공학
트렌드

소프트웨어공학센터 경영지원TF팀

품질고도화를 위한 실용적인 소프트웨어 아키텍처 리뷰

Part 1 : 프랙티컬 아키텍처 리뷰의 소개와 리뷰 양식

2014. 5. 27. [제94호]

Contents

정보통신산업진흥원 부설
nipa 소프트웨어공학센터

Ⅰ. 아키텍처 리뷰의 목표와 진행 방향

IT 회사에서 많은 소프트웨어 솔루션과 서비스가 만들어진다. 설계하는 사람들의 경력은 천차만별이고, 그런 만큼 다양하게 설계된다. 게다가 역량을 가진 엔지니어들이 동일한 요구사항을 각각 다르게 이해하고 다양한 아키텍처를 설계하며 개발을 진행하는 식이다. 이런 상황에서는 무엇보다 아키텍처 모듈의 중복을 최대한 줄이고 모호한 부분은 최대한 명확하게 잡아주는 것이 중요하다. 또한 보안이 잘못되어 있거나 불분명한 설계로 발생되는 사이드 이펙트(Side Effect)를 줄여야 하는데, 아키텍처 리뷰를 통해 이 부분을 최소화 할 수 있다.

아무런 문서 없이 수 년 째 내려오는 소프트웨어들을 섣불리 유지보수하기란 쉽지 않기 때문에 많은 소프트웨어들이 방치되고 있다. 또한 이런 문화에 익숙한 사람들까지 생겨서 아무런 설계 문서/요구사항도 없이 개발하고 있는 현실이 엄연히 존재한다. 아키텍처 리뷰는 이러한 잘못된 개발 문화를 올바른 개발 문화로 만드는 '꼭 필요하고, 기본적인' 개발 환경을 구축하는 시작이기도 하다.

그림 1_아키텍처 리뷰 모임

〈그림 1〉과 같이 아키텍처 리뷰는 설계를 하는 실무자와 담당 매니저, 아키텍트(Architect)로 구성해 진행한다. 아키텍트는 네트워크, 서비스, 플랫폼을 잘 알고 있는 사람으로 해당 분야를 가이드를 할 수 있어야 하는 중요한 위치이다. 아키텍처 리뷰는 요구사항들을 구현하기 앞서 개발할 서비스 또는 플랫폼을 설계 문서로 공유하고 그에 대한 아키텍처 품질을 체크, 결과를 피드백하는 것이다. 프랙티컬(Practical)한 아키텍처 리뷰는 기본적인 아키텍처 리뷰의 형식을 따르지만, 문서나 일을 진행하는 방식을 규정하

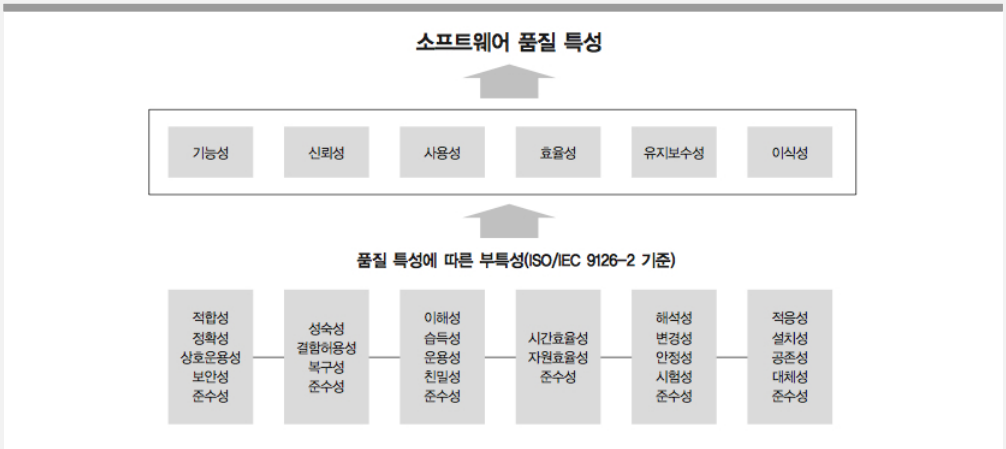
기 보다는 개발자의 자유로운 형식을 존중하면서 진행한다. 대신 정해진 형식 대신 예제나 샘플을 통한 가이드 방식 등으로 쉽게 표현할 수 있도록 방법을 제공한다. 따라서 딱딱한 틀의 잣대가 아닌 실무 환경에 대해 부드럽고 유연한 객관성이 드러날 수 있도록 한다. 결국 아키텍처 리뷰는 아키텍처 수준에서 소프트웨어를 검증하고 다양한 영역의 아키텍트들이 경험과 지식을 공유하는 것을 말한다.

아키텍처는 무엇이고 아키텍처 품질은 어떤 방법으로 체크하는가?

아키텍처는 소프트웨어 시스템에 대한 상위 수준의 구조를 의미하며 각 소프트웨어 요소들을 정의하고 그 요소들 간의 관계를 표현한다. 이런 아키텍처는 품질 지표를 따라 검토 받을 수 있다.

다음 〈그림 2〉에서 볼 수 있듯이 ISO 9126, ISO 14598, ISO 12119 등 소프트웨어 품질에 관한 국제 표준을 기준으로 아키텍처 품질 속성을 정의하고 있다. 다만 본 지표를 통해 소프트웨어 품질의 특성을 리뷰 할 수 있다고 해도, 품질 자체에만 너무 치중하면 리뷰의 속도가 저하되는 것은 물론 리뷰 문서를 작성하는 사람으로서는 막막하거나 단지 일을 위한 일이 될 수 있다. 때문에 누구나 쉽게 이해할 수 있도록 품질 특성을 강조할 필요가 있다.

그림 2_IOS/IEC 9126-1 아키텍처 품질 지표



출처: www.tta.or.kr/data/reportDown.jsp?news_num=2564

품질 속성에 대해서 하나하나 측정하는 것이 쉽지 않다. IOS/IEC 9126-1 아키텍처 품질 지표는 품질 속성을 점수로 평가하는 것이 아닌 실무적인 관점에서 보다 현실적인 이슈를 논의하고 함께 해결하기 위한 방안 또는 가이드를 제시하는 것을 그 바탕에 두고 있다. 따라서 아키텍처 리뷰는 점수만 매기는 수준에 그치는 것이 아니라 협업하는 동료로서 함께 문제의 도메인을 이해하고, 서로 도와줄 수 있는 부분을 체크한다. 또한

중복적인 개발을 줄이며 사전에 이슈를 파악할 수 있도록 하는 것이 프랙티컬 아키텍처 리뷰라고 할 수 있다. 〈그림 3〉과 같이 아키텍처 리뷰는 개발이 완료되기 전에 소프트웨어 개발 관점 뿐 아니라 운영 관점도 중요시 여긴다.

그림 3_소프트웨어를 바라보는 개발관점 / 운영관점

아키텍처 리뷰는 아키텍처를 다양한 관점에서 볼 수 있도록 하고, 잠재된 위험요소 및 이슈들을 설계 단계에서 조기 발견하여 높은 품질의 소프트웨어를 유지보수 할 수 있도록 기초를 마련한다. 또한 프로젝트 간에 빈번하게 발생하는 중복 개발 사례를 피할 수 있도록 한다. 좋은 품질로 설계한 사례는 칭찬/도입/전파하여 소프트웨어의 실질적인 품질 향상 및 효율적인 리소스 사용을 주는 좋은 사례로 만들 수 있다. 창의력이 높은 개발자의 좋은 아키텍처를 통해 다른 실무자들에게 업무에 대한 동기부여는 물론 긍정적인 도전을 받을 수 있다. 현장에서 배운 좋은 사례, 일명 베스트 프랙티스(Best Practice)들이 그 동안 혁신의 사례로 나온 경우와 비슷하다고 할 수 있을 것 이다.

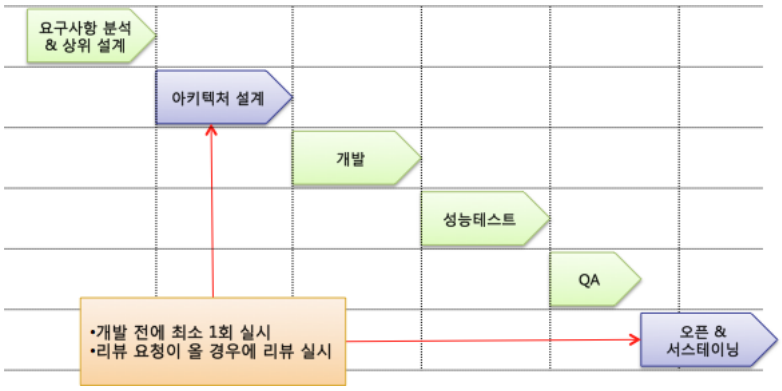
프랙티컬 아키텍처 리뷰의 핵심(DO / NOT DO)은 다음과 같다.

▶ DO
- 개발 초기에 중요한 의사 결정 방향을 정한다.
- 보안, 확장성, 가용성과 같은 품질을 속성을 고려한다.
- 의도가 정확해야 한다.

▶ NOT DO
- 리뷰 자체를 위한 문서를 작성하지 않는다.
- 잘 모르는 UML을 쓰고자 하는 노력은 하지 않는다.
- 상세화를 하지 않는다. 즉, 적당한 수준의 추상화를 추구한다.

〈그림 4〉에서처럼 회사에서 개발(또는 외주를 주는 모든 프로젝트)하는 프로젝트를 소프트웨어 아키텍처 설계 단계에서 최소 한 번 이상 리뷰를 수행하도록 한다. 이를 통해 검증하고 이후 기술 관련 자문 및 지원을 받을 때에도 계속 요청하면 아키텍처 리뷰를 실시할 수 있도록 한다. 결함이 최소화 될 수 있는 수준의 아키텍처를 설계하도록 지원한다.

그림 4_아키텍처 리뷰 시행 시기



아키텍처 리뷰의 목표는 〈그림 5〉와 같이 좋은 성능과 좋은 품질, 글로벌 소프트웨어가 될 수 있는 아키텍처를 지원하는 것이다.

그림 5_아키텍처 리뷰의 목표

또한 리뷰 문서는 최소한으로 하여 업무의 부담을 주지 않을 정도로 결정되어야 한다. 잘 모르는 UML이나 단계별 상세화는 하지 않는다. 따라서 아키텍처는 개발 2~3년차가 이해하는 수준으로 나타낼 수 있어야 하며, 어떠한 툴(Tool)을 써도 상관없도록 가이드 한다.

II. 아키텍처 리뷰 내용

저자의 실무 환경은 Web, WAS / SmartPhone 이 가장 많이 개발되는 환경이라 아키텍처 리뷰의 관점이 이 부분으로 쏠려 있고, 회사 내 아키텍처 자산을 공개할 수 없는 관계로 외부 링크와 개념적으로만 설명함을 양해 바란다.

원활한 이해를 위해 시스템 아키텍처, 어플리케이션 스택 아키텍처, 시스템 구성 및

네트워크 아키텍처, 서비스 흐름도와 같은 소프트웨어 아키텍처와 ERD(entity-relationship diagram) 와 같은 DB 스키마 리뷰로 나누어 살펴보고자 한다.

소프트웨어 아키텍처와 관련된 4개의 아키텍처 다이어그램은 UML을 써도 되고 특정 툴에 얽매이지 않도록 자유롭게 표현하도록 했다. 또한 나누어진 아키텍처 다이어그램으로 보이지 않고, 자유롭게 시스템 아키텍처와 어플리케이션 스택 아키텍처를 동시에 보여줄 수 있도록 했다. ERD는 Mysql Workbench나 ERWin과 같은 DB Modeling 툴을 사용해서 쓸 수 있도록 가이드 했다.

2.1 시스템 아키텍처

시스템 아키텍처는 외부시스템과의 연동구조를 포커스로 다룬다.

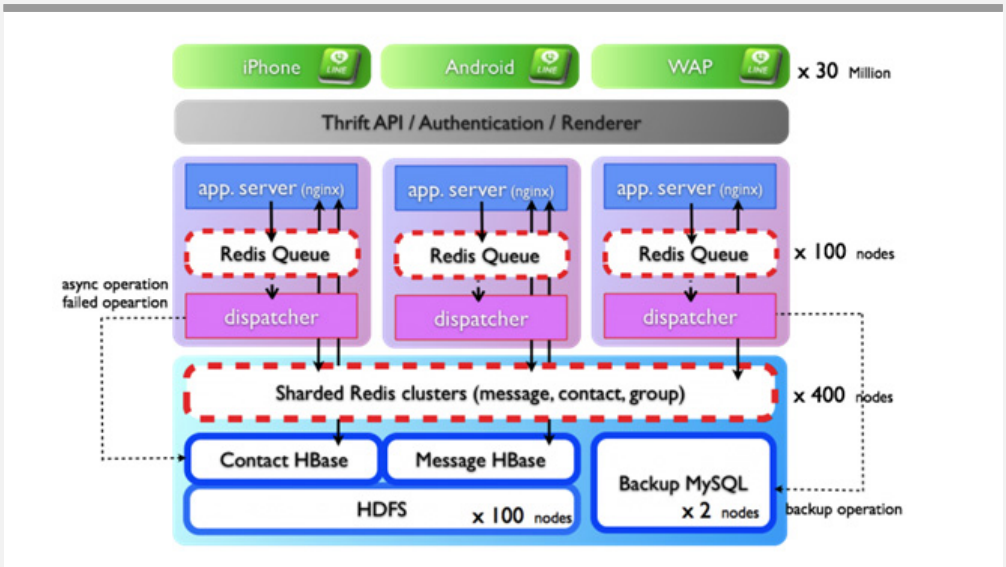
UML의 클래스 다이어그램으로 표현할 필요 없이 아키텍처 전체를 다루고 실행단위와 데이터의 흐름을 개념적으로 보여주도록 한다. 애플리케이션 또는 전체 시스템의 상위레벨 컴포넌트 및 그들 간의 관계를 식별하도록 한다. 세부 내용이 아닌 시스템을 적절히 추상화하여 분할한다. 개발/스테이징/운영 서버의 아키텍처가 모두 같으면 하나로 표현을 할 수 있으나, 개발 서버와 운영 서버의 구조가 다르다면 나누어서 설명해야 한다.

예를 들어 표현할 때 아래 표의 요소 관점으로 설명하면 좋을 것이다.

분류	내용
Communication	Sync/Async(Queue,Comet), Pub/Sub, cookie/session/OAUTH, json/xml/raw/rest, keepalive 여부, http/https/암호화방식
Data Storage	DBMS(Oracle, Mysql), Nosql (MongoDB,Hbase,Redis)
Node	Client/Server, 내부/외부시스템
Network	Firewall in/out

〈그림 6〉의 예시처럼 스토리지(storage)와 서버(server) 관점의 전체적인 아키텍처를 대용량 관점에서 쉽게 이해할 수 있게 한다.

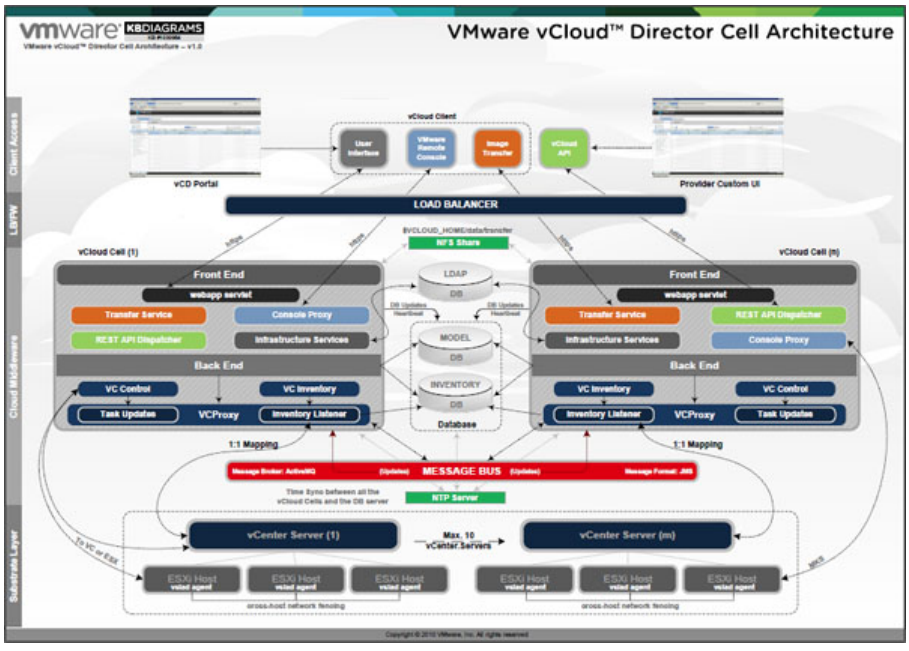
그림 6_Line Storage



출처: http://tech.naver.jp/blog/?p=1420

〈그림 7〉은 VMWare의 vCloud 아키텍처를 쉽게 이해할 수 있다. Front Server, Backend Server, Message Bus, Web Server, Storage 간의 관계를 볼 수 있다.

그림 7_VMWare vCloud Architecture



출처: http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1030954

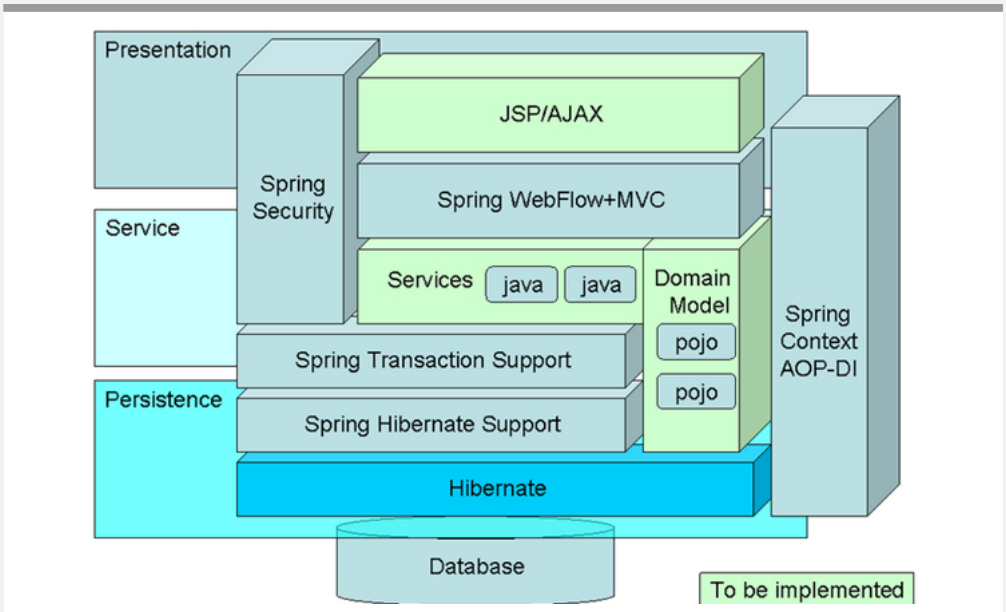
2.2 어플리케이션 스택(Stack) 아키텍처

중요 컴포넌트에 대한 어플리케이션 스택의 구조를 설명한다. 구현할 시스템의 주요 컴포넌트의 스택 아키텍처와 연동하는 중요 노드(Node) 간의 관계를 식별한다. 컴포넌트와 노드 간의 관계는 간단히 표현한다. 주요 컴포넌트는 스택 형태로 세부 내용이 들어가야 하며, 통신하는 외부 시스템의 노드는 간단히 표현한다.

분류	내용
Solution	Open Source/Commercial Product
Open Source	PHP/Java, Apache/Nginx, Tomcat/Jetty/Netty, MVC Framework, json/xml parser, connection pool, message platform, cache lib

〈그림 8〉의 예는 WAS를 어떻게 구성했는지에 대한 그림을 보여준다. Spring과 Hiberat를 이용하여 구성했음을 확인할 수 있다.

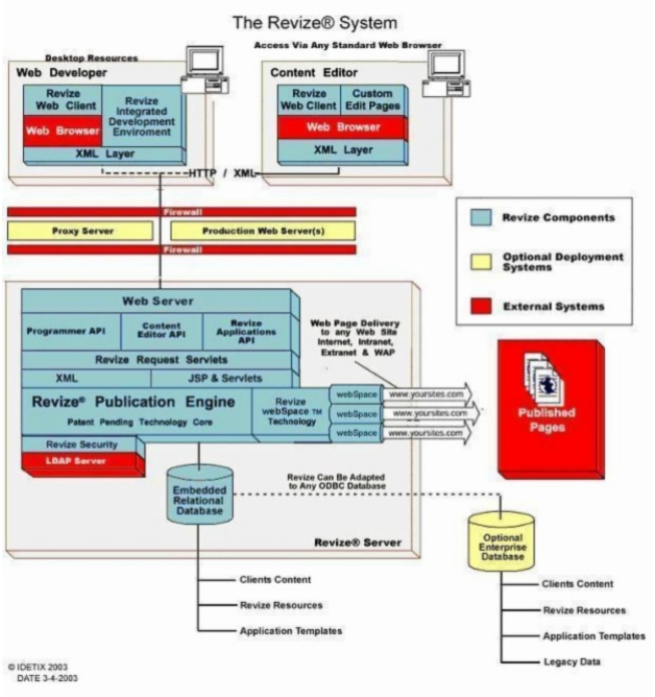
그림 8_Hibernate와 Spring을 이용한 Stack Diagram



출처: http://mashupfactory.wordpress.com/2008/02/12/application-architecture

〈그림 9〉의 예처럼 만약 오픈 소스가 아닌 3rd Party Framework인 Revize Framework을 쓰는 경우인데, 시스템 아키텍처와 어플리케이션 스택 아키텍처를 동시에 그림으로 보여줄 수 있는 예제이다.

그림 9_Revize Server Application 아키텍처



출처: http://www.revize.com/technical_architecture.html

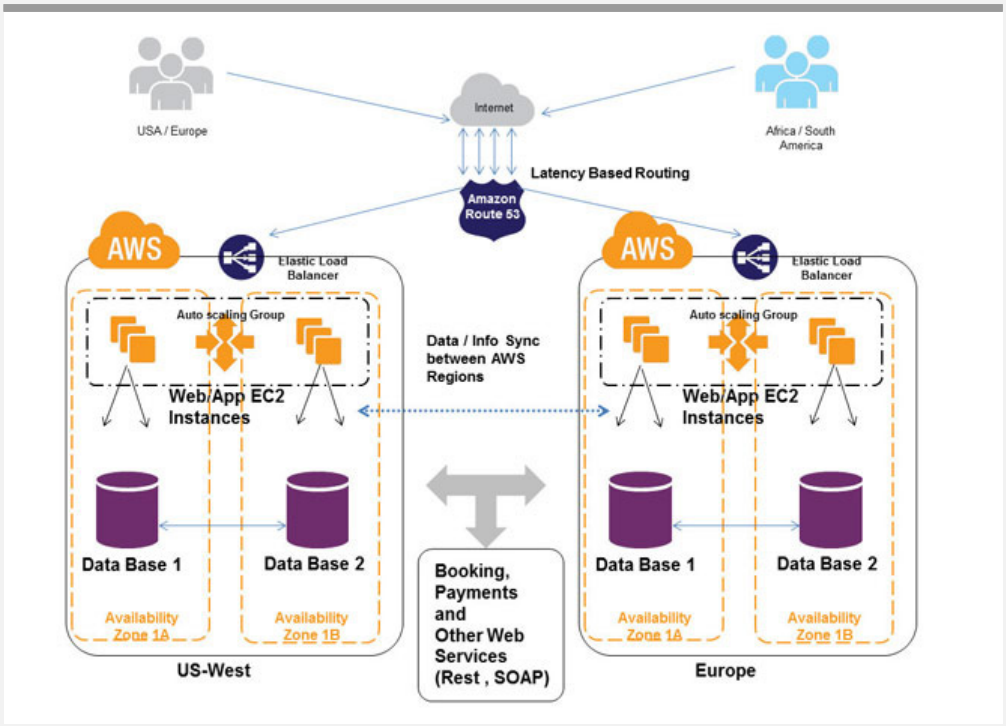
2.3 시스템 구성 및 네트워크 아키텍처

단말부터 서버, 스토리지 단까지의 시스템/네트워크 구조 관점으로 설명한 다이어그램이다. 시스템 구성 및 네트워크 장비 관점으로 본 구성 요소를 보여준다. 네트워크 레이어(Layer) 단위로 네트워크 장비의 가용성 및 서버 가용성 및 스토리지 가용성을 확인할 수 있다. 필요하다면 가용성에 대한 상세 대처 방안에 대한 시나리오를 추가할 수 있다.

분류	내용
Server	Server Spec
Network	L4(L7) / Zookeeper
Zone	Firewall in/out, IDC location, Firewall Zone
Stroage	Oracle RAC/Mysql MMM, NFS/SAN, RAID, DRBD

〈그림 10〉은 AWS를 이용한 아키텍처 구성안이다. Route 53과 ELB를 통해서 서버 요청을 받고 US West와 Europe에 서버와 DB를 두고 Replication 하는 방식을 설명하고 있다.

그림 10_AWS 를 이용한 Onlie Cruise Company 시스템 구성도



출처: http://yourstory.com/2012/08/geo-distributed-architecture-using-route53-lbr-part-2

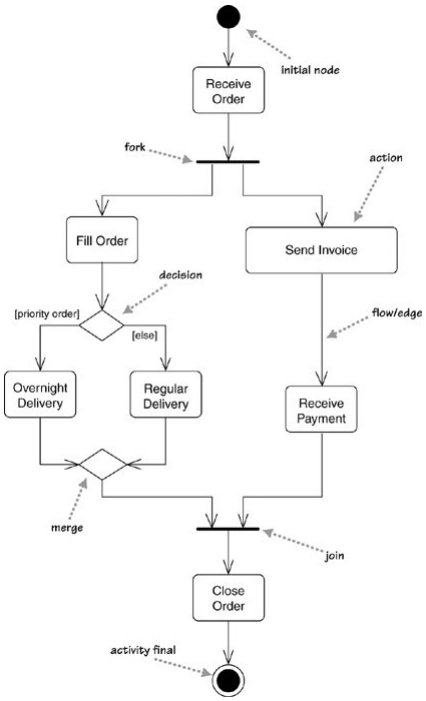
2.4 서비스 흐름도

주요한 유즈케이스(UseCase)의 Conceptual한 흐름도를 설명하는 것이다. UML의 Sequence 혹은 Activity 다이어그램과 비슷한 개념을 가지고 있다.

시나리오를 이해할 수 있는 수준의 서비스 흐름도를 보여준다. 구성요소, 하위시스템, 행위자 인스턴스 간의 메시지 시퀀스를 Conceptual하게 보여준다. 클래스 단위가 아닌 흐름과 구현 요소에 집중한다. 모든 내용을 보여주기보다는 주요한 내용으로 보여줄 수 있도록 한다.

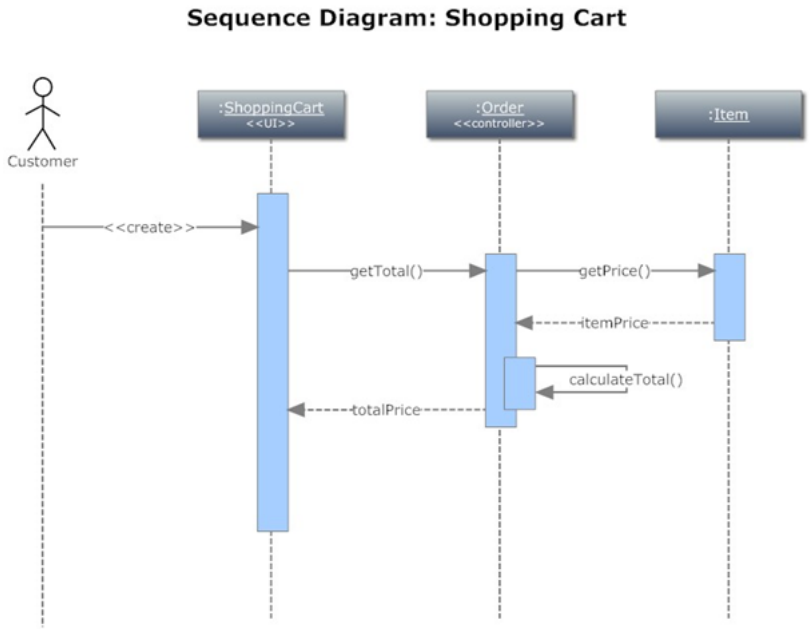
〈그림 11〉과 같이 액티비티(Activity) 다이어그램으로 보여줄 수 있고, 〈그림 12〉와 같이 시퀀스(Sequence) 다이어그램으로 보여줄 수 있도록 한다. 어느 타입으로 어느 툴로 그려도 이해할 수 있는 형태로 되게 한다.

그림 11_액티비티(Activity) 다이어그램 예제



출처: http://pagesperso.lina.univ-nantes.fr/~molli-p/pmwiki/pmwiki.php/Main/Umltd2

그림 12_시퀀스(Sequence) 다이어그램 예제



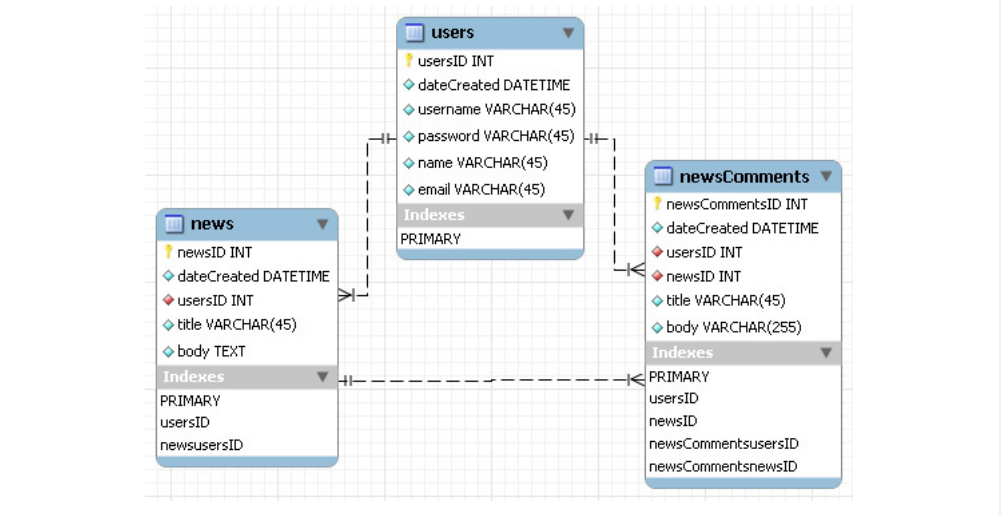
출처: https://www.smartdraw.com/examples/view/shopping+cart+sequence+diagram

2.5 ERD (Entity-Relation Diagram)

ERWin 또는 Mysql Workbench와 같은 DB 모델링(Modeling) 툴을 이용하여 만든 개체-관계 모델이고, 구조화된 데이터에 대한 표현 다이어그램이다. 완벽하게 세세한 ERD가 아닌 Key Attribute가 포함된 Conceptual한 ERD의 수준을 의미한다. 테이블 내용만으로 이해를 할 수 없을 것 같다면 내용을 서술하는 것도 좋다.

〈그림 13〉과 같이 Mysql Workbench 으로 만든 핵심 포인트만 적용한 ERD를 만든다.

그림 13_Mysql Workbench로 만든 Entity Relation 다이어그램



출처: http://superiorwebsys.com/blog/58/What_is_ERD_and_Why_Is_It_Important_In_Website_Development

Ⅲ. 정리

지금까지 아키텍처 리뷰의 의미와 아키텍처 리뷰 시 소개할 다이어그램 5개를 소개했다. 아키텍처 리뷰를 통해 개발자와 아키텍트들이 함께 개발 전에 성능/보안/확장 정보를 확인하여 이슈를 체크해볼 수 있다. 또한 중복되는 컴포넌트 사용도 사전에 방지할 수 있으며, 사전 협의를 통해 비논리적인 부분을 파악할 수 있다.

part 2에서는 아키텍처 리뷰를 통해 어떻게 품질을 평가할 수 있는지 살펴보고 체크리스트, 사례 등을 살펴볼 예정이다.