소프트웨어공학특론

Distributed Vending Machine Software Architecture Document (SAD)

CONTENT OWNER:

전다윤

허윤아

김두리

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
Ver 1.0	Ver 1.0	21.06.14

Table of Contents

List of Figures

List of Tables

1 Documentation Roadmap

1.1 Document Management and Configuration Control Information

- Revision Number: version 1.0
- Revision Release Date: 21.06.14
- Purpose of Revision: Document overall architecture
- Scope of Revision: Overall system architecture

1.2 Purpose and Scope of the SAD

본 문서는 분산자판기(Distributed Vending Machine, DVM) 시스템을 위한 software architecture 를 구체화한다. 이 문서는 Software architecture 에 대한 모든 정보들을 찾을 수 있지만, 많은 정보들이 다른 문서들의 참조로 통합된다.

What is software architecture? The software architecture for a system[1] is the structure or structures of that system, which comprise software elements, the externally-visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. This definition provides the basic litmus test for what information is included in this SAD, and what information is relegated to downstream documentation.

Elements and relationships. The software architecture first and foremost embodies information about how the elements relate to each other. This means that architecture specifically omits certain information about elements that does not pertain to their interaction. Thus, a software architecture is an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. Elements interact with each other by means of interfaces that partition details about an element into public and private parts. Software architecture is concerned with the public side of this division, and that will be documented in this SAD accordingly. On the other hand, private details of elements—details having to do solely with internal implementation—are not architectural and will not be documented in a SAD.

[1] Here, a system may refer to a system of systems.

Multiple structures. The definition of software architecture makes it clear that systems can and do comprise more than one structure and that no one structure holds the irrefutable claim to being the architecture. The neurologist, the orthopedist, the hematologist, and the dermatologist all take a different perspective on the structure of a human body. Ophthalmologists, cardiologists, and podiatrists concentrate on subsystems. And the kinesiologist and psychiatrist are concerned with different aspects of the entire arrangement’s behavior. Although these perspectives are pictured differently and have very different properties, all are inherently related; together they describe the architecture of the human body. So it is with software. Modern systems are more than complex enough to make it difficult to grasp them all at once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system’s structures. To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment—which *view* we are taking of the architecture. Thus, this SAD follows the principle that documenting a software architecture is a matter of documenting the relevant views and then documenting information that applies to more than one view.

예를 들어, 중요하지 않은 모든 소프트웨어 시스템은 구현 단위로 분할된다. 이러한 단위에는 특정 책임이 부여되며 프로그래밍 팀의 작업 할당의 기초가 되고이러한 종류의 요소는 다른 구현 단위의 소프트웨어가 호출하거나 액세스 할 수 있는 프로그램 및 데이터와 개인용 프로그램 및 데이터로 구성된다. 대규모 프로젝트에서 요소는 하위 팀에 할당하기 위해 세분화된다.

시스템의 기능이 분할되고 구현 팀에 할당되는 방식에 초점을 맞춘 매우 정적 인 구조다. 다른 구조는 시스템 기능을 수행하기 위해 런타임에 요소가 서로 상호 작용하는 방식에 훨씬 더 중점을 두며 시스템이 일련의 병렬 프로세스로 구축된다고 가정한다. 런타임에 존재하는 일련의 프로세스, 각 프로세스를 형성하기 위해 순차적으로 결합 된 이전에 설명 된 다양한 구현 단위의 프로그램 및 프로세스 간의 동기화 관계는 시스템을 설명하는 데 자주 사용되는 또 다른 종류의 구조를 형성한다.

이 구조들은 아키텍처 정보를 전달하지만 아키텍처라고 할 수 없다. 아키텍처는 이러한 구조와 기타 여러 구조로 구성된다. 이것은 아키텍처가 여러 종류의 요소(구현 단위 및 프로세스), 요소 사이의 상호작용, 하나이상의 컨텍스트 등 여러 종류의 구조를 구성 할수 있다는 것을 보여준다.

이러한 구조는 섹션 3 에서 제공되는 소프트웨어 아키텍처의 view 에서 설명한다.

Behavior. 소프트웨어 아키텍처는 구조적인 정보에 초점을 맞추는 경향이 있지만 각 요소의 동작은 다른 요소의 관점에서 관찰/식별 할 수 있다. 이 동작을 통해 요소가 서로 상호 작용하고 SAD 에 문서화 된다. 동작은 각각 view 의 요소 카탈로그에 문서화 되어있다.

1.3 How the SAD Is Organized

This SAD is organized into the following sections:

- **Section 1 ("Documentation Roadmap") provides information about this document and its intended audience**. It provides the roadmap and document overview. Every reader who wishes to find information relevant to the software architecture described in this document should begin by reading Section 1, which describes how the document is organized, which stakeholder viewpoints are represented, how stakeholders are expected to use it, and where information may be found. Section 1 also provides information about the views that are used by this SAD to communicate the software architecture.
- **Section 2 ("Architecture Background") explains why the architecture is what it is.** It provides a system overview, establishing the context and goals for the development. It describes the background and rationale for the software architecture. It explains the constraints and influences that led to the current architecture, and it describes the major architectural approaches that have been utilized in the architecture. It includes information about evaluation or validation performed on the architecture to provide assurance it meets its goals.
- **Section 3 (Views") and Section 4 ("Relations Among Views") specify the software architecture.** Views specify elements of software and the relationships between them. A view corresponds to a viewpoint (see Section 1.5), and is a representation of one or more structures present in the software (see Section 1.2).
- **Sections 5 ("Referenced Materials") and 6 ("Directory") provide reference information for the reader.** Section 5 provides look-up information for documents that are cited elsewhere in this SAD. Section 6 is a directory, which is an index of architectural elements and relations telling where each one is defined and used in this SAD. The section also includes a glossary and acronym list.

1.4 Stakeholder Representation

Table 1 각 stakeholder 별 아키텍쳐 문서의 필요성

Stakeholder	Description	Purpose of Using Architectural Documentation
User	시스템의 최종 사용자이다. 사용자가 얼마나 쉽고 빠르게 원하는 동작을 수행할 수 있는지 평가할 수 있다.	1) 원하는 기능의 유무 확인
Implementer	Design, Requirement, Architecture 문서의 요구대로 시스템 중 소프트웨어를 개발한다.	2) 시스템에 대한 이해 3) 개발의 제약사항 파악
Maintainer	개발 이후 소프트웨어의 유지보수를 담당하는 사람이다. 시스템에 생기는 버그를 찾아 수정하고, 특정 성능을 개선한다.	2) 시스템에 대한 이해 4) 연결된 컴포넌트들 확인 가능
Testers	시스템과 시스템의 각 요소를 아키텍처 문서와 시스템 요구사항에 대해 테스트한다.	2) 시스템에 대한 이해 5) 테스트 케이스 생성
Project manager	시스템을 개발하기 위한 계획과 순서, 일정, 예산, 역할 분배, 테스트 등을 관리한다.	6) 예산 편성 요건 확인 7) 시스템 개발 및 테스트 계획 수립
Customer	시스템에 대한 비용을 지불 후 원하는 기능을 요구하여 시스템을 제공받는다.	1) 원하는 기능의 유무 확인 8) 원하는 품질 달성 여부 확인

		9) 진행 상황 확인
Software architect	Architecture 개발과 문서화를 담당한다.	10) Architecture 수립 시 중요한 요소 파악
COTS specialist	시스템과 상호작용할 수 있는 COTS 를 알아보고 그 시스템과의 인터페이스를 관리한다.	11) COTS 필요성 확인 12) COTS 와의 인터페이스 확인

Table 2 각 stakeholder 별 concern 의 중요도

Concern / Stakeholder	1	2	3	4	5	6	7	8	9	10	11	12
User	H	L	L	L	L	L	L	M	L	L	L	L
Implementer	L	H	H	M	L	L	M	M	M	L	L	M
Maintainer	L	H	M	H	L	L	L	M	L	L	L	M
Testers	L	H	L	L	H	L	M	M	M	L	L	M
Project manager	M	M	M	M	M	H	H	M	M	L	M	M
Customer	H	M	L	L	L	M	M	H	H	L	M	L
Software architect	L	M	M	M	L	L	L	M	L	H	M	M
COTS specialist	L	M	M	M	L	M	L	L	L	L	H	H

*H = high, M = medium, L = low

1.5 Viewpoint Definitions

SAD 는 ANSI / IEEE 1471-2000 에서 요구하는 바와 같이, 소프트웨어 집약적 시스템의 아키텍처 문서화에 권장되는 모범 사례 [IEEE 1471]에 따라 아키텍처 문서에 대한 이해 당사자 중심의 다중 뷰 접근 방식을 사용한다.

섹션 1.2 에 설명 된 대로 소프트웨어 아키텍처는 둘 이상의 소프트웨어 구조로 구성되며 각 구조는 서로 다른 시스템 품질에 대한 엔지니어링 핸들을 제공한다. 뷰는 이러한 구조 중 하나 이상을 지정하고 소프트웨어 아키텍처를 문서화하는 것은 관련 뷰를 문서화 한 다음 하나 이상의 뷰에 적용되는 정보를 문서화한다.

ANSI / IEEE 1471-2000 은 stakeholder 의 관심을 가져옴으로써 문서화 할 view 를 선택하기 위한 지침을 제공한다. Stakeholder 커뮤니티를 만족시키기 위해 view 를 정의하고 concern 을 식별하며 모든 viewpoint 에서 사용되는 모델링 기법, 평가 기법, 일관성 검사 기법 등을 확인한다. 따라서 view 는 시스템에 적용되는 viewpoint 다. 함께 선택한 뷰 세트는 전체 아키텍처와 모든 관련 속성을 보여주고. SAD 에는 시스템에 대한 전체적인 설명을 제공하기 위해 둘 이상의 view 에 적용되는 viewpoint, 관련 view 및 정보가 포함됩니다.

섹션 1.5 의 나머지 부분은 이 SAD 에서 사용되는 view 를 정의한다. 다음 표에는이 프로젝트의 stakeholder 와 우려 사항을 concern 을 해결하기 위한 view 가 설명되어 있습니다.

Table 3: Stakeholders and Relevant Viewpoints

Stakeholder	Viewpoint(s) that apply to that class of stakeholder’s concerns
User	1) Decomposition & Layered style
Implementer	2) Decomposition & Layered style, peer-to-peer style 3) Install style, Deployment style
Maintainer	2) Decomposition & Layered style, peer-to-peer style 4) Decomposition & Layered style
Testers	2) Decomposition & Layered style, peer-to-peer style

Stakeholder	Viewpoint(s) that apply to that class of stakeholder's concerns
	5) Decomposition & Layered style, Deployment style
Project manager	6) Deployment style, Work Assignment style 7) Work Assignment style
Customer	1) Decomposition & Layered style 8) Deployment style, Peer-to-Peer style 9) Work Assignment style
Software architect	10) Decomposition & Layered style, Peer-to-Peer style, Deployment style, Install style, Work Assignment
COTS specialist	11) Decomposition & Layered style, Deployment style 12) Decomposition & Layered style

1.5.1 Module Decomposition & Layered Viewpoint Definition

1.5.1.1 Abstract

시스템을 Layer 단위로 분해한 후, 각 Layer 내부를 모듈, 혹은 서브-모듈 단위로 분해하여 시스템의 responsibility 가 어떻게 각 Layer, 그리고 각 모듈마다 나누어져 있는지 파악하는 데에 사용한다.

1.5.1.2 Stakeholders and Their Concerns Addressed

Table 4 Module View - Stakeholders and Their Concerns Addressed

Stakeholders	Concerns addressed
User	원하는 기능의 유무를 확인할 수 있다.

Implementer	시스템을 이해할 수 있다
Maintainer	시스템을 이해할 수 있다. 연결된 컴포넌트를 알 수 있다.
Tester	시스템을 이해할 수 있다. 테스트 케이스를 생성 할 수 있다.
Customer	원하는 기능의 유무를 확인할 수 있다.
Software architect	Architecture 수립 시 중요한 요소를 파악할 수 있다.
COTS specialist	COTS 필요성을 확인할 수 있다. COTS 와의 인터페이스를 확인할 수 있다.

1.5.1.3 Elements, Relations, Properties, and Constraints

Table 5 Module View - Elements, Relations, Properties, and Constraint

Elements	Layer, Module
Relations	각 module 은 layer 와 ‘is-part-of’의 관계 형성(Layer⊃Module) 자세한 criteria 는 3 장에서 서술한다.
Properties	Module 과 Layer 의 property 는 3 장의 view 에서 서술한다.
Constraint	Loop 가 있으면 안 된다. Module 은 하나의 부모만 가지고 있어야 한다.

1.5.1.4 Language(s) to Model/Represent Conforming Views

UML 을 통해 그리고, 자연어를 통해 description 을 작성한다.

1.5.1.5 Applicable Evaluation / Analysis Techniques and Consistency / Completeness Criteria

Consistency/Completeness Criteria 는 아래와 같다.

a) 한 개의 module 은 하나 이상의 layer 에 속하지 않는다.

b) 한 개의 module 은 한 개의 기능을 제공한다.

c) 하나의 layer 는 캡슐화 되어 다른 layer 에서 접근 시 요청에 의해서만 접근한다.

d) 한 개의 layer 는 한 개 이상의 module 을 포함한다.

e) 모든 source code 는 분해된 각 모듈에 mapping 된다.

f) 모든 layer 와 모듈의 기능을 결합하여 하나의 시스템을 구성한다.

Applicable Evaluation/Analysis Techniques 는 아래와 같다.

a) ATAM 과 같은 evaluation technique 을 사용하여 해당 architecture 를 사용하기 위해서 경제적으로 지원이 가능한지 평가한다.

1.5.1.6 Viewpoint Source

[Clements 2010, Section 2.2] describes the module decomposition style, which corresponds in large measure to this viewpoint.

1.5.2 C&C Peer-to-peer Viewpoint Definition

1.5.2.1 Abstract

1.5.2.2 Stakeholders and Their Concerns Addressed

Table 6 Peer-to-Peer View - Stakeholders and Their Concerns Addressed

Stakeholders	Concerns addressed
Implementer	시스템을 이해할 수 있다
Maintainer	시스템을 이해할 수 있다.
Tester	시스템을 이해할 수 있다
Customer	원하는 품질 달성 여부를 확인할 수 있다.
Software architect	Architecture 수립 시 중요한 요소를 파악할 수 있다.

1.5.2.3 Elements, Relations, Properties, and Constraints

Table 7 Peer-to-Peer View - Elements, Relations, Properties, and Constraint

Elements	Peer Component , Call-return Connecter
Relations	attachment
Properties	Peer 간의 interaction 과 관련된 protocol 을 강조하고 performance 에 의해 영향을 받는다. Attachment 는 작동 중에 바뀔 수 있다.
Constraint	-

1.5.2.4 Language(s) to Model/Represent Conforming Views

UML 을 통해 그리고, 자연어를 통해 description 을 작성한다.

1.5.2.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

Consistency/Completeness Criteria 는 아래와 같다.

a) 하나의 peer 는 동시에 여러 peer 와 통신하지 않는다.

b) Peer 가 추가될 때 확장성(scalability)가 보장된다.

c) Service 를 요청하여 다른 요소와 상호작용 해야 한다.

1.5.2.6 Viewpoint Source

[Clements 2010, Section 2.4] describes the C&C Peer-to-Peer style, which corresponds in large measure to this viewpoint.

1.5.3 Allocation Deployment Viewpoint Definition

1.5.3.1 Abstract

Deployment Style 에서 C&C Style 에서 나온 Software Element 를 컴퓨터 플랫폼의 하드웨어에 적용한다. 위로 인해 필요한 소프트웨어의 요소 특성이 하드웨어 요소의 특성에 의해 충족된다.

1.5.3.2 Stakeholders and Their Concerns Addressed

Table 8 Deployment View - Stakeholders and Their Concerns Addressed

Stakeholders	Concerns addressed
Implementer	개발의 제약사항을 파악할 수 있다
Tester	테스트 케이스 생성할 수 있다
Customer	원하는 품질 달성 여부 확인할 수 있다
Software architect	Architecture 수립 시 주요 요소를 파악할 수 있다

COTS specialist	COTS 의 필요성을 확인할 수 있다
Project Manager	예산 편성 요건을 확인할 수 있다

1.5.3.3 Elements, Relations, Properties, and Constraints

Table 9 Deployment View - Elements, Relations, Properties, and Constraint

Elements	Elements of C&C View, Hardware of the computing platform (processor, memory, disk, network)
Elements Properties	Elements of C&C View: Software Element 로서 예상되는 Processing, Memory, 요구사항 수행정도, 오류의 저항성 이다 Hardware of the computing platform : Allocation 에 영향을 미치는 하드웨어적 측면
Relations	Allocate-to ,Migrates-to, copy-migrates-to, execution-migrates- to (dynamic allocation options) Properties: the trigger causing the migration
Relation Properties	Allocate-to : Physical Unit 이 있는 Software element 을 표현한다. Allocation 이 Dynamic 하게 바꿀 수 있는지 알려준다. Migrates-to, copy-migrates-to, execution-migrates- to : 이동의 Trigger 를 표현한다.
Constraint	Topology 에 대한 제한은 없다. 그러나 소프트웨어에서 요구되는 특성이 하드웨어에서 제공된 특성으로 만족되어야 한다.

1.5.3.4 Language(s) to Model/Represent Conforming Views

UML 을 통해 그리고, 자연어를 통해 description 을 작성한다.

1.5.3.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

Consistency/Completeness Criteria: SW component 와 HW component 가 중복 없이 mapping 된다.

1.5.3.6 Viewpoint Source

[Clements 2010, Section 2.5] describes the Allocation deployment style, which corresponds in large measure to this viewpoint.

1.5.4 Allocation Install Viewpoint Definition

1.5.4.1 Abstract

C&C style 의 구성 요소를 production environment 의 파일 관리 시스템에 할당한다. 소프트웨어 시스템이 구현되면 결과 파일을 패키지화하여 production platform 에 설치한다.

1.5.4.2 Stakeholders and Their Concerns Addressed

Table 10 Install View - Stakeholders and Their Concerns Addressed

Stakeholders	Concerns addressed
Implementer	개발하는데 제약사항을 확인할 수 있다.
Software architect	진행상황을 확인 할 수 있다.

1.5.4.3 Elements, Relations, Properties, and Constraints

Table 11 Install View - Elements, Relations, Properties and Constraint

Elements	- Software element: a C&C component - Environment element: a configuration item

Relations	Component 는 configuration item 에 ‘allocated to’ 되어 있다. 하나의 configuration item 은 다른 하나에 ‘contained to’ 되어 있다.
Properties	configuration items 에 대한 소프트웨어 allocation 에 영향을 주는 속성
Constraint	파일과 폴더는 ‘is-contained-in’관계에 따라 트리 구조로 이루어져 있다.

1.5.4.4 Language(s) to Model/Represent Conforming Views

UML 을 통해 그리고, 자연어를 통해 description 을 작성한다.

1.5.4.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

시스템 파일이 하나의 패키지로 구현된다.

1.5.4.6 Viewpoint Source

[Clements 2010, Section 2.5 describes the allocation install style, which corresponds in large measure to this viewpoint.

1.5.5 Allocation Work Assignment Viewpoint Definition

1.5.5.1 Abstract

Work Assignment style 은 working style 을 위한 소프트위어의 주요 unit 과 이를 생산하는 사람, 소프트웨어가 개발되는 도구 및 환경을 보여준다. 팀 리소스 할당을 계획 및 관리하며 책임을 할당하고 프로젝트 구조를 설명하는데 도움이 된다.

1.5.5.2 Stakeholders and Their Concerns Addressed

Table 12 Work Assignment View - Stakeholders and Their Concerns Addressed

Stakeholders	Concerns addressed

Project manager	예산을 편성하고 확인할 수 있다.
Customer	진행상황을 확인할 수 있다.
Software architect	아키텍처 개발 시 중요한 요소에 대해 알 수 있다.

1.5.5.3 Elements, Relations, Properties, and Constraints

Table 13 Work Assignment View - Elements, Relations, Properties, and Constraint

Elements	Software element: a module Environment element: an organization unit
Relations	Allocated-to :software element 가 organization unit 에 ‘allocate to’되어 있다.
Properties	Properties of the software elements : a description of the required skill set Properties of the people elements: provided skill sets
Constraint	일반적으로 allocation 은 제한되어 있지 않지만 하나의 모듈이 하나의 organization unit 에 할당되도록 한다.

1.5.5.4 Language(s) to Model/Represent Conforming Views

UML 을 통해 그리고, 자연어를 통해 description 을 작성한다.

1.5.5.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

시스템을 구분 할 수 있어야한다.

1.5.5.6 Viewpoint Source

[Clements 2010, Section 2.5] describes the allocation work assignment style, which corresponds in large measure to this viewpoint.

1.6 How a View is Documented

3 장에서 각 view 를 모든 stakeholder 에 대해서 작성할 것이다.

1.7 Relationship to Other SADs

해당 없음

1.8 Process for Updating this SAD

추후 변경사항이 생길 경우 본 문서 작성자에게 요청 필요

2 Architecture Background

2.1 Problem Background

2.1.1 System Overview

분산 자판기 시스템은 품절된 메뉴에 대해 재고가 있는 가장 가까운 자판기를 알려줄 수 있고, 이 음료에 대한 선결제가 가능하다. 또한 평범한 자판기처럼 카드를 통한 음료 구매가 가능하다

2.1.2 Goals and Context

- Goal
  - 여러 stakeholder 의 요구사항을 파악하고, 이를 만족시킨다.
  - 개발 전 시스템에 대한 전체적인 이해를 돕는다.
  - 경제적으로 시스템을 개발할 수 있도록 한다.
- Context
  - 본 시스템은 iterative 개발 환경에서 개발한다.
  - 본 문서는 architect experience 를 위해 작성한다.
  - 본 시스템의 Primary functionality 는 2.1.3 절과 같다.
  - QAW 를 통해 도출된 Architecturally significant requirement 는 다음과 같다.

Table 14 Primary Functionality

No.	Use-Case	Description
UC-1	Select Item	사용자가 선택한 메뉴의 재고가 있을 경우 사용자에게 '구매하시겠습니까?' 등의 메시지로 알리고, 재고가 없을 경우 사용자에게 '다른 자판기의 재고를 찾아보시겠습니까?' 등의 메시지로 알린다.
UC-3	Search Other DVM	사용자가 다른 자판기에서 재고를 찾겠다고 선택했을 때, 선택된 상품의 재고 여부를 가장 가까운 자판기부터 순서대로 요청하면, 메시지를 받은 자판기는 상품의 재고 여부를 확인한 후 메시지로 응답한다. 재고가 있는 경우, 사용자에게 해당 자판기의 위치를 알리고, 선결제를 진행할 것인지 묻는다.
UC-4	Provide Verification Code	사용자가 선결제를 하면 인증코드를 생성하고 사용자와 해당 자판기에 제공한다.
UC-5	Verify Verification Code	사용자가 다른 자판기에서 선결제를 통해 받은 인증코드를 입력하면, 제공받았던 인증코드와 동일한지 그 유효성을 확인한다. 유효한 인증코드라면, 사용자에게 해당 상품을 제공한다.

Table 15 Constraints

ID	Constraint
CON-1	여러 자판기에서 같은 동작이 일어날 수 있어야 한다.
CON-2	네트워크는 낮은 대역폭을 가질 수 있지만, 신뢰할 수 있어야 한다.

CON-3	매번 동일한 환경에서 동일한 동작을 해야 한다.

Table 16 Architectural Concerns

ID	Concern
CRN-1	전체 초기 System Structure 를 수립해야 한다.
CRN-2	구성원이 System Domain 에 대한 이해도를 높여야 한다.

2.1.3 Significant Driving Requirements

QAW 를 통해 작성한 quality attribute 는 아래 표와 같다.

Table 17 Quality Attribute

ID	Quality Attribute	Scenario
QA-1	Usability	사용자는 자판기에서 원하는 상품을 뽑기 위해 버튼을 누른다. 자판기는 재고가 있는 경우 결제 이후 최소 5 초 안에 상품을 제공하고 재고가 없는 경우 최소 30 초 안에 다른 자판기의 재고를 30 초 안에 파악하여 안내한다.
QA-2	Marketability	마케터는 출시 이전 분산 자판기에 대한 설문조사를 한다. 설문조사를 통해 사용자들의 니즈를 파악하여 DVM 의 시장 경쟁성을 높인다.
QA-3	Reusability	코드 안에서 코드의 중복성을 최소화하기 위하여 정적분석을 한다. 정적분석을 통하여 코드의 중복성을

		최대 10%로 줄여 다른 시스템이나 애플리케이션에서 해당 시스템을 활용할 수 있도록 한다.
QA-4	Performance	하나의 자판기에서 여러 자판기로부터 최대 4 개의 메시지를 수신한다. 메시지를 수신한 자판기는 60 초 이내에 메세지를 모두 처리한다.
QA-5	Maintainability	3 명의 개발자가 하나의 컴포넌트 코드를 수정하고자 할 때, 3 시간 이내에 수정 가능하다.
QA-6	Testability	테스터가 12 시간 이내로 테스트를 위한 가상의 환경을 구축한다. 테스터가 테스트를 위한 가상의 환경이 구축된 상태에서 코드를 테스트하고 자 할 때, 6 시간 이내로 85%이상 테스트한다.
QA-7	Security	외부시스템에서 시스템 탈취를 시도하거나 네트워크로 전달되는 메세지에 접근하고자 할 때 , 30 초 이내에 접근을 감지 후, 5 초 이내에 관리자에게 위험 메세지를 전달한다.

2.2 Solution Background

2.2.1 Architectural Approaches

- Module View
  - Decomposition & Layered View 를 충족시키기 위하여 Rich Client Application Architecture 를 사용하였다. Rich Client application 은 자판기에 설치되는 프로그램의

개발에 도움을 준다. 또한, QA-1 의 usability 를 향상시킬 수 있고, 사용자의 요청이 있을 때만 네트워크에 접속하면 된다는 점에서 네트워크 비용을 아낄 수 있다.

- C&C View
  - Peer-to-Peer View 를 충족시키기 위하여 Forwarder-Receiver Pattern 을 사용하였다. 각 DVM 간 peer-to-peer 통신을 하도록 하면 DVM 간 통신에 있어서 효율적인 inter-process communication 이 가능하고 IPC 기능을 encapsulate 이에 가장 적절한 forwarder-receiver pattern 을 사용한다.
- Allocation View
  - Install View, Deployment View 를 충족시키기 위하여 Non-distributed Deployment Pattern 을 사용하였다. CON-1 에 따라 여러 자판기에서 동일한 동작을 하도록 할 필요가 있고, QA-1 에 따라 사용자에게 빠르게 서비스를 제공하도록 할 필요가 있으므로 DVM 자체를 하나의 server 로 고려하는 non-distributed deployment pattern 을 사용한다.
  - Deployment View 를 충족시키기 위하여 COTS 를 사용하였다. 외부 결제 시스템을 활용함으로써 개발 비용을 줄일 수 있다.
  - Work Assignment View 를 충족시키기 위하여 Rich Client Application Architecture 를 사용하였다. 각 layer 내부에 모듈을 포함하므로, 각 모듈에 어떻게 implementer 를 assign 할 지 결정할 수 있다.

2.2.2 Analysis Results

ATAM 을 사용하지 않음

2.2.3 Requirements Coverage

ADD 3.0 의 1st iteration 을 통해 전체 시스템의 architecture 를 수립하였다.

ADD 3.0 의 2nd iteration 을 통해 functionality 를 충족하기 위한 architecture 를 수립하였다.

ADD 3.0 의 3rd iteration 을 통해 quality attribute 와 기타 architecturally significant requirement 를 충족하기 위한 architecture 를 수립하였다.

2.2.4 Summary of Background Changes Reflected in Current Version

해당 없음

2.3 Product Line Reuse Considerations

추후 시스템의 일정 부분을 재사용할 수 있도록 하기 위해서, 시스템을 모듈 단위로 분해하였고, 각 기능이 중복되지 않도록 하였다. 또한, 시스템의 확장성을 높이기 위하여 Peer-to-Peer 방식을 선택하였다.

3 Views

Table 18 여러 view 의 종류

Name of view	Viewtype that defines this view	Types of elements and relations shown		Is this a Module View?	Is this a Component & Conecter View?	Is this an Allocation View?
Decomposition & Layered View	Logical View	Module, Layer	'is-part-of' 'allowed-to-use'	O		
Peer-to-Peer View	Process View	Peer Component, Call-Return Connector	Attachment		O	
Deployment View	Development View	Elements of C&C View, Hardware of the computing platform	Allocate-to ,Migrates-to, copy-migrates-to, execution-migrates- to			O
Install View	Development View	A C&C component , a configuration item	Allocated-to, Containment			O
Work Assignment	Development View	Module, organization unit	Allocated-to			O

3.1 Module Decomposition & Layered View

3.1.1 View description

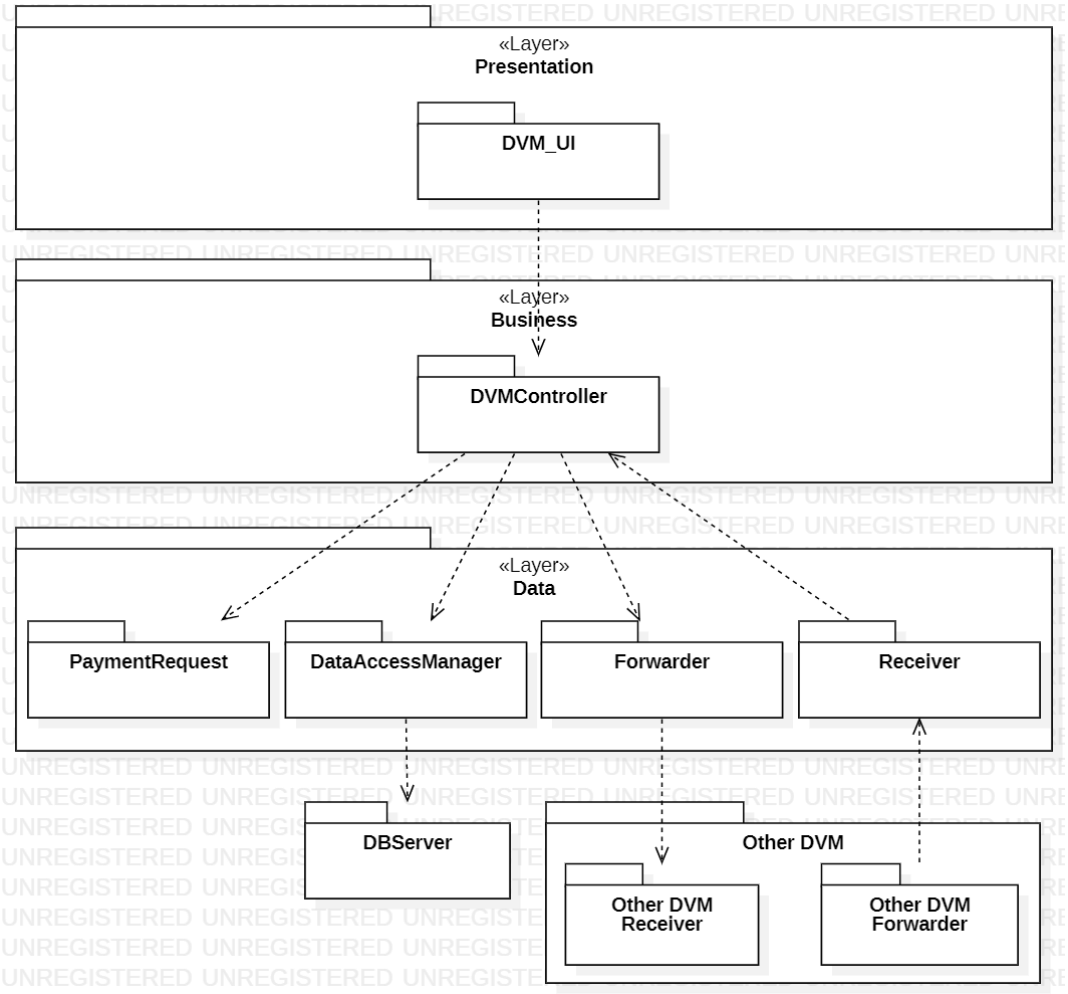

Figure 1 Module View

선정한 architectural design pattern(Rich client application architecture)에 알맞게 시스템을 3 개의 layer 로 구분하고, 각 layer 내부에 적절한 모듈이 배치되도록 선정하였다.

3.1.2 Primary presentation

Table 19 Decomposition View - primary presentation

Element		Relation
Layer	Module	
Presentation Layer	DVM_UI	DVMController 와 'allowed-to-use' 관계
Business Layer	DVMController	PaymentRequest, DataAccessManager, Forwarder 와 'allowed-to-use' 관계
Data Layer	PaymentRequest	DataLayer 와 'is-part-of' 관계 PaymentModule 과 'allowed-to-use' 관계
	DataAccessManager	DataLayer 와 'is-part-of' 관계 DBServer 와 'allowed-to-use' 관계
	Forwarder	DataLayer 와 'is-part-of' 관계
	Receiver	DataLayer 와 'is-part-of' 관계 DVMController 와 'allowed-to-use' 관계

3.1.3 Element catalog

3.1.3.1 Element

Table 20 Decomposition View - element

Element	Property
DVM_UI	사용자와 직접 interaction 이 가능한 UI 로, 사용자가 선택할 수 있는 상품을 보여주고, 결제를 완료한 상품을 제공하거나 다른 자판기에서 사용 가능한 인증코드를 제공한다.
DVMController	사용자의 입력에 따른 전체적인 동작을 위한 기능을 포함한다.
PaymentRequest	사용자가 (선)결제를 요청할 경우, 외부 결제 모듈에 결제를 요청을 한다.
DataAccessManager	각 DVM 의 위치, DVM 의 ID, 취급하는 상품 종류, 각 상품의 재고에 대한 정보와 송수신한 인증번호를 DBServer 에 저장하거나 불러온다. DBServer 에 저장하고 불러올 때 private key 를 활용하여 암/복호화를 진행한다.
Forwarder	전송할 정보를 파일의 형태로 marshalling 하여 다른 자판기의 receiver 로 전송한다. 파일을 전송하기 전, public key 를 활용하여 암호화를 진행한다.
Receiver	다른 자판기의 forwarder 로부터 파일의 형태로 수신한 정보를 unmarshalling 하여 자판기에 전달한다. 파일을 수신하기 전, public key 를 활용하여 복호화를 진행한다.

3.1.3.2 Relations

해당 없음

3.1.3.3 Interfaces

Table 21 Decomposition View - Interfaces

Element	Interface
DVM_UI	User Interface 제공 User 로부터의 input 을 method 를 통해 전달
DVMController	Method 를 통해서 control 전달
PaymentRequest	PaymentModule 에 request(JDBC)
DataAccessManager	DBServer 에 암호화하여 file I/O(JDBC)
Forwarder	다른 DVM 의 Receiver 에 암호화된 Msg 전달
Receiver	다른 DVM 의 Forwarder 로부터 암호화된 Msg 수신 Method 를 통해서 처리해야 할 Msg 전달

3.1.3.4 Behavior

Primary Functionality 와 관련이 있는 sequence diagram 은 아래의 5 개 diagram 과 같다.

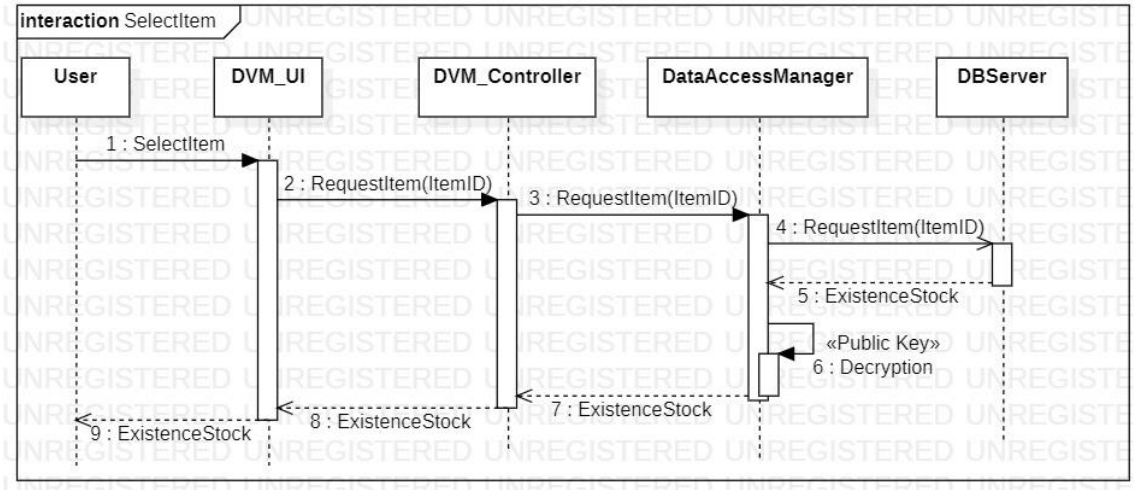

Figure 2 Module View Behavior(UC-1)

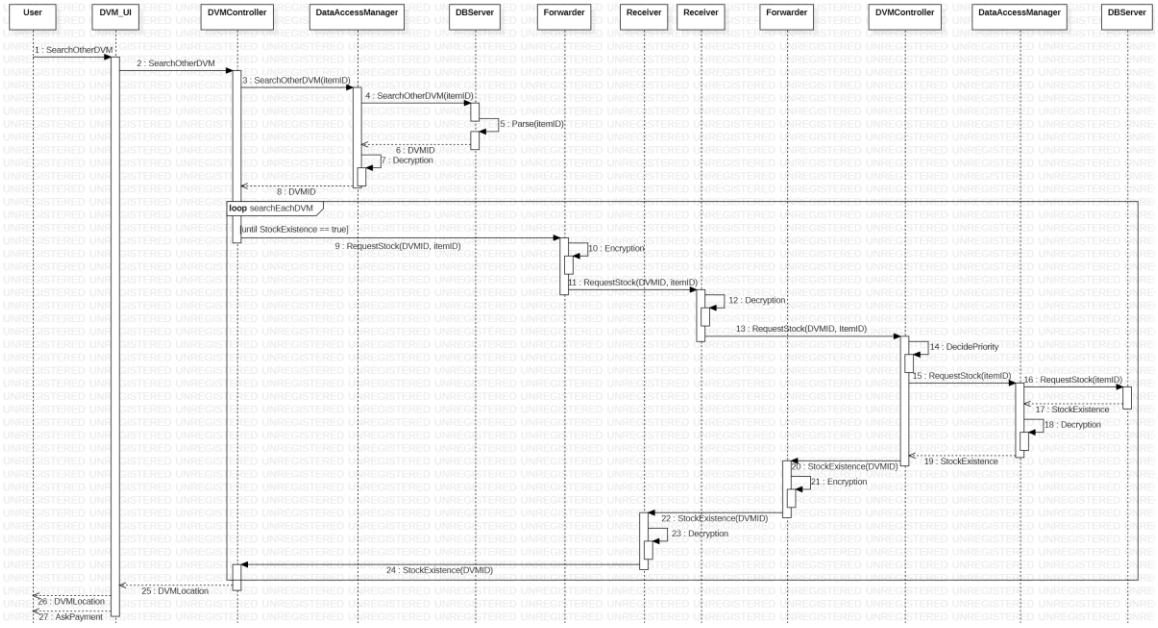

Figure 3 Module View Behavior(UC-3)

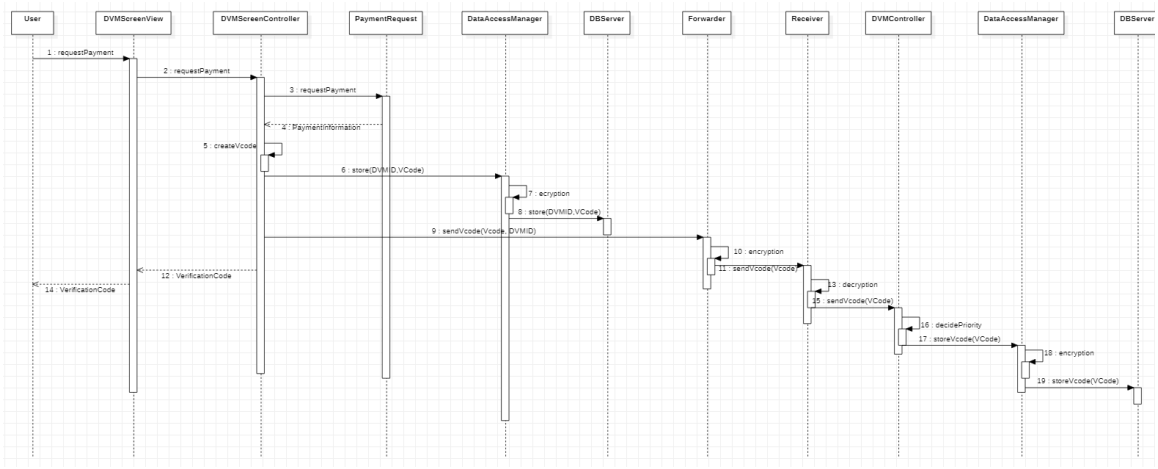

Figure 4 Module View Behavior(UC-4)

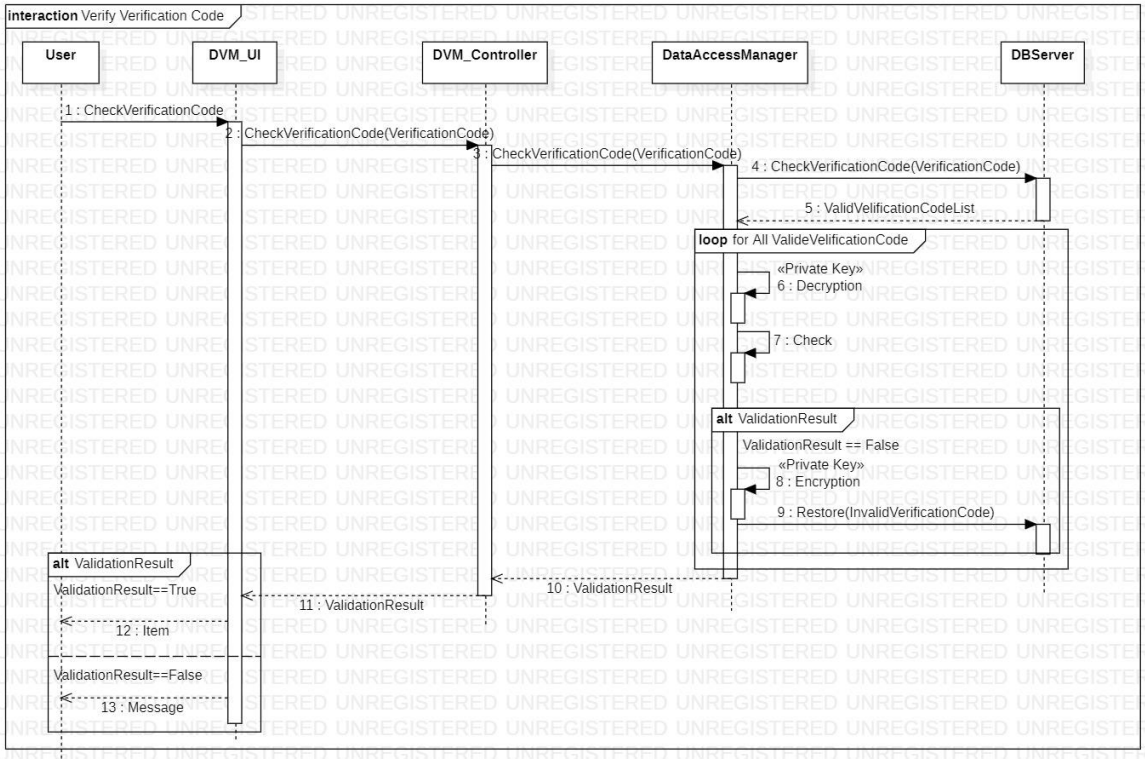

Figure 5 Module View Behavior(UC-5)

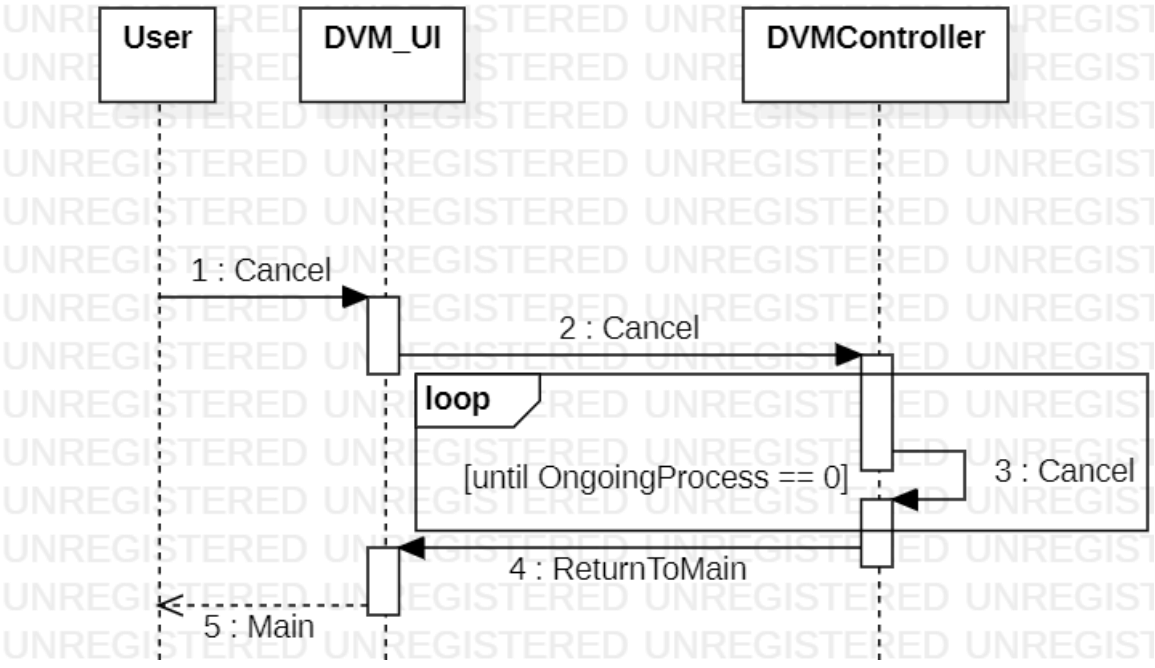

Figure 6 Module View Behavior(UC-8)

3.1.3.5 Constraints

Forwarder-Receiver 에서 데이터를 주고받을 때에는 public key 를 이용하여 주고받는 데이터를 암호화한다.

3.1.4 Context diagram

- DBServer

DataAccessManager 와 JDBC protocol 을 활용하여 통신한다. File I/O 를 담당하며, 시스템에서 사용하는 log 와 인증코드 등, 파일로 저장되어야 하는 데이터들을 담는 data source 이다. 외부에서 쉽게 정보를 파악할 수 없도록 일부 식별 코드를 제외하고는 암호화되어 있다. 각 데이터는 private key 로 암호화한다.

- PaymentModule

PaymentRequest 로부터 요청 받아 Payment 를 동작하도록 한다. 이 부분은 COTS 를 사용한다.

3.1.5 . Variability mechanisms

해당 없음

3.1.6 . Architecture background

기능별로 시스템을 파악해야 했기 때문에 decomposition 을 해야 했고, 시스템이 asynchronous 한 동작을 하기 때문에, layered 구조를 가질 필요가 있었다.

3.2 C&C Peer-to-Peer View

3.2.1 View description

3.2.2 Primary presentation

3.2.3 Element catalog

3.2.3.1 Element

3.2.3.2 Relations

3.2.3.3 Interfaces

3.2.3.4 Behavior

3.2.3.5 Constraints

3.2.4 Context diagram

3.2.5 . Variability mechanisms

3.2.6 . Architecture background

3.3 Allocation Deployment View

3.3.1 View description

3.3.2 Primary presentation

3.3.3 Element catalog

3.3.3.1 Element

3.3.3.2 Relations

3.3.3.3 Interfaces

3.3.3.4 Behavior

3.3.3.5 Constraints

3.3.4 Context diagram

3.3.5 . Variability mechanisms

3.3.6 . Architecture background

3.4 Allocation Install View

3.4.1 View description

3.4.2 Primary presentation

3.4.3 Element catalog

3.4.3.1 Element

3.4.3.2 Relations

3.4.3.3 Interfaces

3.4.3.4 Behavior

3.4.3.5 Constraints

3.4.4 Context diagram

3.4.5 . Variability mechanisms

3.4.6 . Architecture background

3.5 Allocation Work Assignment View

3.5.1 View description

3.5.2 Primary presentation

3.5.3 Element catalog

3.5.3.1 Element

3.5.3.2 Relations

3.5.3.3 Interfaces

3.5.3.4 Behavior

3.5.3.5 Constraints

3.5.4 Context diagram

3.5.5 . Variability mechanisms

3.5.6 . Architecture background

4 Relations Among Views

4.1 General Relations Among Views

Module View 를 통해 시스템의 구조를 파악한다.

C&C View 를 통해 여러 DVM 간의 통신을 정의한다.

Allocation View 를 통해 SW 와 HW 사이의 mapping 을 정의한다.

4.2 View-to-View Relations

Decomposition View 와 Layered View 를 통합하여 사용함으로써 시스템의 전체 구조를 나타낸다.

Peer-to-Peer View 를 활용하여 각 DVM 간 통신에 활용한다.

Decomposition View 와 Work Assignment View 를 통해서 분해된 각 module 을 어떻게 분배할지 정한다.

Decomposition View 를 통해서 Peer-to-Peer View View 를 구체화한다.

5 Referenced Materials

Table 22 References

IEEE 1471	ANSI/IEEE-1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 21 September 2000.
Clements 2010	Clements, Bachmann, Bass, Garlan, Ivers, Little, Merson, Nord, Stafford, *Documenting Software Architectures: Views and Beyond 2nd Edition*, Addison Wesley Professional, 2010.
Mark Richards 2009	Mark Richards, Richard Monson-Haefel, David A Chappell, Java Message Service, 2nd Edition, 2009

6 Directory

6.1 Index

6.2 Glossary

Table 23 Glossary

Term	Definition
software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
view	A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.
viewpoint	A specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view, and the techniques for its creation and analysis [IEEE 1471]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view.

6.3 Acronym List

Table 24 Acronym List

API	Application Programming Interface; Application Program Interface; Application Programmer Interface
ATAM	Architecture Tradeoff Analysis Method
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CORBA	Common object request broker architecture
COTS	Commercial-Off-The-Shelf
EPIC	Evolutionary Process for Integrating COTS-Based Systems
IEEE	Institute of Electrical and Electronics Engineers
KPA	Key Process Area
OO	Object Oriented
ORB	Object Request Broker
OS	Operating System
QAW	Quality Attribute Workshop
RUP	Rational Unified Process
SAD	Software Architecture Document
SDE	Software Development Environment
SEE	Software Engineering Environment
SEI	Software Engineering Institute Systems Engineering & Integration Software End Item
SEPG	Software Engineering Process Group
SLOC	Source Lines of Code
SW-CMM	Capability Maturity Model for Software
CMMI-SW	Capability Maturity Model Integrated - includes Software Engineering
UML	Unified Modeling Language