항을 모니터링하고, 진행된 사항을 분석해서 분석사항을 저장한다. 운전실을 통해서 저장, 이적, 출하의 사항이 통제된다.
- 제품 출하, 이송, 반납, 선별 등 후판에 대한 이적 지시를 자재 관리시스템으로부터 통고받고, 이적업무를 수행한다. 이적에는 출하시 이적, 선별시 이적, 재고 정리시 이적, 동간 이적, 특정 제품 이적등의 종류가 있다.
- 후판 창고 관리시스템은 자재 관리시스템으로부터 후판에 대한 입고 예정 정보를 통고받고, 후판에 대한 저장 정보를 바탕으로 저장 업무를 수행한다.
- 후판 창고 관리시스템은 자재 관리시스템으로부터 후판에 대한 출고 예정 정보를 통고받고, 출하할 후판의 전담동 위치, 물량에 대한 정보를 바탕으로 후판 출하업무를 수행한다.
- 저장, 이적, 출하 업무의 수행 후 후판 창고 관리시스템은 창고의 자체 데이타베이스에 저장되어 있는 정보를 갱신하고, 갱신된 정보를 자재 관리시스템에 전달한다.

4.1 문맥 뷰

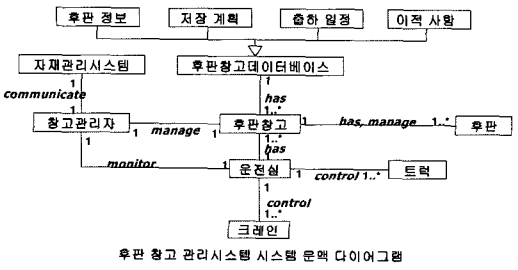
4.1.1 시스템 모델

후판 창고 관리시스템의 시스템 모델은 그림 18과 같다. 그림 18에 제시된 바와 같이 자재관리 시스템, 후판 창고 데이타베이스, 운전실 등 후판 창고 관리시스템과 연관된 개체간의 관계를 제시함으로써 후판 창고 관리시스템에 대한 전반적인 문맥을 제공하게 된다. 또한 그림 8에 제시된 윤곽 항목을 XML로 표현하기 위해 DTD를 정의한 예를 제시한다.

4.1.2 유즈케이스 모델

유즈케이스 다이어그램과 유즈케이스 서술로 구성된 유즈케이스 모델은 그림 19와 같다.

그림 19는 후판 창고 관리시스템의 유즈케이스 다이어그램과 유즈케이스 중 저장 계획 생성이라는 <<Manage goods receipts>>에 대해 그림 9에 정의된 요소로 기술한 예를 제시한다.



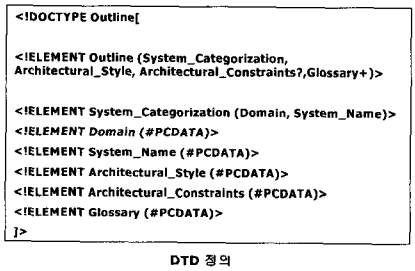


그림 18 후판 창고 관리시스템 시스템 모델

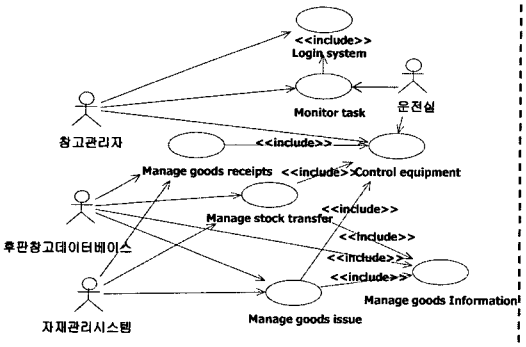
4.1.3 컴포넌트 구성 모델

컴포넌트 집합, 외부적으로 볼 수 있는 속성, 인터페이스 명세 다이어그램으로 정의되는 컴포넌트 구성 모델은 그림 20과 같다. 후판 창고 관리 시스템을 구성하는 컴포넌트 집합과 login 컴포넌트에 대해 인터페이스 명세 다이어그램과 외부적으로 볼 수 있는 속성 항목이 제시되어 있다.

4.2 논리 뷰

4.2.1 정적 구조 모델

패키지와 클래스 다이어그램으로 구성된 정적 구조 다이어그램으로 정의되는 후판 창고 관리 시스템 정적 구조 모델은 그림 21과 같다. 그림 21은 창고 관리시스템의 기능 중 로긴(Login system), 저장 계획 생성(Manage goods receipts), 이적 계획 생성(Manage stock transfer), 작업 모니터링(Monitor task)에 대해 나타낸 정적 구조 다이어그램이다.

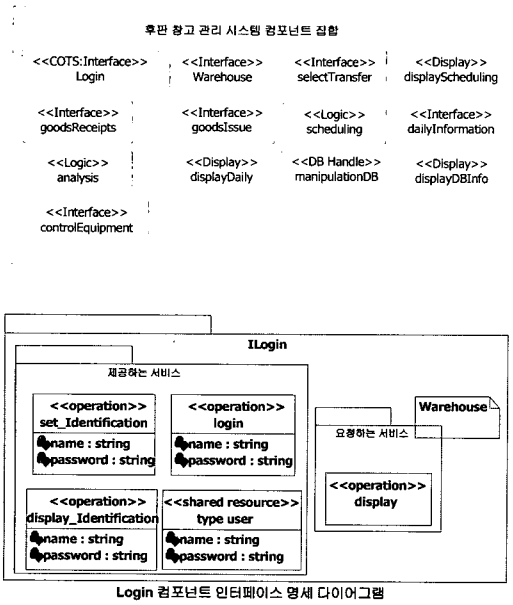


후판 창고 관리시스템 유즈케이스 다이어그램

유즈케이스 명	Manage goods receipts
범위	시스템은 "후판 창고 관리시스템"을 의미한다.
사용 문맥(선택 사항)	후판에 대한 저장 업무 계획을 생성할 경우 사용한다.
목적	후판 창고 내에서 후판 저장에 대한 업무를 수행한다.
관여자	자재 관리시스템, 창고 관리자, 후판 창고 데이터베이스
활성화 조건	창고 관리자는 후판의 치수, 종류, 수량에 대한 입고 예정 정보를 받고 시스템에 Login한다.
사건의 흐름 (주 흐름, 대체 흐름)	① 창고 관리자는 시스템에 후판의 입고 예정 정보를 입력한다. ② 시스템은 입력된 입고 예정 정보, 후판 데이터베이스에 저장된 정보를 바탕으로 후판의 저장 계획을 생성한다. ③ 시스템은 수립된 저장 계획을 화면에 출력한다.
사후 조건 (성공, 실패)	성공 조건: 후판의 저장 계획(후판 종류, 치수, 저장될 위치)이 시스템에 출력된다. 실패 조건: 후판 저장 계획 생성에 문제가 발생하면 에러 메시지와 원인을 출력한다.
다양성(선택 사항)	생성된 후판 저장 계획 출력 형태는 다양해 질 수 있다.
비기능성(선택 사항)	2초 이내에 저장 계획의 수립이 완료되어야 한다.

<<Manage goods receipts>> 유즈케이스 서술

그림 19 후판 창고 관리시스템 유즈케이스 모델



Login 컴포넌트 인터페이스 명세 다이어그램

컴포넌트 명	<<COTS>> Login
프로그래밍 언어 (컴포넌트 타입)	Web Component
팀(멘더)	Software Engineering
비용	5$
비기능적 사항	성능 특징: Client Web Component, Security
공유자원 사용	type user (String name, String password)
인터페이스 템플릿	
인터페이스 명	Ilogin
인터페이스 분류	제공하는 서비스(0), 요청하는 서비스(0)
제공하는 서비스	기술부분
Void set_Identification (string name, string password)	관리자로서 이름과 암호를 지정한다.
boolean login (string name, string password)	관리자로서의 권한을 인증한다.
Void display_Identification (string name, string password)	인증된 사용자인지 아니지를 출력한다.
요청하는 서비스	기술부분
void display()	후판 창고 관리 시스템 메인 화면을 출력한다.
시그너쳐	오퍼레이션: set_Identificaiton, login, display_Identification 어트리뷰트: name, password 리턴 타입: boolean (true, false)
제약 사항	Set_Identification→login→display_Identification→display
품질	오퍼레이션 set_Identification→ 보안 설정

Login 컴포넌트 외부적으로 볼 수 있는 속성

그림 20 후판 창고 관리 시스템 컴포넌트 구성 모델

4.2.2 행동 모델

행동 모델은 UML 순차 다이어그램과 상태 다이어그램으로 구성된다. 그림 22는 상태 다이어그램으로 저장, 이적, 출하 계획을 생성하는 scheduling 컴포넌트의 내부 행동을 나타낸다.

4.3 조립 뷰

4.3.1 조립 모델

동적 연결 다이어그램과 Credential 프로파일, 커넥터로 구성되는 조립 모델을 나타내면 그림 23과 같다. 그림 23은 컴포넌트와의 연결 관계를 제시하는 커넥터 중 scheduling 컴포넌트와 ManipulationDB 컴포넌트와의 연결관계를 제시하는 커넥터에 대한 항목을 정형화된 요소인 CSP(Communicating Sequential Processes)를 사용하여 표현한다. 또한 warehouse, scheduling 컴포넌트의 Credential 프로파일 표현을 예로 제시한다.

4.4 처리 뷰

4.4.1 처리 모델

UML의 클래스 다이어그램과 상태 다이어그램으로 구성되는 처리 모델은 그림 24와 같다.

4.5 물리 뷰

4.5.1 물리적 모델

UML의 물리적 모델은 UML 배치 다이어그램을 이용하여 그림 25와 같은 형태로 제시된다.

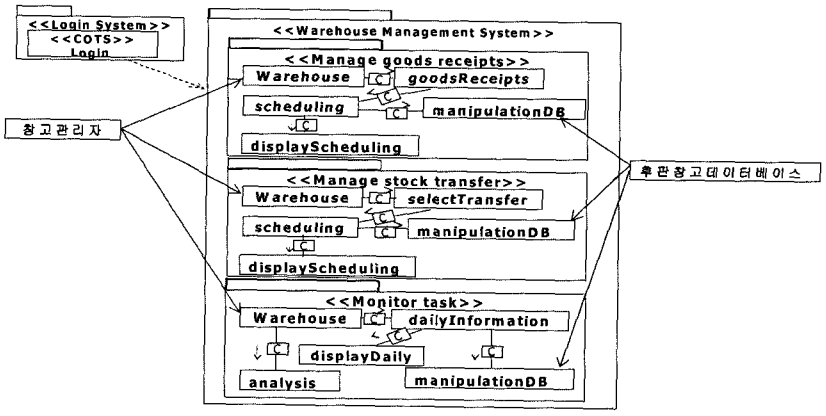


그림 21 후판 창고 관리시스템 정적 구조 모델

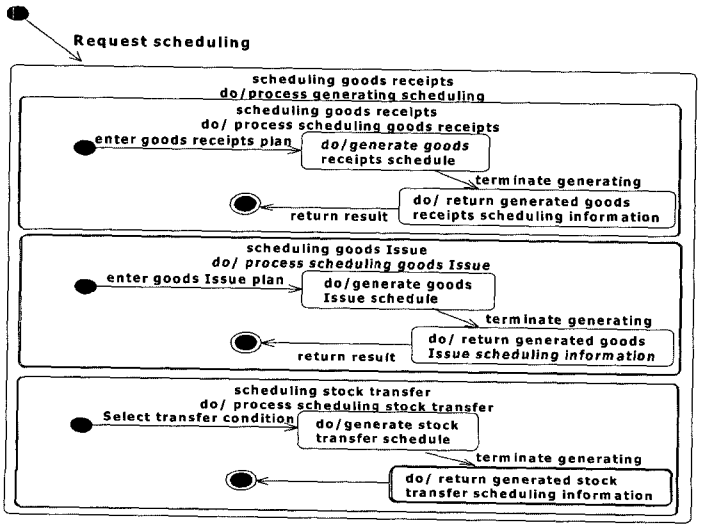


그림 22 scheduling 컴포넌트 상태 다이어그램

4.6 사례 적용 분석

본 절에서는 본 논문에서 제시한 컴포넌트 기반 4+1 뷰 모델의 사례 적용을 통해 Kruchten이 제시한 4+1 뷰 모델, Hofmeister등이 제시한 뷰 모델, Davis와 Williams 제시한 뷰 모델 등 기존 뷰 모델들과의 정성적 분석 결과를 표 1과 같이 제시한다.

표 1 정성적 분석

기존의 뷰 모델	컴포넌트 기반 4+1 뷰 모델
뷰의 관점 반영을 위한 뷰의 요소 명확	뷰의 관점 반영을 위한 뷰의 요소 명확히 정의
요소의 표현 방법에 대해 거의 제시하지 않음	요소의 표현방법 정의
CBD의 속성 반영에 모호성 존재	CBD를 기반으로 정의, 속성 명확히 반영
아키텍처 개발 및 표현방법, 요소 정의 등에 많은 시간 투자로 Time To Market 측면의 문제점	정의된 표현 형태 및 모델을 사용한 Time To Market 의 이점

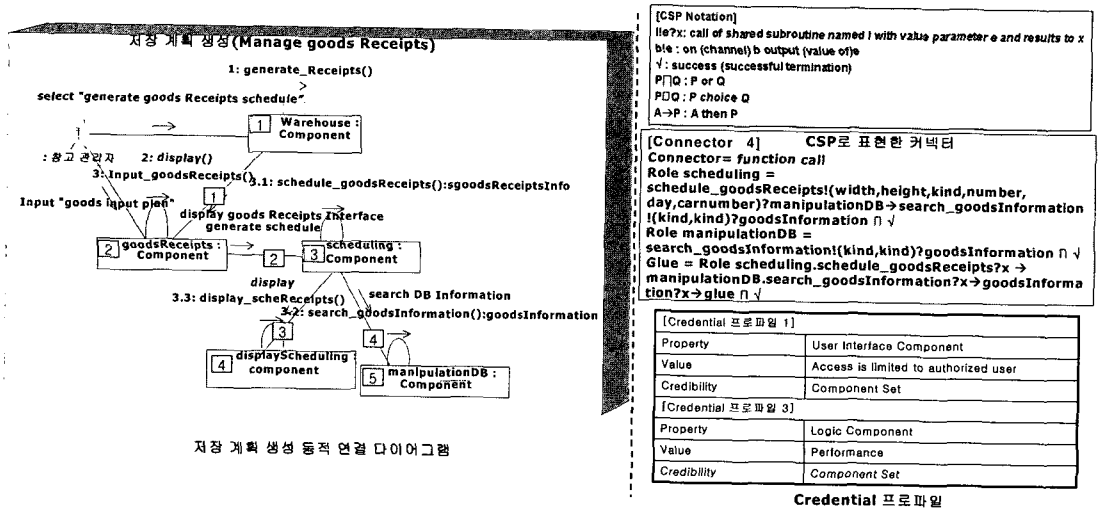


그림 23 조립 모델 표현 예

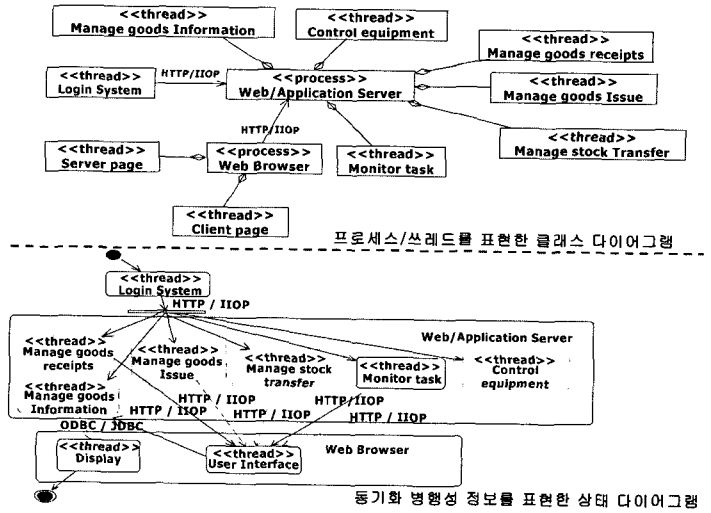


그림 24 처리 모델

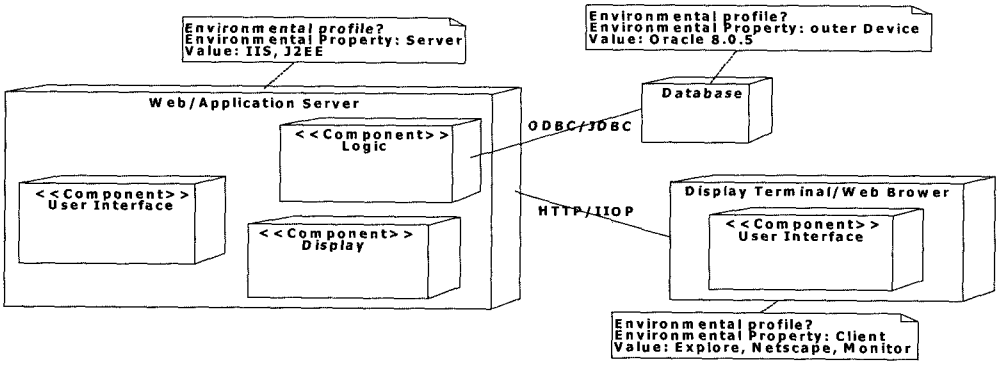


그림 25 후판 창고 관리시스템 물리적 모델

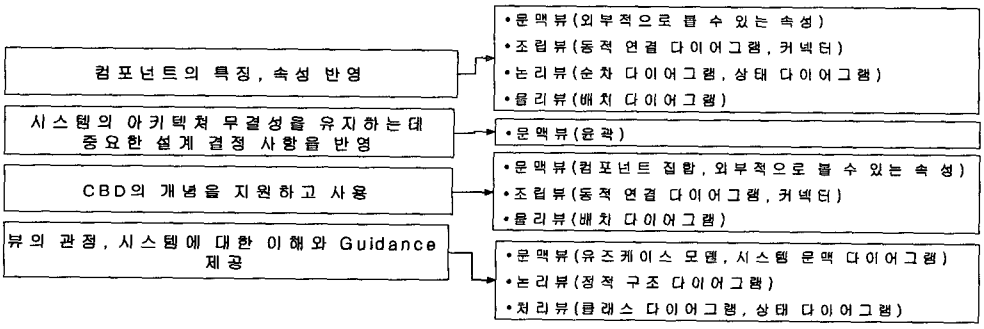


그림 26 컴포넌트 기반 4+1 뷰 모델 반영 측면과 뷰 요소

컴포넌트 기반 4+1 뷰 모델은 컴포넌트 기반 시스템 개발이라는 전제를 바탕으로 후판 창고 관리 시스템 사례 연구에서 제시된 바와 같이 1) 컴포넌트의 특징과 속성, 2) 시스템의 아키텍처 무결성을 유지하는데 중요한 설계 결정 사항, 3) CBD의 개념, 4) 뷰의 관점과 시스템에 대한 이해와 Guidance 제공 등의 측면이 아래 그림 26에 도식화된 것처럼 컴포넌트 기반 4+1 뷰 모델에 반영됨으로써 소프트웨어에 대한 이해와 컴포넌트간의 연결관계에 대한 정보 등 완전한 컴포넌트 기반 소프트웨어 구성에 대한 틀을 구축한다.

5. 결론 및 향후 연구방향

본 논문에서는 컴포넌트에 기반한 소프트웨어 개발을 지원하기 위한 소프트웨어 아키텍처 뷰 모델을 어떻게 정의하고, 어떠한 요소를 표현할 것인가에 초점을 두었다. 본 논문에서 제시한 컴포넌트 기반 4+1 뷰 모델은 Kruchten이 제시한 4+1 뷰 모델을 바탕으로 컴포넌트 기반으로 소프트웨어를 개발하기 위해 뷰 모델을 재 정의한 모델이다. 이러한 관점 분리를 통한 소프트웨어 아키텍처 뷰 모델은 CBD와 함께 접목됨으로써 컴포넌트의 개발자와 조립자가 독립적으로 존재해서 소프트웨어 구성이 가능한 CBD의 개념을 효율적으로 반영한다. 그리고 뷰 모델을 통해 컴포넌트가 소프트웨어에 사용되는 다양한 측면 반영으로 시스템 개발의 효율성과 컴포넌트의 대체로 발생하는 소프트웨어의 유지·보수의 효율을 극대화 할 수 있다.

재 정의된 뷰 모델을 바탕으로 현재 널리 쓰이는 모델링 언어인 UML을 도입하여 뷰 모델 요소의 표현에 대해 제시하였다. 뷰 모델 요소와 표현 방법을 함께 제시함으로써 컴포넌트 기반 4+1 뷰 모델을 사용하여 컴포넌트 기반의 소프트웨어 개발을 하는데 있어서 실용성과 사용의 편이성을 제공해주고, 아키텍처를 사용하는 Stakeholder 사이의 의사소통을 원활하게 촉진시킴으로써 소프트웨어 개발, 유지·보수 시간의 단축 효과를 얻는다.

향후 연구 방향으로는 제안한 컴포넌트 기반 4+1 뷰 모델을 응용 소프트웨어 개발에 적용하여, 응용 소프트 웨어 아키텍처를 만들고, 이에 기반한 소프트웨어 개발에 대한 연구 및 아키텍처 설계 방법에 대한 연구가 필요하다.

참 고 문 헌

[1] O. C., Kwon, S. J., Yoon, and G. S., Shin, "Component-Based Development Environment: An Integrated Model of Object-Oriented Techniques and Other Technologies", International Workshop on Component-Based Software Engineering, pp. 47-53, 1999.

[2] SEI in Carnegie Mellon University, "Component-Based Software Development/COTSIntegration", http://www.sei.cmu.edu/str/descriptions/cbsd_body.html

[3] 하현주, 문미경, 염근혁, "컴포넌트 기반 소프트웨어 개발을 위한 도메인 분석 및 설계 방법", 한국정보과학회 논문지(소프트웨어 및 응용),제 28권, 제 10호, pp. 743-756, 2001.

[4] OMG, "Unified Modeling Language(UML), Version 1.4", http://www.omg.org

[5] D.E., Perry, and A. L, Wolf, "Foundations for the Study of Software Architecture, ACM Software Engineering Notes", OCT, pp. 40-52, 1992.

[6] SEI in Carnegie Mellon University, "How Do You Define Software Architecture", http://www.sei.cmu.edu/architecture/definitions.html

[7] L., Bass, P., Clements, and R., Kazman, "Software Architecture in Practice", Addison-Wesley, 1998.

[8] The Architecture Working Group, "IEEE Recommended practice for architectural description of software-intensive systems", IEEE Std 1471-2000, pp. 1-23, 2000.

[9] J., Chen, "Architecture for Systematic Development of Mechatronics Software Systems", Licentiate Thesis, Royal Institute of Technology, Sweden, ISSN 1400-1179, ISRN KTH/MMK-01/06-SE.

[10] P., Kruchten, "The 4+1 View Model of Architecture", IEEE Software, Volume:12 Issue:6, Nov, pp. 42-50, 1995.

[11] D., Soni, R., Nord, and C., Hofmeister, "Software Architecture in Industrial Applications", Proceedings of the 17th international conference on software engineering, pp. 196-207, 1995.

[12] M., J. Davist, and R., B. Williams, "Software Architecture Characterization", Proceedings of the 1997 symposium on software reusability, pp. 30-38, 1997.

[13] M., Jung, and E., W. Biersack, "A Component Based Architecture For Software Communication Systems", Proceedings of Seventh IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp. 36-44, 2000.

[14] H., Mili, A., Mili, S., Yacoub, and E., Addy, "Organizations, Components, and Metrics: Foundations for an Engineering Discipline of Software Reuse", Software Reuse Technologies: 21. Component Based Software Development, http://www.info.uqam.ca/~mili/Enseignement/MIG8500-s00/book-outline.html

[15] M., Shaw, "Large Scale Systems Require Higher-Level Abstraction", Proceedings of Fifth International Workshop on Software Specification and Design, IEEE Computer Society, pp. 143-146, 1989.

[16] B., Frank, R., Meunier, H., Rohnert, P., Sommerlad, and M., Stal, "Pattern-Oriented Software Architecture A System of Patterns", John Wiley & Sons, 1996.

[17] A., Cockburn, "Writing Effective Use Cases", Addison-Wesley, 2000.

[18] G., Booch, J., Rumbaugh, and I., Jacobson, "The Unified Modeling Language User Guide", Addison-Wesley, 1998.

[19] J., Han, "A Comprehensive Interface Definition Framework for Software Components", In proceedings of the Asia-Pacific Software engineering Conference, pp. 102-111, 1998.

[20] Bredemeyer Consulting, "Action Guides for the Enterprise Architect", http://www.bredemeyer.com

[21] M., Shaw, "Truth vs Knowledge: The Difference Between What a Component Does and What We Know It Does", Proceedings of the 8th International Workshop on Software Specification and Design, pp. 181-185, 1996.

[22] K., Wallnau, and J., Stafford, "Ensembles: Abstractions for A New Class of Design Problem", In Proceedings of the IEEE 27th Euromicro Conference, pp. 48-55, 2001.

[23] R., Allen, and D., Garlan, "Beyond Definition/Use: Architectural Interconnection", Proceedings of ACM Workshop on Interface Definition Languages, pp. 35-45, 1994.

박 준 석
정보과학회논문지 : 소프트웨어 및 응용
제 30 권 제 5 호 참조

문 미 경
정보과학회논문지 : 소프트웨어 및 응용
제 30 권 제 5 호 참조

염 근 혁
정보과학회논문지 : 소프트웨어 및 응용
제 30 권 제 5 호 참조