정보통신산업진흥원 부설
nipa SW공학센터

SW아키텍처 참조모델

[의료정보교환 시스템의 SW아키텍처 참조모델]

[Version 1.0 20121230]

SW공학센터

SW공학기술팀

SW아키텍처 실무자 포럼 의료정보 분과

SEC-2012-RM001

본 문서는 국내 기업의 소프트웨어 품질 및 생산성 향상을 지원하기 위하여 작성되었습니다.

본 문서는

지식경제부 산하

정보통신산업진흥원 부설

SW공학센터

SW아키텍처 실무자 포럼

에서 작성되었습니다.

SW공학센터는 SW제품 생산능력 향상, SW공학기술 산업현장 적용 등을 위해 대학 및 전문 연구기관과 기업 현장을 연결하는 중심 허브가 되어 SW개발 중소기업들에게 전문 컨설팅을 제공하고 있습니다. 이 같은 역할을 충실히 수행하기 위해 산업과 기업의 SW공학기술 관련 요구사항에 전문적이고 신속한 대응을 할 수 있는 핵심 기능 연구와 SW개발 프로젝트의 성공 여부와 문제점을 미리 예측하여 SW품질과 생산성, 제품결함 등을 총체적으로 진단 할 수 있는 SW공학 컨설팅 등도 추진하고 있습니다. 이와 더불어 SW품질과 생산성, 비용 등을 체계적으로 추적 평가할 수 있는 데이터 수집체계도 강화하여 SW기업들이 이를 손쉽게 활용하게 함으로써 전체적인 국가 SW품질을 향상시키는 업무도 수행중입니다.

사전에 SW공학센터의 허가를 받거나 논의하지 않은 모든 형태의 책임에 대하여 SW공학센터에서는 보증하지 않습니다.

본 지침에 대한 더 많은 정보와 SW공학에 대한 추가 정보를 얻고 싶다면, SW공학센터 홈페이지(www.software.kr)를 방문하여 주십시오.

담당자 강승준 책임 [ksj@nipa.kr, 02-2132-1344]

의료분과

리퍼런스 아키텍처에 대한 설명

Version 1.0

개정 이력

버 전	변경일	변경 사유	변경 내용	작성자	승인
1.0	2012-12-20		초기 항목 작성	신현묵	
1.1	2012-12-27		세부 항목 수정	신현묵	

1. 개 요

이 문서는 2012년 추진되었던 NIPA 실무아키텍트 포럼의 의료분과의 리퍼런스 아키텍처에 대한 결과물에 대한 설명을 하기 위한 문서이다. 이 실무 아키텍트 포럼을 추진하고, 세부적으로 진행한 내용에 대해서 정의된 내용을 설명 하고 세부적인 산출물들에 대해서 설명하기 위하여 이 문서를 만들었다. 이 문서의 세부적인 내용을 참조하여 의료분과의 리퍼런스 아키텍처가 어떤 취지로 만들어졌으며, 어떤 이유로 만들어 졌는지에 대해서 설명이 되었으면 한다.

또한, 실제 회의상에서 진행되었으며 논의되었던 내용들을 언급하는 것을 목적으로 하고 있다.

2. 실무아키텍트 포럼의 참가자

성명	소속	비고
신현묵	(주)헬스허브	분과장
김태현	(주)인피니트 헬스케어	
정국상	LG U+	
양수열	(주)헤임달	
박재범	(주)휴레이	
안동영	소유커뮤니케이션	
이한아	㈜헤임달	
안소현	KTH	

이상의 8명이 **실무아키텍트포럼의 의료분과 연구모임을 추진하였다**. 이상의 연구자들은 의료정보에 익숙한 전문가들과, 소프트웨어 아키텍트, 의료정보를 다루는 소프트웨어 개발자들로 이루어져 있

다.

3. 실무아키텍트 포럼의 연구 진행사항

모임	모임내용	날짜	비고
1차	아키텍처 실무포럼의 추진방향에 대해서 토의	2012/6/27	
2차	NIPA착수보고 이후의 세부적인 방향성 논의	2012/7/30	
3차	의료정보시스템의 전체적인 개괄사항에 대해서 토의	2012/8/9	
4차	의료정보의 구조적인 형태에 대한 새로운 접근법에 대해서 토의	2012/8/30	
5차	의료정보의 영역중 의료영상정보에 대한 토의	2012/9/19	
6차	의료정보학회에서 진행되는 방향성에 대한 토의	2012/9/26	의료정보학회
7차	최종 산출물에 대한 방향성 수립 및 검토	2012/10/10	
8차	대용량 의료영상정보 리퍼런스 아키텍처 기본개념 정의	2012/10/24	
9차	대용량 의료영상정보 리퍼런스 아키텍처 정의	2012/11/6	
10차	대용량 의료영상정보 리퍼런스 아키텍처 정의	2012/11/13	
11차	대용량 의료영상정보 리퍼런스 아키텍처 정의	2012/11/20	
12차	대용량 의료영상정보 리퍼런스 아키텍처 정의	2012/11/27	
13차	대용량 의료영상정보 리퍼런스 아키텍처 정의관련 발표자료논의	2012/12/7	

현재 연구모임은 총 13차례 포럼의 형태로 진행이 되었으며, 2012년 12월 14일에 NIPA주최하에 '2012 SW 아키텍트 컨퍼런스'에 참가하였다. 장소는 : 전문건설회관 국제회의실(3층)에서 진행하였으며, 발표내용은 다음과 같다.

'**의료영상정보 참조 아키텍처의 취지와 국내/해외 의료정보환경**'이라는 주제와 '**의료영상정보 교환 참조 아키텍처 소개**'의 주제로 신현욱 이사와 김태현 책임이 발표하였다.

4. 의료분과의 목표 및 취지

의료분과에서는 대한민국의 의료정보환경의 현실적인 측면을 고려하여, 향후 의료정보시스템을 디자인할경우에 효과적이고 도움이 될 수 있는 실무적인 내용들을 중심으로하여 차세대 의료정보시스템을 디자인할때에 의료정보시스템의 소프트웨어의 품질 향상을 위하여, 소프트웨어 아키텍처 활동을 수행하기 위한 참조자료를 구성하기 위하여 구성된 연구분과이다. 해당 연구분과에서는 의료정보와 IT투자의 상관관계에 대해서 알아보는 것으로 해당 연구모임이 시작되었다.

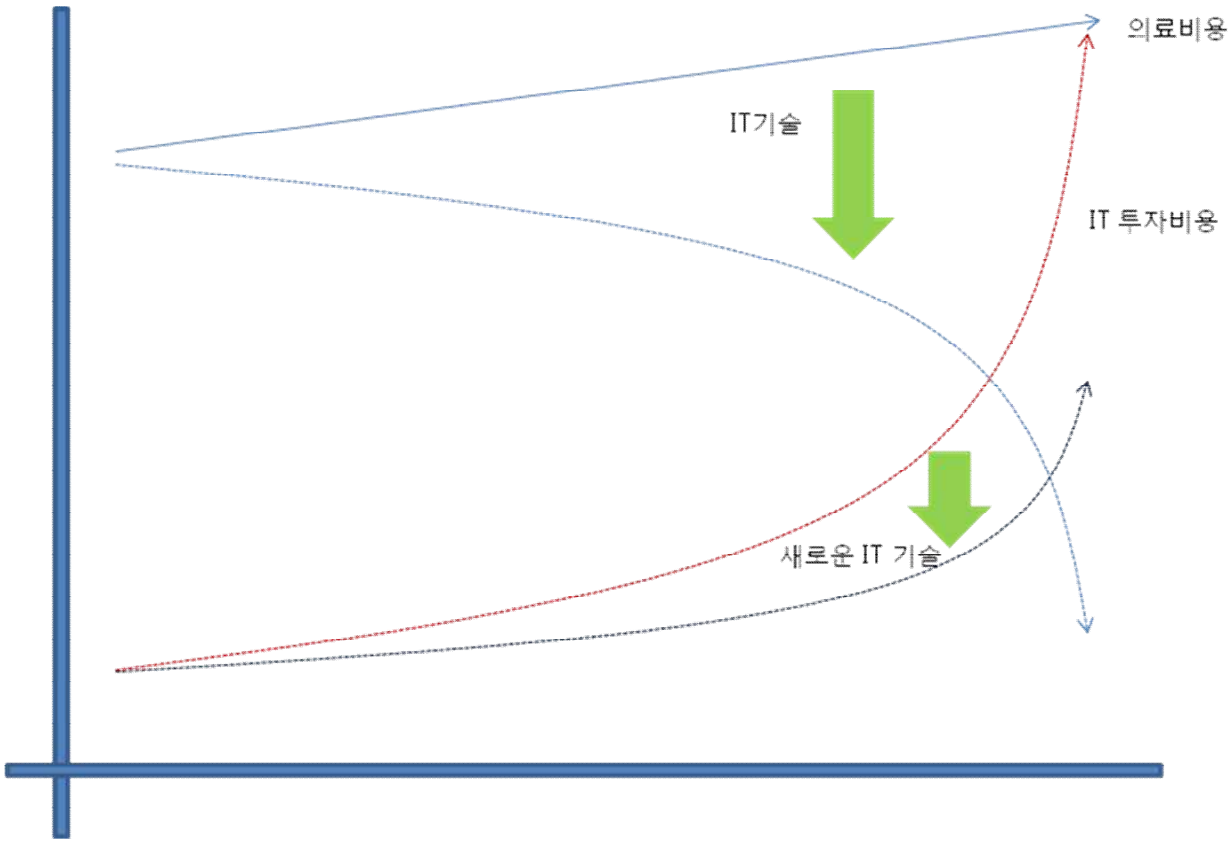


그림 1 의료비용과 IT투자의 상관관계

그림1에서 보여주듯이 의료비용이 상승되는 것을 줄이기 위해서 IT기술이 도입되었지만, 실질적으로 IT투자비용이 그 기대의 효과를 이루지 못하고, 오히려 비용을 증가시키는 그래프를 만드는 것

이 현재의 의료정보시스템이 가지고 있는 아이러니라고 볼 수 있다.

이러한 문제들은 어디에서 오는가에 대해서는 끊임없는 화두를 두고 연구해야할 내용이지만, 현재의 상황을 정리한다면 다음과 같다.

그것은 현재의 **대한민국의 의료정보시스템들은 대부분 개별적인 SI형태의 개발을** 하고 있으며, **대형 병원의 경우에는 년 수백억원의 연간 유지보수료를 지불하는 구조로** 진행하고 있는 가장 큰 이유는 무엇일까?

또한, 대한민국의 의료보험체제에 있어서 자주 변화하고 있는 의료보험체계와 수가. 그리고, 제약사항들에 대해서, 왜? 비슷한 모듈을 비슷한 사람들이 동일한 작업을 하고 있는 것일까?

이러한 대한민국의 의료정보시스템의 주변적인 환경을 고려하여, 그것에 대해서 보다 효과적으로 대응할 수 있는 방법은 없을까 하는 점이, 의료분과의 중요한 동기와 목표가 되었다.

5. reference architecture의 필요성

현재의 의료서비스와 IT의 관계는 다음과 같이 간단하게 정의할 수 있다.

- 의료기관간의 정보교환과 의료비 상관관계

- 의료기관간의 정보교환과 의료의 질 상관관계

이 두가지 모두, 의료기관간에 의료정보를 교환함으로써 의료비는 줄어들고, 의료의 질은 상승한다는 것이 대부분의 국가에서 얻어지고 있는 수치이며 결과들이다. 물론, 대한민국의 의료환경은 이러한 것을 초월한 형태로 의료서비스를 극소의 비용으로 처리하는 해외와는 아주 차별적인 상황으로 이루어져있다. 그러함에 따라서, 의료기관간의 의료정보를 교환하고자하는 이슈가 그렇게 심각하게 보이지 않는다.

그것은, 대한민국의 의료서비스체제는 단지 의료정보를 교환함으로써, 의료비를 절감하는데 있어서, 의료기관에서 받아들이기 어려운 구조를 가지고 있고, 실제 해당 서비스를 제공하는 것도 건강보험의 주체가 해당 서비스를 만들어서 사용할 수 있는 구조를 만들지 않는다면, 실제 사용하기도 매우 어려운 상황이라고 할 수 있다. 이것과 관련한 상세한 내용을 정의하기 위하여, 다음과 같은 문서를 별도로 정의하도록 한다.

HEALTH-02-디지털병원에 대한 정의

HEALTH-03-의료정보시스템 기능목록

HEALTH-04-의료정보시스템 표준기능목록

이 정의와 기능목록은 대한민국의 의료정보시스템의 표준기능들과 그 형태들을 정의함으로써, 우리의 의료정보체계에 대해서 학계가 아닌 정보산업계에서 바라본 표준정의를 위한 하나의 시작을 하기 위함이다.

이러한 조사와 연구를 기반으로, 다음과 같은 목표를 가지고 리퍼런스 아키텍처를 정의하도록 한다.

- **통합적인 진료서비스 체계 디자인**
- **축적되어지고 방대한 의료정보 환경 (생체정보 분석)으로의 확대**
- **교환되어지는 정보를 구현할 통합적인 방법**
- **가능하다면, 지속적으로 활용가능한 아키텍처의 필요성**

미래의 의료정보시스템 환경은 위와 같은 큰 목표에 어울리도록, 의료정보시스템이 진화될것으로 예측되어진다. 그러한, 예측되어지는 환경을 위해서 고려해야할 사항들은 다음과 같다.

- **개념모델이 실제 구현 가능해야 한다.**
- **시스템 디자인에 필요한 요소를 충분하게 도출해야 한다.**
- **다양한 서비스를 구현할 수 있어야 한다.**
- **의료서비스 디자인 측면에서 병원의 시스템 디자인시의 요구사항을 만족하여야 한다.**
- **의료서비스의 IT기술적인 측면에서 향후 확장 가능한 시스템의 형태를 가져야 한다.**
- **의료서비스의 비용적인 측면 - 향후 시스템의 추가 비용과 활용에 필요한 사항을 고려하야 한다.**

이 처럼, 고려하여야 할 내용들을 포함한 리퍼런스 아키텍처에 대한 필요성을 바탕으로, 의료분과는 세부적인 토의와 연구를 진행하였다.

6. 리퍼런스 아키텍처의 방향성

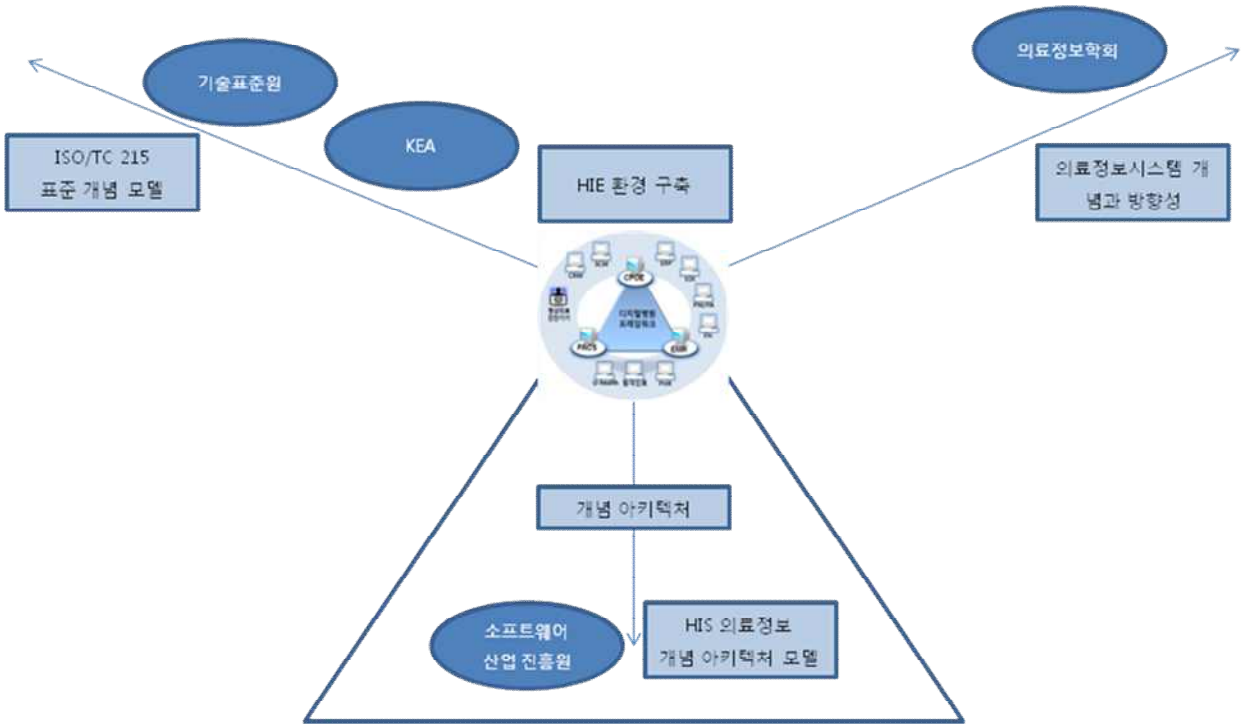


그림 2 연구의 방향성

의료분과의 연구 방향성은 기술표준원, KEA와 같은 단체에서 추진하는 표준개념 모델과는 다르다. 그것은 국내 학계의 의료표준과 의료기기 중심의 연구 방향성과는 다르며, 의료정보학회에서 추진하고 있는 의료정보시스템의 개념과 방향성과도 다르다.

실무 아키텍트 포험의 의료분과에서는 실제 의료정보시스템을 구현하고, 대한민국의 상황을 고려한 의료정보시스템의 개념 아키텍처 모델을 만드는 것을 목적으로 한다. 이는, 표준을 지향하되, 실제 상황을 고려한 리퍼런스 아키텍처를 수립하겠다는 목표와 일치한다.

이는 실제 의료정보환경에서 소프트웨어 개발시에 참조하거나 소프트웨어 아키텍처링을 할 수 있는 과정에 대해서 의료도메임의 지식이 담겨진 상태의 표준화된 산출물과 개발의 프로세스에 대해서 보여주는 것이 리퍼런스 아키텍처를 만드는 기본적인 연구의 방향성이다.

의료정보소프트웨어를 개발하는 수많은 개발자들이 참조할 수 있는 형태로 리퍼런스 아키텍처를 연구하고 이를 발표하여 실제 활용될 수 있도록 하는 것이 목적이다.

7. 실무아키텍트 포럼에서 다루는 영역

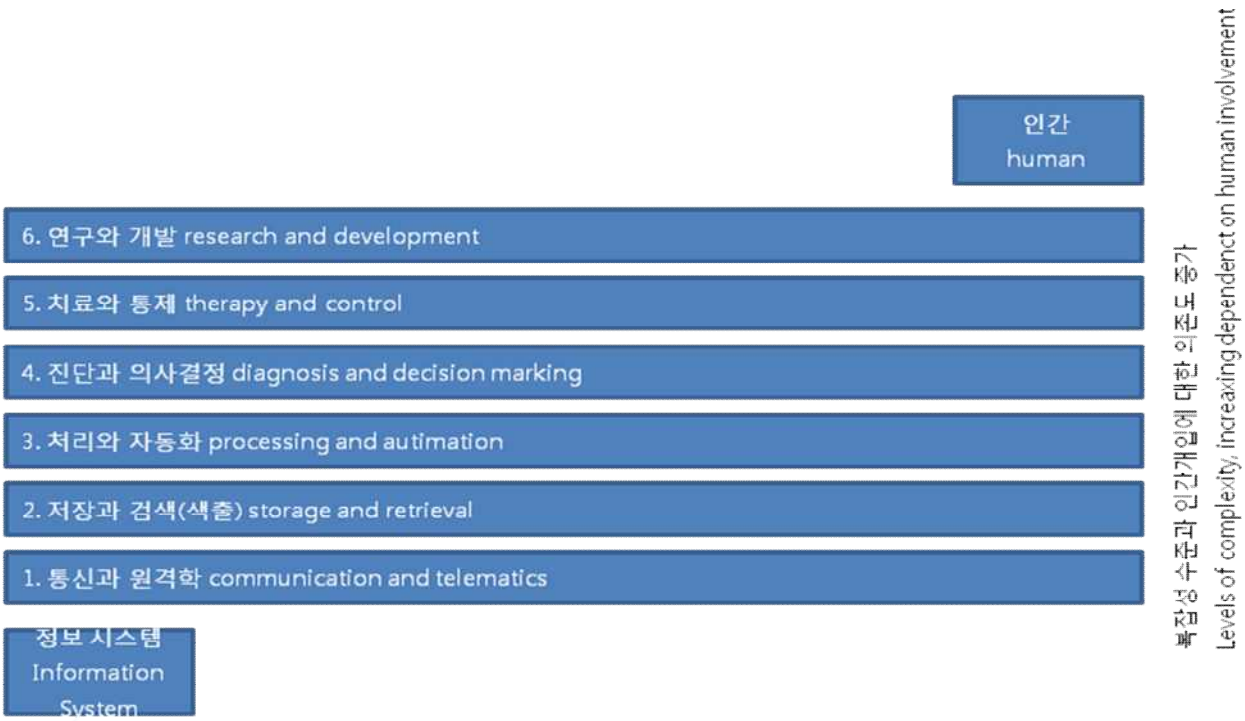


그림 3 보건의료의 정보시스템 응용수준

실무 아키텍트 포럼의 의료분과에서는 실제 의료정보시스템을 구현하고, 대한민국의 상황을 고려한 의료정보시스템의 개념 아키텍처 모델을 만드는 것을 목적으로 하지만, 이러한 의료정보시스템의 응용수준의 전체를 고려하고 있다.

이러한 정보시스템의 전체적인 정보교류를 효과적으로 수행할 수 있는 리퍼런스 아키텍처를 수립하는 것을 고려한다.

전체적인 소프트웨어에 필요한 개념과 환경에 대해서 모두 고려하고 있으며, 실제 의료환경과 의료서비스의 형태도 같이 고려하여 진행하도록 한다.

8. 의료정보서비스의 전체 아키텍처

의료분과에서 예측하고 있는 미래의 의료정보서비스의 형태를 전반적으로 표현하자면, 다음의 그림 4의 전체적인 모델로 표현할 수 있다. 향후, 미래에는 이러한 의료서비스들이 구축되고 구현될 것이다.

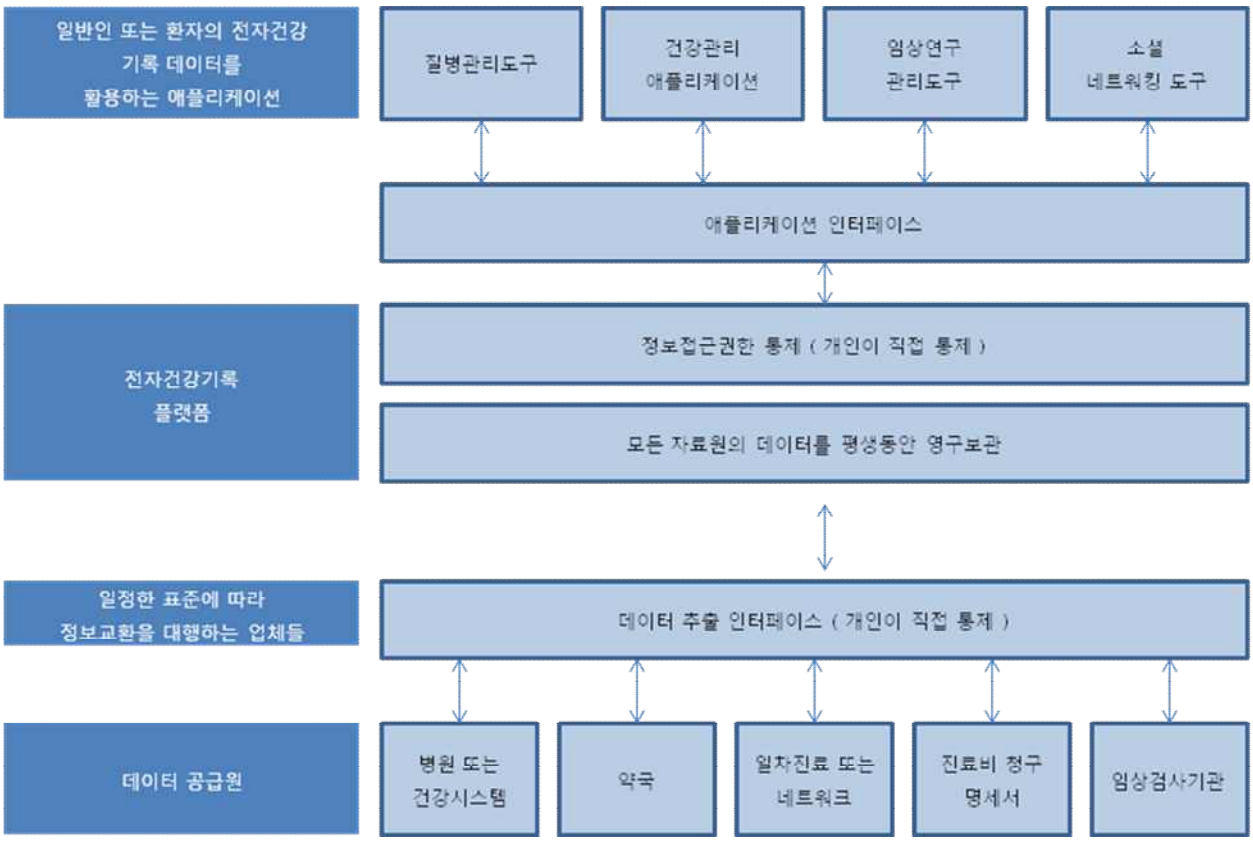


그림 4 의료서비스의 전체 아키텍처

환자를 중심으로한 질병관리도구와 각종 어플리케이션들과 인터페이스를 할 수 있는 영역과 실제가 존재하고, 이러한 전자건강기록에 대해서 개인이 직접 통제할 수 있는 통제기능과, 모든 자료를 영구보관하거나 평생동안 보관할 수 있는 영역들이 존재하고, 이러한 전자건강기록 플랫폼에 일정한 표준에 따라서 정보교환을 대행하는 업체들이 존재할 것이다.

그리고, 현재의 의료정보를 제공할 수 있는 병원과 약국, 의료기관들의 시스템과 연동하여 데이터를 교환할 것이다. 이러한 정보시스템의 전체적인 정보교류를 효과적으로 수행할 수 있는 리퍼런스 아키텍처를 수립하는 것을 고려한다.

9. 의료서비스 체제의 변환

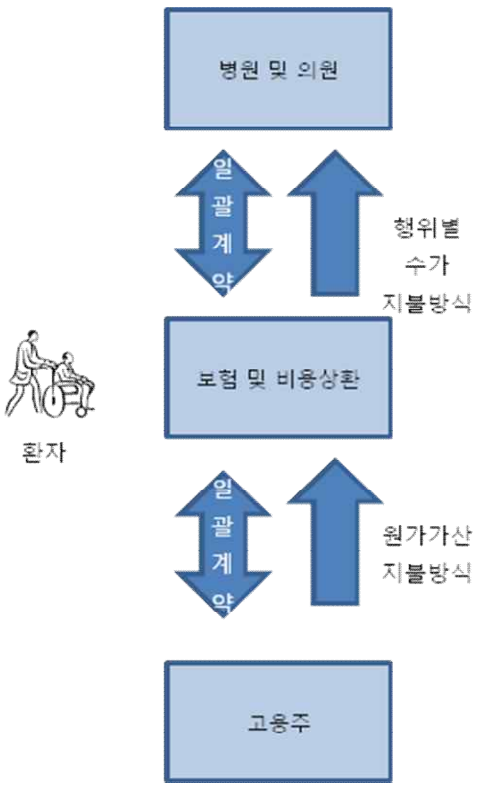


그림 5 현재의 의료서비스 체제

현재의 의료서비스 체제는 보험회사와 환자, 고용주간의 일괄계약에 의한 방식으로 데이터 교환이 복잡하지 않으며, 정해진 방식으로 의료서비스가 구현되어 있다.

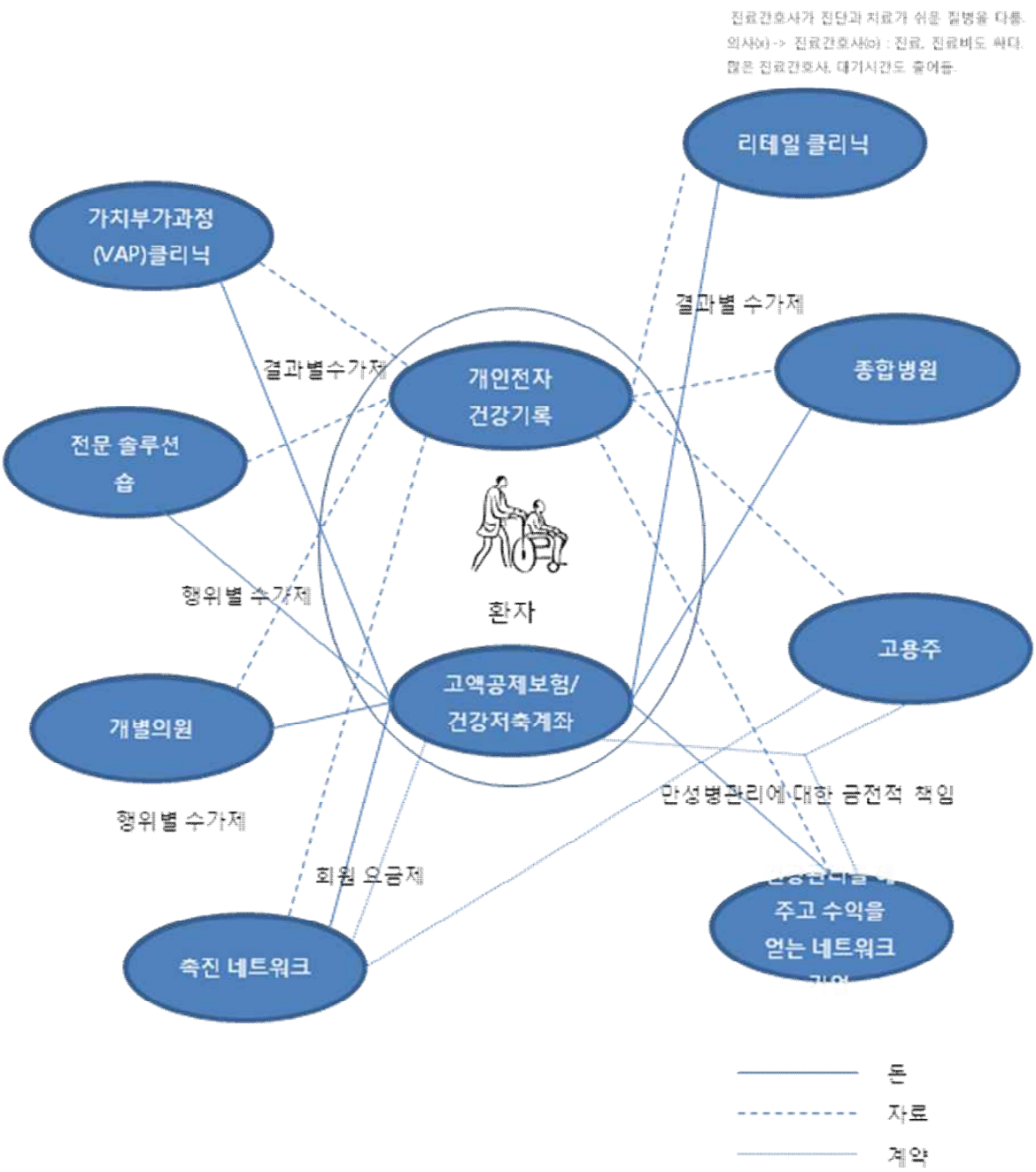


그림 6 미래의 의료서비스 체계

하지만, 미래에 다양한 의료보험제도와 다양한 형태의 의료기관이 생성되면서 실제 자료와 계약관계, 돈의 흐름은 매우 다양한 형태로 구현될 것이다.

현재의 의료서비스 체제는 보험회사와 환자, 고용주간의 일괄계약에 의한 방식으로 데이터 교환이 복잡하지 않으며, 정해진 방식으로 의료서비스가 구현되어 있다.

10. 리퍼런스 아키텍처의 목표

실무아키텍트 포럼의 목표는 이러한 복잡한 의료서비스 체계를 가지는 미래의 의료서비스 환경을 고려하여, 실제 사용가능하며, 참조가 가능한 아키텍처를 정의하는 것이 그 목적이었다.

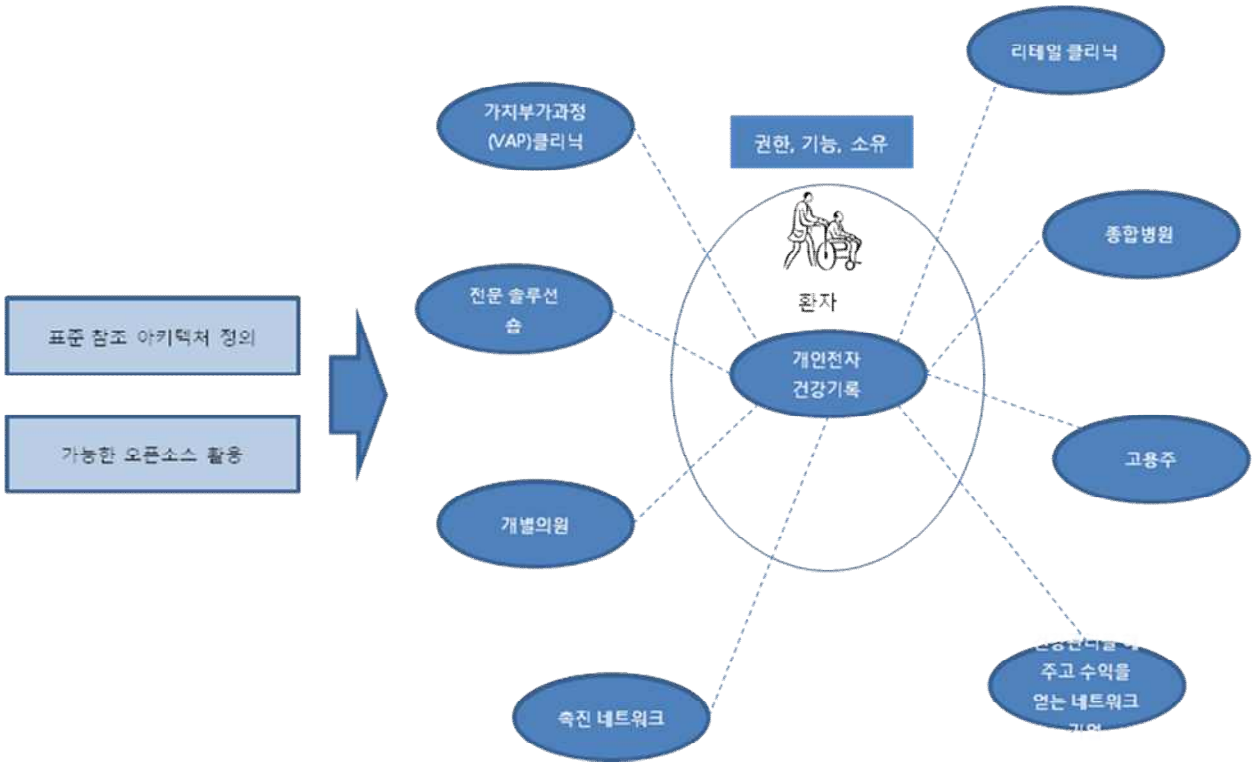


그림 7 리퍼런스 아키텍처의 목표

다만, 이 상태에서 실제 데이터의 흐름만을 고려한 것으로써, 돈과 계약관계에 대해서는 고려하지 않고, 의료정보시스템의 교환에만 그 초점을 맞추어 디자인하는 것을 목표로 하여 디자인하는 것을 목적으로 한다.

11. 리퍼런스 아키텍처의 주요 고려사항

리퍼런스 아키텍처는 다음의 5가지를 모델 디자인의 최우선 고려사항으로 하고 있다. 그것은 비용, 복잡도, 속도, 클라우드 호환성, 보안의 측면이다.

11.1. Cost (비용)

가. 가격모델, SLA(Service-Level Agreement)

나. 데이터 전송비용

11.2. Complexity(복잡도)

가. 요구사항의 해결책

나. 간결함

다. 복잡성은 장애대응과 유지보수의 어려움을 증가한다.

11.3. Speed(속도)

가. 시스템의 속도와 반응속도의 유연함을 고려한다.

11.4. Cloud Portability(클라우드 호환성)

가. 로드밸런싱, 데이터베이스 서버의 실제 구현을 고려한다.

나. 아키텍처 전환이 자유로워야 한다.

11.5. Security(보안)

가. 실제 사용가능한 암호화 방법을 제공한다.

12. 리퍼런스 아키텍처의 주요 참조내용

리퍼런스 아키텍처를 고려하면서, 다음의 기본적인 아키텍처에 대해서도 충분하게 고려를 하였다.

그것은, 실제 의료정보시스템을 구축하는 환경에서 아주복잡한 아키텍처만을 고려하여 디자인한다면, 실제 활용할 수 없는 사례가 만들어질 수 있다는 점을 충분하게 고려하였다.

기본적인 아키텍처는 다음의 3가지로 정의할 수 있다.

- **Single ‘All-in-one’ Server**
  - LAMP(Linux, Apache, Mysql, PHP) 하나의 Server
- **Single Cloud Site Architectures**
  - 표준적인 3-Tier 웹 사이트 아키텍처.
  - 각 Tier마다 한대이상의 서버
    - Load Balncing Server
    - Application Server
    - Database Server
- **Non-Redundant 3-Tier Architure**
  - 비용과 리소스 절약. 이중화(x)
  - 개발목적, 테스트

이러한 기본적인 아키텍처의 형태로도 구성이 되지 못하는 의료정보시스템의 환경도 상당히 많으므로, 이에 대해서도 참조할 수 있는 형태로 디자인되도록 노력하였다.

13. System Techincal Architecture

리퍼런스 아키텍처를 고려하면서, 다음의 사항들에 대해서도 충분한 검토를 진행하였다. 그것은, 과거의 디자인방식과 개발방법에 대해서도 정의를 한것이다.

- 과거의 디자인방식
  - WebLogic / JBOSS - J2EE
- 가장 심플한 방식
  - WebServer + WAS + RDBMS
- 주요 프레임웍
  - Spring, iBatis / Hibernate, Struts

- 어떤 목적으로 디자인할 것인가?
  - OLAP (데이터 분석 용도)
  - OLTP (온라인 트렌잭션 처리)

- 주요이슈
  - 대용량 이미지, DICOM정보등을 혼용할 수 있어야 함.
  - 추후 보안관련 레이어를 충분하게 포용할 수 있어야 함.
  - 클라우드 기반위에서 가능 하는 것도 고려해야함.
  - 최대한 오픈 소스 기반위에 디자인 되어야 함.
  - 주요한 성능 및 기능 이슈에 대한 고려가 있어야 함.
  - 개념 모델로서 사용이 가능해야 함.
  - 시스템의 확장을 위한 확장성을 최대한 고려하여야 함.
  - 각각의 Layer는 충분하게 모듈로 분리되어야 함.

이상의 이슈들은 의료정보시스템에서 다루고자 하는 아주 기본적인 이슈와 요구사항들을 정의한 것으로 해당 환경위에서 시스템을 디자인하는 것을 원칙으로 한다.

14. Technical System Architecture

이 프로젝트에서 고려해야 할 품질요소는 다음과 같이 정의할 수 있으며 다음의 그림에서 해당되는 아키텍처의 전체적인 모습을 표현하고 있다.

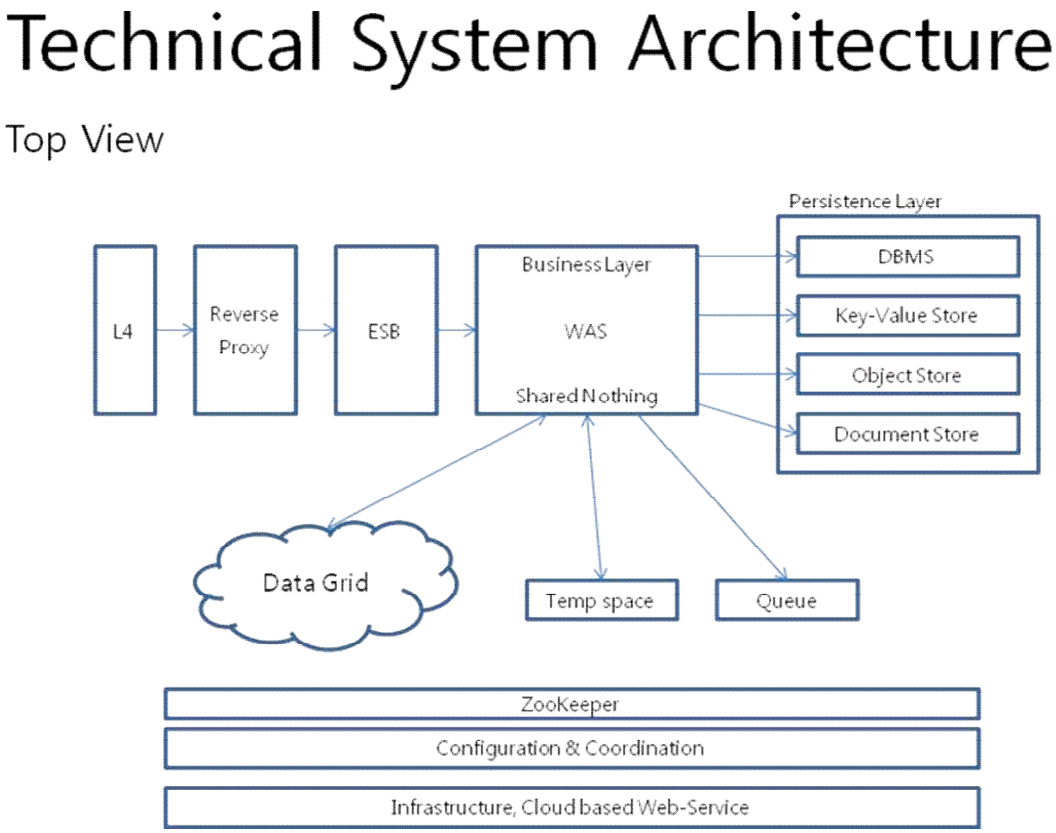


그림 8개념 아키텍처

가. Reverse Proxy Layer - Routing & Load Balacing

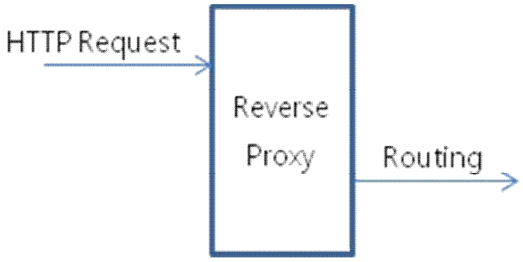


그림 9 Reverse Proxy Layer

이 계층은 Reverse Proxy 계층으로 주요한 기능은 다음과 같이 정의할 수 있다.

- **Request Loging**
- **Authentication & Authorization**
- **back Request**

들어오는 HTTP Request에 대한 관문 역할을 하며. 이 Reverse Proxy에서는 초기 Request에 대한 Logging을 처리한다. 중요한 Authentication & Authorization 처리를 수행하고, 뒷쪽에 Request를 보낼때, 뒷단의 Node로 Routing 또는 Load Balancing을 하는 것을 주요 기능으로 한다.

뒷단 클러스터가 업무에 따라서 여러 클러스터로 나뉘므로 이에 대한 라우팅을 수행하거나, 같은 업무에 대해서는 단일 클러스터에 대해서 여러 서버에 대한 로드 밸런싱을 수행한다. 뒤에 살아 있는 서버에 대한 리스트나 클러스터에 대한 정보는 ZooKeeper에 저장하며, Scale In/Out시 또는 장애시에는 이 정보를 ZooKeeper에 업데이트 하여, ZooKeeper에 저장된 클러스터 정보를 기준으로 라우팅을 한다.

사용할만한 솔루션으로는 Apache, NginX, HA Proxy등이 있다.

성능 상으로는 NginX가 가능 높으나. NginX는 대용량 HTTP Request에 대해서 아직 제대로 지원하지 않는 부분이 있다.(지속적인 검토는 필요함.). 결정적인 이유는 파일 업다운 로드의 경우 성능이 급격하게 떨어지는 부분이 있다. 이 부분은 대용량 DICOM정보 전달시에 성능저하가 발생할수 있다. 그러므로, 현재 상태에서는 **Apache**가 가장 유력하다.

하지만, NginX는 외부 시스템과의 신속한 정보교환에는 매우 효과적이다. 현재 상태에서 의료요약정보를 전달하거나, EDI와 같은 필요메시지만을 전송하는 경우에는 매우 효과적으로 사용할 수 있다. 이때에는 위에서 설명한 ZooKeeper연동이나 Routing 기능들은 module을 구현하여 plug-in해야 한다. (대표적으로 NGINX - naver.com / facebook.com / twitpic 사용하고 있다.).

나. Enterprise Service Bus (ESB) Layer - Cross Cutting Concern, Mash up,Routing,MEP (Message Exchange Pattern Converting),Integration, SLA management,Protocol Converting

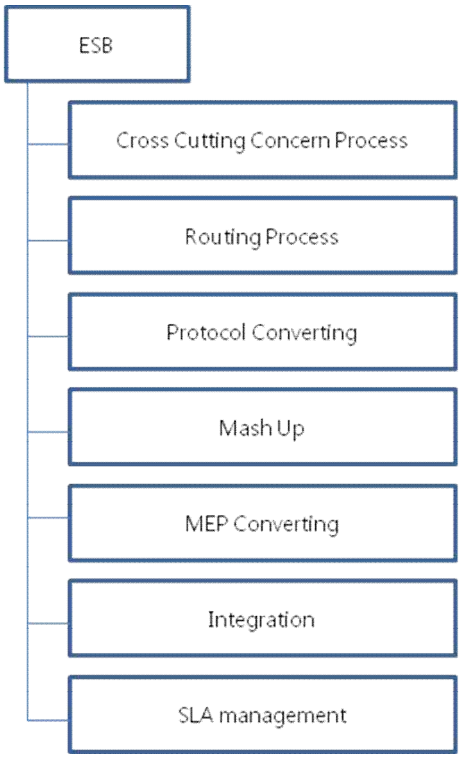


그림 10 ESB

이 계층은 당장 필요한 부분은 아니나, 추후 대규모 서비스를 위해서는 필수적으로 만들어져야 하는 부분이다. 초기의 프로토타입 개발시에는 Layer만 존재하는 것으로 충분하다. 이 Enterprise Service Bus 는 시스템으로 들어오는 메세지에 대해서 좀 더 확장된(진보된) 기능을 제공하는 계층으로, SOA (Service Oriented Architecture)에서 만들어진 개념을 적용한 계층이다.

이 레이어의 중요한 키 포인트는 **성능**이다. BY PASS의 경우 **10~50ms** 이내에 통과(IN/OUT 포함), 몬가 작업을 할 경우에는 **100ms** 이하에 통과되어야 한다. (이보다 느린경우에는 문제가 야기될 수 있다.)

전체적으로 ESB 계층에서 다루어야 하는 일들은 다음과 같다.

(1) Cross Cutting Concern 처리 : Cross Cutting Concern는 횡종단 처리라고 하는데, 모든 메세지에 대해서 뒷단의 비지니스 로직이 공통적으로 처리해야 하는 부분을 이야기 한다. 대표적인 예로, Logging, Authentication & Authorization 등이 있다. 의료정보의 생성과 교환, 변경등에 대한 전반적인 영역을 모두 처리하도록 하는 중요한 부분이다.

(2) Routing 처리 : Reverse Proxy 보다 향상된 Routing 기능을 제공한다. 보통 Reverse Proxy에서는 HTTP Header나 URI등의 최소한의 정보를 바탕으로 라우팅을 하는데, 이 ESB 계층에서는 Message 본문의 Header나 Message 자체의 내용을 가지고 라우팅을 한다. 세부적인 기능들을 구현하고 처리하는 부분으로 활용한다.

(3) Protocol Converting : 전체적인 ESB 계층에서는 또한 Protocol 변환을 할 수 있다. 예를 들어 뒷단의 비지니스 컴포넌트가 XML/REST를 지원하는데, 전체 표준을 JSON/REST를 사용한다면, ESB 계층에 프로토콜 변환 기능을 넣어서 JSON to XML 변환을 수행할 수 도 있다.

대표적으로 DICOM to Jpeg/png와 같은 서비스에 사용할 수 있다. Header에 의료기관 고유의 헤더를 삽입하는 등의 일이 발생할 수 있는데. 이와 같은 경우에 사용한다.

(4) Mash Up : Mash Up은 뒤의 비지니스 로직 여러개를 합쳐서 하나의 비지니스 로직을 만들어 내는 것을 이야기 한다. 쉬운 예로 기존 서비스가 "구매" 라는 Function이 있었는데, "포인트 적립" 이라는 기능이 새롭게 추가 되었을때, 비지니스 로직 자체를 변경하는 것이 아니라 기존 "구매" 라는 기능에 +"포인트 적립" 이라는 기능을 Mash up으로 더해서 기능을 변경하는 것이다. 이렇게 하면 비지니스 로직 변화 없이 새로운 기능을 구현할 수 가 있다. 이 부분은 현재 적극적으로 Node.js를 활용하는 것에 대해서 고민중이다. 실제, 의료정보시스템의 기능 구현시에 Mash Up을 통하여 내부적으로 구현되는 경우에 사용이 가능하거나, CP와 같은 영역에서도 충분하게 이 영역을 활용할 수 있다.

(5) MEP Converting : MEP란 Message Exchange Pattern의 약자로 메세지의 호출 방식을 이야기 한다. 쉽게 말하면 Sync,Async 와 같은 호출 방식을 정의하는데, 비지니스 로직이 Long Running 하는 Sync 형

식의 서비스 였을 때, ESB를 이용하여, Sync 호출을 Async 형태로 변경할 수 있다. (ESB에서 응답을 먼저 보내고, ESB에서 비지니스 컴포넌트로 ASync로 보내는 형태)

(6) Integration : 타 시스템과의 통합이나 연동을 이야기 한다. 일종의 EAI (Enterprise Application Integration) 기능인데, 앞에서 언급된 Mash up + Protocol Conversion + MEP Converting을 합쳐놓은 기능과도 비슷하다.

크게 원내 시스템간의 통합과 원외 시스템과의 통합 등으로 나뉘어지며, 대내 시스템과의 통합은 Legacy (SAP와 같은 ERP, Siebel과 같은 CRM과 같은 패키지 형태의 Application)과 통합 하는 경우가 많으며 이런 경우 전용 아답터를 사용하는 경우가 많다. 혹은, 데이터베이스의 테이블을 이용하는 경우도 많다.

대외 시스템 통합의 경우 예를 들어 정보교환과 같은 PUSH 서비스 등과 통합하는 경우이며 이 경우 필요에 따라 프로토콜 변환이나 Authentication & Authorization 처리를 하며, 특히 과금이 연동되는 경우에는 향후 Audit을 위해서 로그를 기록하고 향후 비교하는 경우가 많다.

이는 향후의 시스템 확장을 고려하여, 충분하게 고려되어야 한다.

(7) SLA management : SLA (Service Level Agreement)로, Service의 품질을 보장 하는 기능이다. 정확하게는 SLA를 보장한다기 보다는 SLA에 문제가 생겼을때 이를 빠르게 감지하여 후처리를 할 수 있게 한다.

ESB는 시스템으로 들어오는 모든 메세지에 대한 관문 역할을 하기 때문에 응답 시간이나 TPS에 대한 변화가 생겼을때 이를 검출할 수 있는 단일 지점으로, 장애 상황에 대한 검출이 있었을때 이에 대한 후

처리를 하도록 관리자에게 통보할 수도 있고, 또는 ZooKeeper를 통해서 성능이 떨어지는 노드들에 대한 Scale Out등을 지시할 수 있다.(매우 중요한 기능임.)

ESB는 설계시에 적용을 해놓으면 후에 시스템의 변화가 있을 경우에 도움이 많이 되는 계층이다. 시스템 초기 운영시에는 오히려 큰 이득을 보지 못한다. 왜냐하면 처음에는 모든 비지니스 컴포넌트가 초기 요구 사항에 맞춰서 구현이 되었고, 위의 기능들은 시스템을 운영하면서 요구사항이나 환경적인 변화에 따라 발생하는 요구 사항이기 때문이다.(다만, 확장되는 것에 대한 충분한 대비책을 이곳에서 가진다.)

앞에서도 언급했으나, ESB는 메세지가 지나가는 중간에 위치 하기 때문에 전체 시스템의 성능에 영향을 주게 되기 때문에 성능에 각별한 신경을 써서 디자인을 해야 하며, 특히 메세지의 파싱하는 과정과 메세지 자체 설계에 신경을 많이 써야 한다. 예를 들어 일반적인 경우 메세지의 BODY 부분을 파싱할 일이 없는데 모든 요청에 따라서 BODY 부분을 파싱하게 한다면 이에 대한 오버로드가 상당히 크게 된다.(주요한 작업공간은 대부분 메모리에서 처리하도록 디자인 되어야 한다.)

이 기능은 의료정보의 전달과 안전한 관리를 위해서 필수적으로 필요한 기능으로 확장되어진다.향후, HIPPA의 규정과 결합하여 확장될 가능성이 크며, 보안영역도 이 부분에 설치 운용되어진다.

사용 가능한 솔루션으로는 **Apache Mule**이나 **Oracle사의 Oracle Service Bus**등이 있다.

가장 재미있는 장비중의 하나는 Oracle Service Bus 제품중에 XML 기반의 메세징을 파싱하는 부분을 Hardware로 구현해놓은 제품이 있다. Oracle Service Bus도 내부적으로 JAXP 기반의 XML Parser를 이용하는데, 이 구현 부분을 ASIC으로 구현해 놓은 제품이 있는데 이 제품의 경우 메세지 처리 속도를 많이 높일 수 있다.(문제는 비용이다.)

다. WAS Layer - Business Logic

이 계층은 비지니스 로직을 핸들링 하는 계층이다. Web Application Server (WAS)로 구현이 가능하며, 고속 멀티플렉싱 기반의 고속 처리가 필요한 경우나 대규모 Stateful Connection이 필요한 경우에는 Netty와 같은 네트워크 서버를 사용하여 디자인한다.

이 계층 구현시 중요한점은 Shared Nothing 아키텍쳐를 적용하는 것을 원칙으로 한다. Shared Nothing 이랑 WAS 인스턴스끼리 클러스터링등을 통해서 묶지 않고 각각의 WAS를 독립적으로 돌아가게 설계하는 것이다. (대표적으로 Session Clustering을 사용하지 않는것)

이렇게 하는 이유는 특정 인스턴스가 장애가 났을 때 클러스터를 타고 전파되는 현상을 방지하고 또한 횡적인 확장 (Horizontal Scalability)를 보장하기 위해서이다.

이 영역에 대해서는 실제 구현되는 비즈니스 로직과 관련하여 보다 세밀하게 디자인하도록 한다.

라. Async message Handling

WAS 계층이 Sync 형태의 동기 (Request/Response) 메세지를 처리한다면, 비동기 메세징 처리나 Publish/Subscribe와 같은 1:N 기반의 비동기 메세지 처리를 하는 계층이 필요한 경우가 있다. 예전에는 MQ나 JMS를 많이 사용했으나, 근래에는 좀더 향상된 프로토콜인 AMQP를 기반으로 한 RabbitMQ가 많이 사용된다.

대표적으로 사용되는 RabbitMQ의 경우에도 수억명의 사용자를 커버하기에는 클러스터의 확장성이 문제가 있기 때문에 이런 경우에는 MySQL등의 RDBMS의 테이블을 큐 처럼 사용하고, 메세지를 읽어가는 부분을 Quartz등을 이용해서 주기적으로 읽어가서 처리하는 구조로 만들게 되면 확장성을 보장할 수 는 있으나, 복잡한 비동기 메세징 (에러처리, Pub/Sub)을 구현하기에는 난이도가 높기 때문에, RabbitMQ를 복수의 클러스터로 묶는 Sharding이나 분산큐(Distributed Queue) 개념을 고려할 필요가 있다.

이 부분은 원내 시스템용도와 원외시스템용도에 따라서 세부적으로 정의하도록 한다.

마. Temporary Storage Layer - Temporary space

이 계층은 Temporary Storage Layer - 작업 공간으로 활용되는 영역이다. 자세하게 설명한다면, 이 작업 공간은 4번의 WAS들이 서로 데이타를 공유할 수 있는 "휘발성", 작업 공간이라고 설명할 수 있다.

필수 조건은 높은 성능을 보장해야 하며, 모든 WAS Node가 접근할 수 있어야 한다. 저장 매체가 Memory냐, Disk냐에 따라서 다음과 같이 나눠볼 수 있다.

1) Data Grid (Memory)

데이타 그리드는 쉽게 생각하면 자바의 HashTable 같은 **Key/Value Store** 기반의 메모리 Store이다. 이 그리드는 클러스터 구성을 통해서 용량 확장이 가능하고, 별도의 서버 클러스터로 구성되어 여러개의 WAS 노드들이 접근할 수 있다. 일종의 WAS간의 공유 메모리이다.

솔루션으로는 Oracle Coherence (예산만 넉넉하다면 이걸 쓰는게 맘편하다), Redis, memecahed, Terracota 등이 있다.

2) Working Space (DISK)

트랜잭션을 처리하다 보면, 종종 임시적인 작업 공간이 필요 할때가 있다. 예를 들어 DICOM 이미지 화일을 전송하는 파일 서비스의 경우에, 해당 서비스는 이미지 파일을 하나 올리면, 모바일 디바이스의 화면 해상도에 맞게 필요한 썸네일 이미지나 모바일 관련 필요한 내용을 재생산한다. 이런 작업을 하기 위해서는 이미지 파일을 저장하기 위해서 임시로 저장해놨다가 썸네일을 추출하는 공간이 필요한데 이를 임시 작업 공간이라고 한다.

데이타 그리드와 마찬가지로, 여러 노드들이 해당 공간을 공유할 수 있어야 한다. 그래서 NFS (Network File System)이 많이 사용되며, Gluster와 같은 소프트웨어 기반의 NFS나 NetApp社의 NFS appliance server (하드웨어) 등이 있다.

바. Persistence Layer

이 레이어는 의료정보를 영구 저장 공간이다. 영구 저장 공간은 우리가 일반적으로 생각하는 데이타가 저장되는 공간이라고 보면된다. 쉽게 예를 들 수 있는 공간으로는 데이타 베이스와 파일 시스템을 들 수 있다. 이러한 영구 저장소는 대용량 B2C 시스템의 유행과 함께 새로운 DBMS들이 등장하였는데, DBMS 측면에서는 Key Value Store 기반의 NoSQL이나, 대용량 파일을 저장할 수 있는 Object Store등을 그 예로 들 수 있다.

1) Relational Data

개체간의 관계가 있는 경우에 대한 데이타를 관계형 데이타라고 하고, 이를 핸들링 하기 위해서는 관계형 데이타 베이스 RDBMS를 사용한다. 우리가 지금까지 일반적으로 사용해왔던 데이타 베이스가 이

RDBMS이다. RDBMS는 대용량 서비스를 위해서는 태생적인 한계를 가지고 있는데, 예를 들어 MySQL의 경우 하나의 데이타베이스에서 저장할 수 있는 레코드의 수가 10억개 정도가 최적이다.

이런 문제를 해결하기 위해서는 대용량 시스템에서 몇가지 기법을 추가로 사용하는데 "Sharding" 과 "Query Off Loading"이다.

Sharding이란, 데이타의 저장용량의 한계를 극복하기 위한 방안으로 데이타를 저장할때 데이타를 여러 데이타 베이스에 걸쳐서 나눠 저장하는 방법이다. 예를 들어 "서울","대구","대전"등 지역별로 데이타베이스를 나눠서 저장하거나(이를 횡분할 Sharding) 또는 10대,20대,30대 식으로 데이타를 나눠서 저장하는 방식(이를 수직분할 Sharding)을 사용한다. 이러한 Sharding은 데이타베이스 계층에서 직접적으로 지원하기가 어렵기 때문에, 애플리케이션 레벨에서 구현해야 한다.

다음으로 Query Off Loading이라는 기법으로, 이 기법은 성능의 한계를 높이기 위한 기법이다.

"Master DB → Staging DB → Slave DB 1,Slave DB 2,....N"

A. Create/Update/Write/Delete는 Master DB에서 수행하고

B. Master DB의 데이타를 Staging DB로 고속 복사한후

C. Staging DB에서 N개의 Slave DB로 데이타를 복사한다.

D. Read는 Slave DB에서 수행한다.

일반적인 DBMS 트렌젝션은 10~20% 정도가 Update성이고, 나머지 80~90%가 Read성이기 때문에, Read Node를 분산함으로써, 단일 DBMS 클러스터의 임계 처리 성능을 높일 수 있다.

이때 Master/Staging/Slave DB로 데이타를 복제하는 방식이 매우 중요한데, 여기서 일반적으로 사용하

는 방식을 CDC (Change Data Capture)라고 한다. RDBMS는 데이타 베이스 장애에 대한 복구등을 위해서 모든 트렌젝션을 파일 기반의 로그로 남기는 데 이를 Change Log라고 한다. CDC는 이 Change Log를 타겟 DB에 고속으로 복사해서 다시 수행(Replay)하는 형태로 데이타를 복제한다.

MySQL의 경우 Clustering에서 이 CDC 기능을 기본적으로 제공하고 있고, Oracle의 경우 Oracle Golden Gate라는 솔루션을 이용한다. (비싸다..) 중가격의 제품으로는 Quest의 ShareFlex들을 많이 사용한다.

2) Key/Value Data

다음으로 근래에 들어서 "NoSQL"이라는 간판을 달고 가장 유행하는 기술중의 하나가 Key/Value Store이다. 데이타 구조는 간단하게 Key에 대한 데이타(Value)를 가지고 있는 형태이다. RDBMS와 같이 개체간의 관계를 가지지 않는다.

오로지 대용량,고속 데이타 억세스,데이타에 대한 일관성 에만 초점을 맞춘다. (이중에서 보통 2개에만 집중한다. 이를 CAP 이론 - Consistency, Availability, Performance)

이 기술은 태생 자체가 B2C 서비스를 통해서 탄생하였다. 블로그나 트위터, 페이스북 처럼 데이타의 구조 자체가 복잡하지 않으나 용량이 많고 고성능이 필요한 데이타들이다. 태생 자체가 이렇기 때문에 복잡한 관계(Relationship)을 갖는 복잡한 업무 시스템에는 잘 맞지 않는 경우가 많으며, 트렌젝션 처리나 JOIN, SORTING 등이 어렵기 때문에 애플리케이션의 구현 복잡도가 올라간다.

3) Object Data

Object Data는 File과 같이 대용량 데이타 파일 저장을 할 수 있는 Storage이다.

10M,1G와 같은 대용량 파일을 저장할 수 있는 저장소로, Amazon의 S3, Openstack SWIFT등이 대표적인 예이며, 하드웨어 어플라이언스 장비로는 애플의 iCloud로 유명해진 EMC의 isilion등이 있다.

Object Data 저장에 있어서 중요하게 생각하는 부분은 대용량의 데이타를 저장할 수 있는 용량에 대한 확장성과 데이타 저장에 대한 안정성이다. 이러한 Object Data는 Quorum이라는 개념을 적용하여, 원본을 포함하여 N개의 복사본을 유지한다. 일반적으로는 N+3 (3개의 복사본)을 저장하여 데이타에 대한 안정성을 보장한다.

4) Document Data

Document Data는 Key/Value Store에서 조금 더 발전한 데이타 저장 방식으로 Key 자체는 동일하나 Value에 해당하는 부분이 Document가 저장된다. Document 는 JSON이나 XML 문서와 같이 구조화된 데이타를 저장한다.

RDBMS가 다양한 select, where, group, sorting,index 등 여러가지 데이타에 대한 기능을 제공한다면, Key/Value Store는 이런 기능은 거의 제공하지 않는다. Document Data를 저장하는 제품들은 RDBMS와 Key/Value Store의 중간정도에서 데이타에 대한 핸들링 기능을 제공한다. (부족한 Indexing 기능, 부족한 Group 기능, 부족한 Sorting 기능 등)

대표적은 솔루션으로는 MongoDB,CouchDB, Riak등이 있다.

요즘 들어서 자주 사용되는 대표적인 Persistence Store에 대해서 간단하게나마 집고 넘어갔지만, 사실 이 보다 더 많은 형태의 Persistence Store들과 기능들이 있다.

사. Configuration management & Coordinator

대용량, 분산 시스템으로 발전하면서 풀어야되는 문제중의 하나가 "분산되어 있는 노드들에 대한 설정(Configuration)정보를 어떻게 서로 동기화하고 관리 할것인가? (이를 Configuration Management라고 한다.) " 인다. 거기에 더해서 클라우드 인프라를 사용하면서 "전체 클러스터내의 서버들의 상태를 모니터링 해서, 서버의 수를 느리고 줄여야 하며 서버들간의 통신을 중재해야 한다. (이를 Coordination 이라고 한다.)"

여기에 필요한 기능이 작은 량의 데이타(Configuration Data)를 여러 서버가 공유해서 사용할 수 있어야 하며, 이 데이타의 변화는 양방향으로 클러스터 노드내에 전해져야 한다. 즉 Configuration 정보를 각 서버들이 읽어올 수 있어야 하며, 이 Configuration 정보가 바뀌었을 경우 다른 서버들에게 데이타가 변했음을 통지해줄 수 있어야 하며, 중앙 집중화된 Configuration 정보 뿐만 아니라, 서버의 상태가 변했음을 다른 서버들에게 빠르게 알려줄 수 있어야 한다.

이런 역할을 하는 대표적인 솔루션으로는 ZooKeeper 많이 사용된다.

아. Infrastructure

마지막으로 이런 소프트웨어 스택을 구동하기 위한 하드웨어 인프라가 필요한데, 예전에는 일반적인 서버를 Co-Location이나 Hosting 형태로 사용하는 것이 일반적이었으나, 요즘은 가상화 기술을 기반으로 한 클라우드 (Infrastructure as a service)를 사용하는 경우가 많으므로, 이에 대한 고려를 충분하게 해

야 한다.

클라우드의 특징은 "Pay-as-you-go" 로 자원을 사용한 만큼에 대해서만 비용을 지불하는 구조이다. 서비스에 필요한 리소스를 구분하여, CPU를 사용한 만큼, 디스크를 사용한 만큼, 네트워크 대역폭을 사용한 만큼만 비용을 지불하는 방식이다.

일반적인 클라우드 인프라의 경우 Amazone WebService (AWS), Microsoft Azure, Google App Engine등이 대표적인 예인데, 이러한 클라우드의 장점은 Time To Market (시장 진입 시간)이 매우 짧다는 것이다. 앉아서 신용카드와 PC만 있다면 인터넷에 접속해서 30분내에 서버,디스크,네트워크등을 설정해서 사용할 수 있다. (하지만, 의료환경에 적합한 클라우드 환경은 이런 일반적인 클라우드 인프라를 사용하기에는 문제 있는 경우가 많다.)

그러한 이유는 클라우드 인프라는 Public 한 서비스 형태로 공유되서 서비스 되기 때문에 일반적인 호스팅과는 달리 성능등에 대한 한계를 가지고 있기 때문이다.

예를 들어 서버와 디스크간의 네트워크 대역폭이 보장되지 않기 때문에 디스크 IO가 많은 애플리케이션 (DBMS와 같은)에 대한 성능을 보장하기가 쉽지 않고, LAN 설정이 자유롭지 않기 때문에 UDP등을 이용해서 클러스터링을 하는 제품의 경우 클러스터링을 사용할 수 없는 경우가 있다. 이런 이유로, 클라우드위에서 구현되는 시스템의 경우에는 해당 클라우드의 기술적인 특징을 제대로 이해하고 구현해야 한다. 그래서, 이러한 요구사항/요구조건에 적합한 의료용 클라우드의 환경을 정의하거나 만들어야 할 필요가 있다.

하지만, 초기 프로토타입의 경우에는 이러한 클라우드가 효과적이다.

클라우드가 "Pay-as-you-go" 형태로 사용한만큼 비용을 지불한다는 것이 어떻게 보면 "싸다"라고 느껴질 수 있지만, 네트워크,IP 등등 모든 자원에 대해서 비용을 지불하기 때문에 실제적으로 계산해보면 싸지 않은 경우도 많고 기술적인 제약 때문에, 초기 시장 진입을 하는 경우에는 기존의 상용 클라우드를 사용하지만, 실질적으로 서비스를 구현하고 운용할 경우에는 다시 자체 데이타 센타를 구축하여야 한다. (예 소셜 게임 서비스인-Zinga, VOD 서비스인-Netflix)

운영 측면에서 인프라에 대한 관리를 클라우드 업체에 대행시킴으로써 얻는 이득도 있지만 불필요한 비용이 낭비되지 않게 클라우드 인프라에 대한 배포 구조를 끊임 없이 최적화 하는 노력도 필요하다.

자. 기술 아키텍처 디자인

지금까지 현재 사례가 존재하는 시스템을 기반으로 의료정보시스템에서 사용이 가능한 대용량 고성능 시스템에 대한 레퍼런스 아키텍쳐에 대해서 설명하였다. 분명, 확실하게 기술 구조는 변해가고 있다. 유행하는 기술도 변하고 있다. 현재의 대용량 시스템은 이런 구조로 구현하는게 하나의 모범 답안은 될 수 있으나, 대부분의 IT 시스템은 이런 대용량 아키텍쳐 구조 없이도 WAS + RDBMS 구조만으로도 충분히 구현이 가능하다고 할 수 있다. 실제, 이러한 대용량 아키텍처를 구사하는 경우는 극히 일부분이며, 의료기관간의 전체적인 연결고리를 고려한다면, 충분한 고민거리가 될 수 있다.

또한, 의료영상정보를 교환하기 위해서는 적은 데이터를 교환하는 것이 아니라, 대용량의 데이터를 교환하는 것이기 때문에 이와 같은 고민은 필수적으로 있어야 한다. 다만, 여기서 제안된 내용은 이제 1차적으로 정의한 내용이므로, 무수한 손질을 가하여, 필요한 기술들을 정의하는데 활용할 수 있는 근거자료로 사용될 것이다.

아키텍처의 교훈은 '**닭잡는데 소잡는 칼을 사용하지 않는다**'가 원칙이다.

15. 현재 의료정보시스템 모델과 지향하고자하는 시스템모델의 차이점

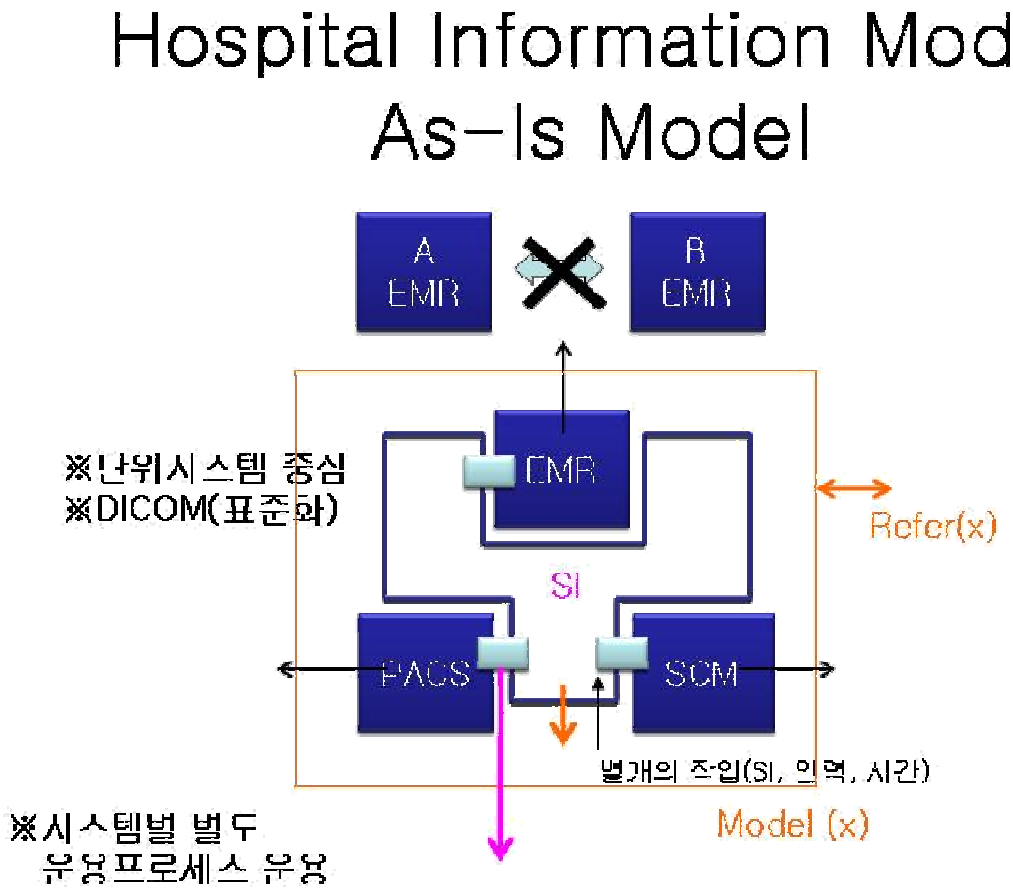


그림 11 현재 의료정보시스템 모델

현재의 의료서비스와 IT의 관계는 다음과 같이 간단하게 정의할 수 있다. 각각의 의료기관에 존재하는 의료정보시스템들은 대부분 몇가지의 큰 시스템으로 구성되어 있다. 대표적인 시스템들은 EMR, PACS, OCS 가 있으며 이를 보조하기 위한 SCM체계를 갖추고 있다.

이중에 PACS는 의료기기와 연동하기 위하여 DICOM이라는 산업표준을 잘 따르고 있으므로, 연동이 어렵거나 비용이 많이 추가되지는 않는다. 하지만, 기존의 EMR, OCS는 대한민국의 특성상 특이하게 구성되어 있다.

대한민국의 의료서비스 체계는 최저의 비용으로 의료서비스를 빠르게 운용하는데 집중화되어있고, 의료기관의 경영효율을 향상시키기 위해서, 의료비를 건강보험공단에 빠르게 청구하고 비용을 지불

받는 것을 중심으로 빠르게 진화하였다.
이처럼 PM/PA의 환자관리와 CPOE와 같은 의료행위에 대한 지시를 하는 시스템이 결합되어, OCS라는 대한민국의 독특한 청구시스템과 의료행위시스템에 결합되어진 시스템으로 발전하였고, 이를 기반으로 EMR의 개념이 추가되어진 시스템으로 현재 운용되고 있다.
대부분의 시스템들은 하나의 시스템들이며, 논리적으로 구분되어진 시스템의 구성형태는 가지고 있으나, 별도로 운용되거나 타 기관에서 만들어진 시스템이거나, 다른 소프트웨어 기업이 만든 소프트웨어의 경우에 상호 호환성은 전혀 확보를 못하고 있다.
그래서, 보통 대한민국의 의료기관들은 이 시스템을 통째로 업그레이드 하거나, 통째로 바꾸는 방법을 사용하여 시스템을 진화시킨다. 이외에 SCM이나 건강검진과 같은 별도의 소프트웨어들이 존재하나, 해당 시스템들은 의료기관 내부의 진료행위와는 별개의 영역으로 구분되어 진다.

해외의 시스템들은 각각의 영역에서 독자적으로 구성되어진 솔루션과 패키지의 형태로 해당 의료기관에 공급되는 형태이나, 대한민국의 의료서비스 체계에 적합하게 만들어진 의료정보시스템들은 하나의 시스템의 형태로 존재하고 있으며, 그 특이성 때문에 대부분의 소프트웨어들은 SI의 형태를 가지고 의료기관내에서 구축되는 방법을 사용하고 있다.

또한, 실제 의료정보시스템이 추구해야할 프로세스의 개선등의 지식공학이 접근하기 어려운 형태로 진행되었다. 그것은, 현재의 의료수가가 매우 저평가 되어 있으며, 해당 서비스에 대해서 비용적인 추가 없이, 빠르게 처리하는 것만을 중요시하기 때문에 시스템의 접근이나 설계는 매우 지식적인 방법이 필요 없는 상태로 대부분 진행이 된것 또한 사실이다.

문제는, 향후 의료서비스나 의료환경의 변화에 맞추어서, 변화를 일으켜야 하는 의료서비스환경에 어울리도록 시스템을 재설계해야 하는데. 현재의 상황에만 맞추고 있으면, 시스템의 발전이 문제가 되게 된다. 그래서, 실무아키텍트 포럼에서는 이러한 부분에 촛점을 맞추고 있다.

실무 아키텍트 포럼이 지향하고 있는 의료정보시스템은 SI형태로 소프트웨어가 구성되어지는 것이 아니라, 개별적인 패키지나 개별적인 서비스들이 좀더 진화할 수 있는 방향으로 전환하는 것을 목표로 하고 있다.

다음의 그림은 To-be로 생각하는 의료정보시스템의 미래의 모습이다.

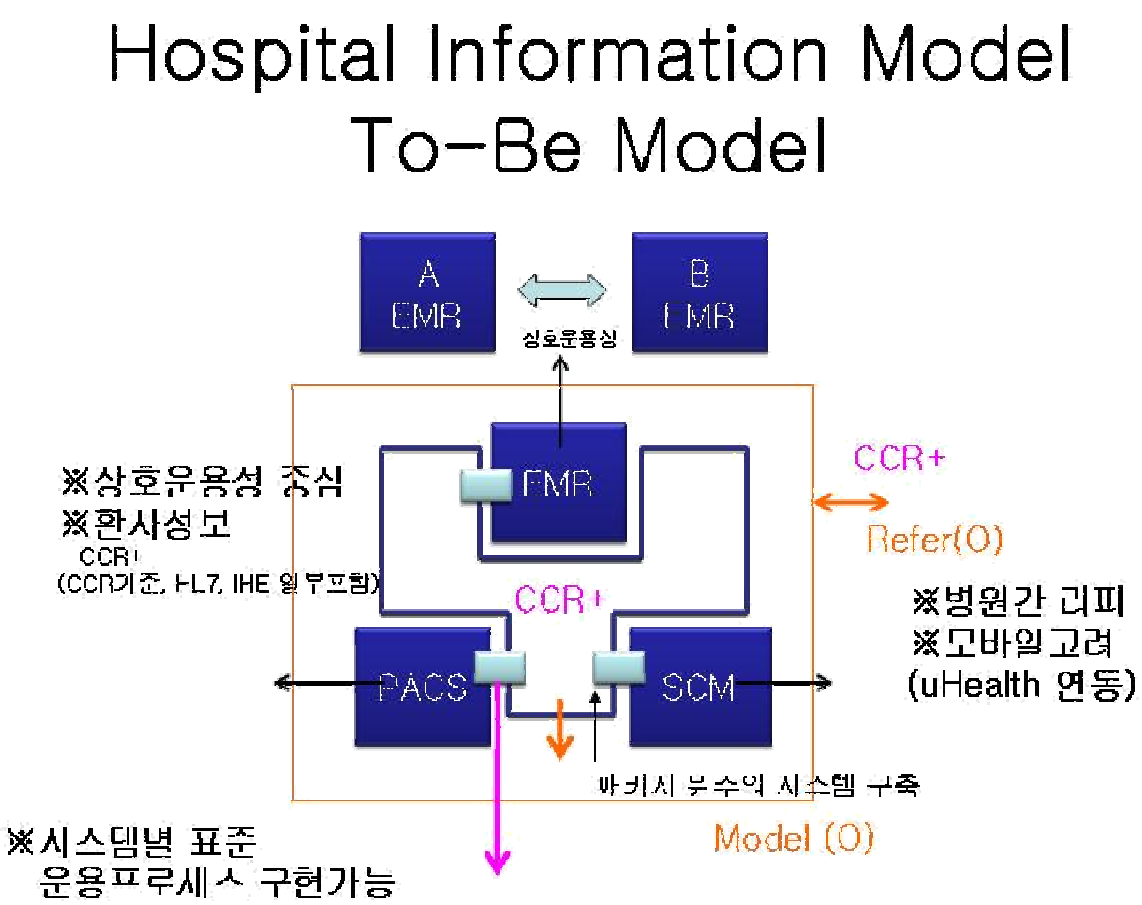


그림 12 향후 의료정보시스템 모델

각각의 시스템과 서비스들이 효과적으로 상호운용이 가능한 형태를 가지고 있는 모습이다. 이러한 시스템 환경은 각각의 서비스들과 소프트웨어들이 상호운용을 표준화된 형태로 지원하기 위한 방향들이다.

이러한 표준 호환을 높이기 위한 방법에 대표적인 방법은 HL7과 CCR과 같은 방법이다. HL7은 광범위한 의료정보교환을 표준화된 방법으로 제공한다. 또한, CCR은 의료기관간의 최소한의 의료정보를 전달할 수 있는 방법으로 XML로 정의하는 방법이다.

이곳에 서술한 HL7, CCR등은 형태와 환경에 따라서 선택되어져야한다. 그리고, 실제 고객이 원하는 의료정보시스템에 대해서 심도있게 고민하여야 한다.

실제 고객이 원하는 의료정보시스템의 4가지 원칙을 정의하자면 다음과 같다.

- **상호운용성의 확보**
  - 복잡하고 많은 소프트웨어와 하드웨어,
  - 고수준의 의료지식, 확장되는 행정직, 10년이상의 모델
- **기술종속적이지 않은 프레임웍**
  - 불투명한 미래에 대한 보장성
- **개발에 필요한 교육 및 유지보수관련 디자인**
  - 진행과 관련된 인력 공급 및 확보계획
- **최고의 기술이 아닌 적절한 품질요소**
  - 적절한 기능/비기능의 배분

이처럼, 고객들은 종속적이지 않고, 호환성이 높은 의료정보시스템을 요구한다. 이를 각 체계의 관점으로 디자인하면 다음과 같은 그림으로 형상화가 가능하다.

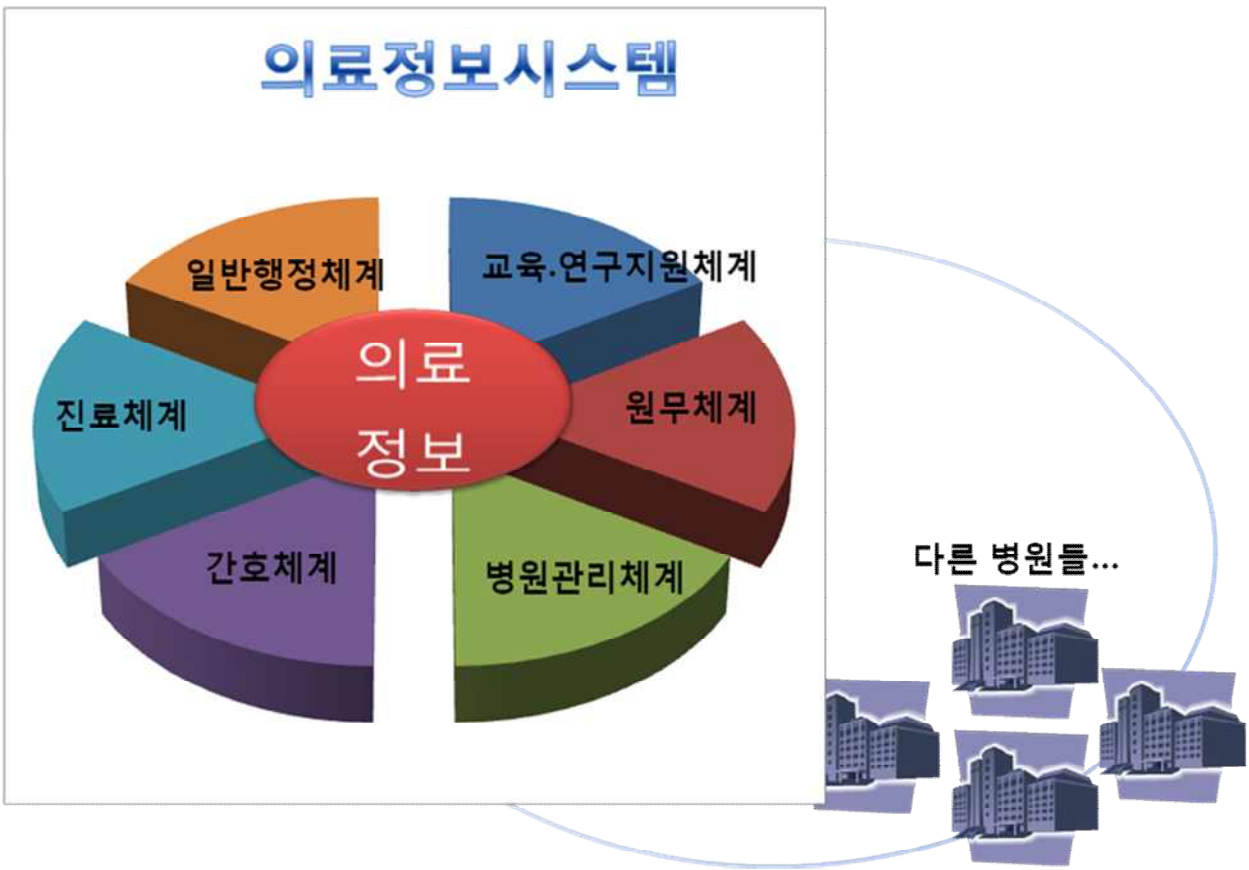


그림 13 고객이 원하는 의료정보시스템 형태

그림13에서 표현하고자 하는 것은 의료정보시스템을 통하여 다른 의료기관간에서 혼용할 수 있으며, 각각의 체계들을 통합하여 운용할 수 있는 형태를 의미한다. 단지, 논리다이어그램으로 써만의 의미가 아니다.

물론, 복잡하고 유연한 의료서비스의 체계를 모두 받아들여지는 정보체계를 하나로 설명할 수 없을 것이다. 그렇지만, 미래에 다양한 의료보험제도와 다양한 형태의 의료기관이 생성되면서 실제 자료와 계약관계, 돈의 흐름은 매우 다양한 형태로 구현될 것이기 때문에, 의료정보시스템의 체계는 매우 정교하게 디자인되어야 한다.

16. Medical Information System 개별 정보시스템

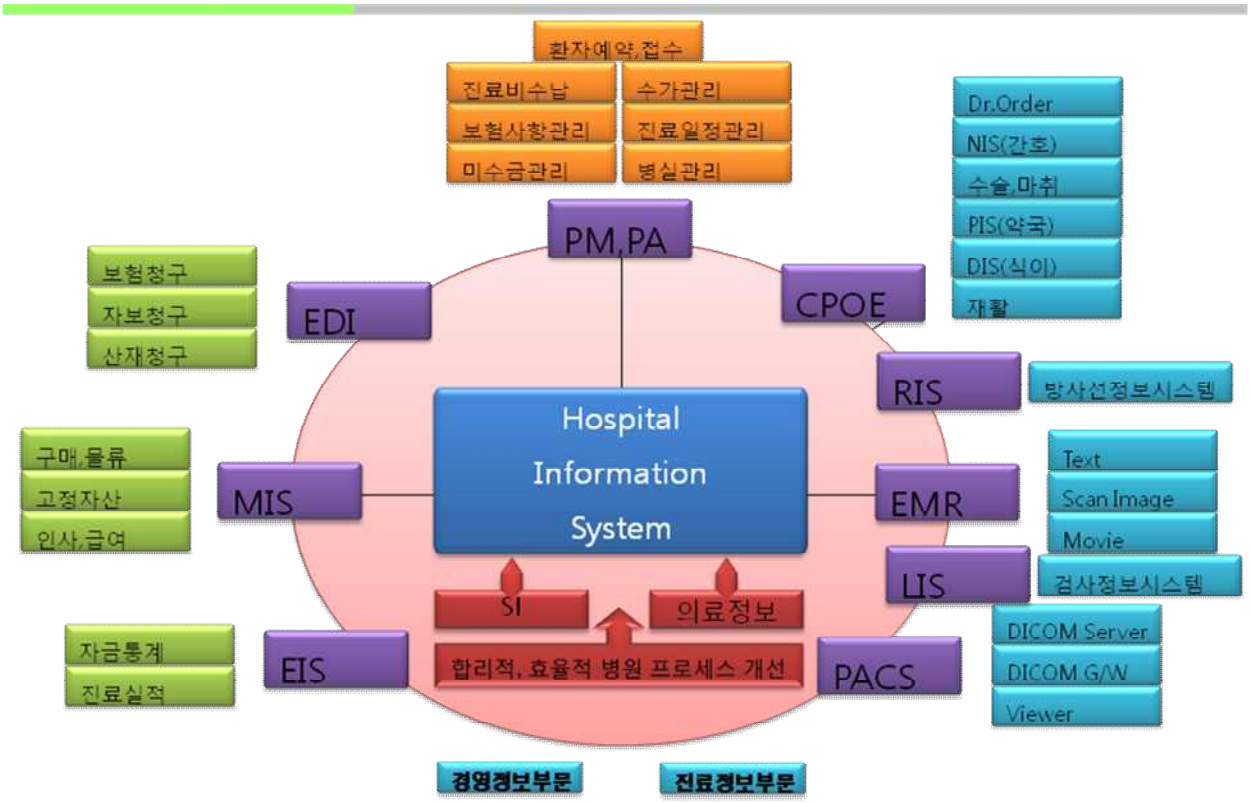


그림 14 개별 정보시스템

그림14에서 표현하고자하는 것은 HIS(Hospital Information System)으로 표현되는 전체적인 의료서비스 체계의 중요한 기능들만 나열한 것이다.

의료분과에서는 이러한 의료정보시스템의 전체적인 상황을 고려하여, 효과적으로 사용될 수 있는 아키텍처를 논의하고 고민하는 것을 가장 중요한 목표라고 정의하고 있다. 이러한, 환경을 구축할 수 있는 소프트웨어 아키텍처를 정의하는 것이다.

17. 의료분과의 소프트웨어 아키텍처 정의의 방향성

17.1. 소프트웨어 아키텍처에 대한 정의

소프트웨어 아키텍처에 대한 다양한 정의를 하면 다음과 같다.

아키텍처는

- ✓ 고-수준의 설계
- ✓ 시스템의 전반적인 구조
- ✓ 프로그램이나 시스템 구성 요소들의 구조
- ✓ 구성 요소들의 상호관계
- ✓ 설계와 시간에 따른 발전을 통제하는 원칙과 지침

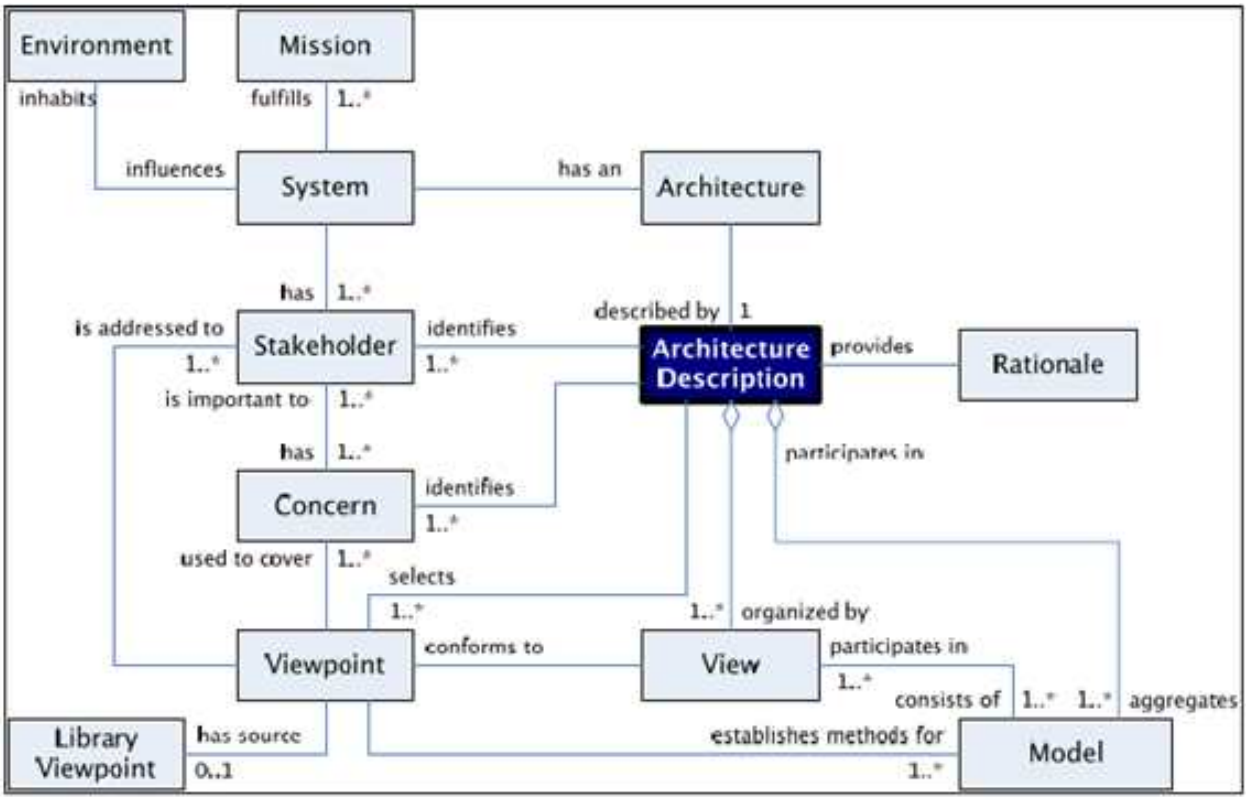
소프트웨어 아키텍처에 대한 다른 견해

- ✓ 발전하고는 있지만 아직도 미숙하므로, 일반적으로 용인된 정의가 존재하지 않음
- ✓ 일반적으로 기본적인 주제와 일관성이 있지만, 매우 다양한 세부가 존재하며 상호교환 되지 않음

이처럼 앞부분에 열거한 요구사항과 모델을 반영하기 위하여 의료정보시스템의 소프트웨어 아키텍처를 정의하는 방향성에 대해서 기술하고자 하며, 이를 연구하기 위한 모임이 실무아키텍트 연구포럼의 연구 방향성이다.

17.2. 아키텍처 기술 표준 - IEEE1471

SW 아키텍처 표현을 위해 SW 아키텍처의 구성요소와 관계를 일반화하여 SW 아키텍처 명세를 제공하는 아키텍쳐 기술서로써, 이를 표준화한것이 IEEE1471로 정의할 수 있다.



이는 소프트웨어 아키텍처링을 하기 위한 전반적인 프로세스에 대해서 모두 언급하고 있다. 이러한 단계에 따라서 의료정보시스템의 소프트웨어 아키텍처링도 동일하게 처리한다.

17.3. 배경지식 - Medical

의료정보시스템을 정의하기 위해서, 기본적인 정보들을 몇가지 정의한다. 보다 상세한 정보들은 다른 문서를 참고하기 바란다. 이곳에서는 최소한의 정의만 정리하겠다.

구분	내용
EMR	• 전자의무기록시스템 (Electronic Medical Record) • 병원 진료 기록을 전산화 한 System (Chartless)
OCS	• 처방전달시스템 (Order Communication System) • 병원 Order / 처방 정보를 전산화 한 System (Slipless)
PACS	• 의료영상저장전송시스템 (Picture Archiving Communication System) • 필름을 가지고 진단하고 판독하던 병원의 업무를 컴퓨터와 네트워크을 통하여 병원의 영상진단업무 (Filmless)
HL7	• HL7은 이와 같이 서로 다른 의료정보시스템 소프트웨어 애플리케이션 간에 정보가 호환될 수 있도록 하는 규칙의 집합이며 현재 북아메리카에서는 의료정보의 전자적 교환을 위한 사실상의 표준
DICOM	• 의료용 디지털영상 및 통신 표준인 DICOM은 의료용 기기에서 디지털영상 표현과 통신에 사용되는 여러 표준을 총칭하는 말로서 의료영상 디지털 정보를 다루고 저장하고 인쇄하고 전송하는 표준

18. 해결하고자 하는 문제

의료도메인의 소프트웨어 아키텍처를 정의하는 것에 있어서도, 가장 중요한 것은 '문제'를 어떻게 정의하느냐가 가장 중요하다. 이러한 점들은 '사용자(환자)'의 측면, '의료인/의료기관'의 측면, 으로써 2개의 시선으로 문제를 정의하여야 한다.

보통 의료정보시스템을 디자인하는 것에 있어서, '사용자(환자)'의 측면에서 요구사항을 분석하지 않는 것이 보통이지만, 현대의 의료정보시스템들은 '환자'의 요구사항들에 대해서도 직접적인 문제에 대해서 언급을 하여야 한다.

그것은 모든 의료서비스의 디자인은 '환자'를 중심으로 디자인하는 '환자중심의 의료서비스'를 중시하는 현대의 의료서비스 디자인의 중요철학 때문이다.

18.1. 문제정의 - 사용자 측면

사용자의 문제정의를 하기 위한 요소와 문제에 대해서 사용성과 위험도, 문제들에 대해서 조사하여 기술한다.

- **불편함**
  - 본인이 직접 병원 방문 후 CD 청구
- **방사선량 과다 위험**
  - 방사선량 과다 노출 우려
  - CT 1회 촬영 시 노출 방사선량은 1년간 일상생활에서 받는 방사선량의 25~50배 수준
- **시간적 / 금전적 문제**
  - 불필요한 재촬영으로 인한 금전적 손실

18.2. 문제정의 - 의료인/의료기관 측면

의료인/의료기관의 측면에서는 주요하고 상세한 내용으로 문제를 정의하도록 한다.

➢ **진료의 연속성 문제**

- 환자에 의한 수작업으로 병원간 영상 정보 교환이 이루어짐.
- 진료의 질적 개선의 한계 (의료 서비스 확대 한계)
- **의료영상 교환 비용 문제**
- 의료기관마다 다른 프로세스와 상이한 프로그램을 사용
- 외부 CD의 영상을 import 하기 위해서는 별도의 작업이 필요
- 사람에 의해 이루어지는 작업임으로 실수 발생 가능성이 높음
- 정보보호 및 보안
- 개인 CD 분실/도난으로 인한 개인 정보 유출 문제

19. 주요요구사항

의료정보시스템에 있어서도 가장 중요한 것은 이렇게 '문제'를 들어낸 이후에, '요구사항'에 대해서도 보다 자세한 관점으로 서술한다. 요구사항들은 개별적인 '속성'과 '중요한 요소'를 기술하고, 세부적인 내용들을 상세하게 요구사항으로 기술하도록 한다. 여기서는 '보안', '인위적인 조작 불가', '온라인', '정합성'이라는 측면에서 세부적인 사항들을 정의하였다.

19.1. 요구사항1. 보안

✓ **Data 유출/불법변경/오용/수신부인 등은 보장**

1. 암호화
   - Data 교환 중 발생 할 수 있는 불법 유출에 대해서 안전해야 함.
   - 허가된 사용자 만이 Data의 내용을 볼 수 있어야 함.
2. 무결성 보장
   - Data가 불법 변경 되더라도 이를 확인 할 수 있어야 함.
   - Data 이동 시 원본과 동일한 내용임을 증명할 수 있어야 함.

3. 인증
    - Data 교환 시 Target 병원이 인가된 사용자 인지가 확인 가능해야 함
    - Data를 수신 받은 Target 병원에서는 그 data가 source target에서 생성된 data인지를 확인할 수 있어야 함
4. 부인불가
    - Data를 수신 받은 병원은 data의 수신 여부를 부인 할 수 없어야 함

19.2. 요구사항2. 인위적인 조작 불가

✓ **기존 system의 최소 변경으로 Data 교환 및 자동화**

1. 기존 시스템과의 통합
2. 기존 시스템의 최소 변경으로 data 교환이 가능한 구조여야 함.
3. 충분한 이식성과 확장성이 보장되어야 함.

2. 자동화
    - Data 교환 시 사람의 개입 없이 Data 교환이 이루어 져야 함.
3. 변경가능성
    - Data 교환 모델은 유연해야 하며 모듈화 되어 있어야 함.
    - 교환 모델의 일부를 대체 시스템과 쉽게 교체 가능해야 하며 interface의 변경이 가능해야 함.
4. 사용성
    - 모델과 구현된 시스템은 사용자가 쉽게 이해하고 사용할 수 있는 수준이어야 함.
    - 시스템의 교환 모델의 이해가 용이해야 하며 단순해야 함. (모델 구현 측면)

19.3. 요구사항3. 온라인

✓ **불안전한 Public 망(Internet망) 상에서도 요구 성능과 가용성이 보장 되는가?**

1. 성능
2. 불안한 NW 망에서도 Data의 유실/변경/손실 등에 대비 되어야 함.
3. Data 교환이 지연될 경우, 예상 시간을 확인 할 수 있어야 함

2. 가용성

- 시스템적으로 충분한 가용성이 보장되는 모델이어야 함.

3. 신뢰성
   - Data 교환 시 발생 할 수 있는 오류에 대해서 정의 되어 있어야 함.
   - 자체 오류 허용 범위에 대해서 명시 할 수 있어야 하며, 충분한 자체 회복이 가능한 모델이어야 함

19.4. 요구사항4. 정합성

✓ **Data 교환 시 환자 ID 교환이 가능하며, Data 이동 시 추적성을 보장하는가?**

1. 환자 ID 호환
2. Source 병원의 환자 ID가 Target 병원의 환자 ID로 자동 변환 되어야 함.
3. 다른 말로, 병원/환자ID Mapping Model이 있어야 함
4. 추가로, 병원간 권한 관리 모델도 필요 함.

2. 영상 이동 시 추적성 보장
   - Data의 이동 흐름이 추적 되어야 함.

20. 아키텍처 목표

이러한 '문제'정의와 '요구사항'에 대한 정의를 통하여, 중요한 향후 시스템의 아키텍처의 목표에 대해서 수립한다. 이 내용들은 '문제'와 '요구사항'을 조합하여 하나의 문장으로 기술하는 것이 원칙이다.

리퍼런스 아키텍처의 주요한 아키텍처는 다음과 같이 정의한다.

병원과 병원간, 의료영상정보를 높은 보안체계하에서 인위적인 조작이 불가한 작업프로세스를 통해, 온라인으로 정합성이 보장된 영상정보를 전달하고자 한다.

이러한 주요한 아키텍처의 목표를 기술하고, 세부적으로 필요한 요소들에 대해서 정의한다.

1. **보안 - Data 유출/불법변경/오용/수신부인 등이 보장 되는가?**

1. 암호화
2. 무결성 보장
3. 인증
4. 부인불가

2. **인위적인 조작 불가 - 기존 system의 최소 변경으로 Data 교환 및 자동화가 가능 한가?**
    1. 기존 시스템과의 통합
    2. 자동화
    3. 변경가능성
    4. 사용성
3. **온라인 - 불안전한 Public 망(Internet망) 상에서도 요구 성능과 가용성이 보장 되는가?**
    1. 성능
    2. 가용성
    3. 신뢰성
4. **정합성 - Data 교환 시 환자 ID 교환이 가능하며, Data 이동의 추적성을 보장하는가?**
    1. 환자 ID 호환
    2. 영상 이동 시 추적성 보장

21. 아키텍처 전략

이렇게 '문제', '요구사항'을 통하여 아키텍처의 목표를 수립하고, 아키텍처를 수립하기 위한 전략은 어떻게 정의하느냐는 아키텍처의 '품질'요소를 어떻게 도출하느냐에 달려있다. 그러므로, 핵심 품질 속성들을 도출하여 서술하는 것이 원칙이다.

여기서는 다음과 같은 품질속성들을 도출하여 서술하였다.

21.1. 핵심 품질속성(Quality Attribute)

실제 품질속성을 도출하는 것은 '문제', '요구사항'을 통해서이다. 그리고, 이것에 대해서 우선순위를 정하는 것이다.

리퍼런스 아키텍처에서는 다음과 같은 품질속성들을 정의하였다.

- 문제 정의 > 요구사항 > 품질 속성 도출
- 아키텍트는 품질 속성의 중요도와 구현성을 통해 우선순위를 선정 함

기호	내용	관련 항목	중요도	구현성	합계
QA1	운영비용	T2	5	5	10
QA2	환자식별정보	D5	5	5	10
QA3	환자 인증	D3	5	4	9
QA4	유지보수 비용	T3	4	5	9
QA5	위/변조 방지	S7	4	5	9
QA6	영상 메타정보	D3	5	2	7
QA7	영상 이미지정보	D4	5	2	7
QA8	네트워크 장비	T5	4	3	7
QA9	개인식별정보	D6	3	4	7
QA10	데이터 보호	S5	5	1	6
QA11	전송구간 보호	S4	4	2	6

21.2. 아키텍처 품질목표 달성 전술(Tactics)

도출되어진 핵심품질속성을 어떻게 달성할 것인가에 대해서 전술적인 측면에 대해서 설계에 반영을 하도록 한다.

핵심 품질 속성 > 달성 방법 > 전술(Tactics) > 설계 반영

전술(Tactics)는

- 아키텍트가 품질 속성을 달성하기 위해 사용할 수 있음
- 하나 이상의 전술을 실현하기 위해 패턴을 선택
- 패턴은 여러 전술을 구현

전술의 예)

✓ 변경 용이성

→ 정보 은폐 / 중재자 / 바인딩 시간

✓ 수행 성능

→ 스케줄링 정책 / 자원 경쟁

✓ 보안

→ 사용자 인증 / 사용자 인가 데이터 기밀성 유지 일관성 유지 노출 제한 접근 제한

이러한 방법으로 세부적인 부분으로 정의하여 상세하게 설계에 반영한다.

21.3. 아키텍처 접근법

이러한 아키텍처의 접근방법은 많은 효과를 볼 수 있으며, 시스템의 아키텍처를 오랫동안 지속할 수 있게 하여준다. 의료정보를 다루는 영역에서도 동일한 효과를 볼 수 있으며, 세부적인 접근 방법을 따르면, 많은 효과를 볼 수 있다.

이러한 아키텍처 접근방법의 장점은 다음과 같다.

- 한번에 모든 것을 바라볼 수 없음.
- 아키텍처는 이해당사자간의 의사소통 수단이고 초기 설계의 기초 자료 임.
- 이해 당사자들의 Viewpoint를 이해하고 그에 맞는 뷰(View)를 작성
- 고려 사항
    1) 구현 단위들이 어떤 식으로 구조화 되어 있는가?
    2) 행위 또는 상호작용하는 요소들이 어떤 식으로 구조화 되어 있는가?
    3) 주변 환경에 존재하는 요소들과의 어떤 식으로 연결되어 있는가?

세부적인 뷰타입과 뷰스타일을 결정하고, 뷰를 작성하는 단계로 진행이 되어진다.

21.4. 아키텍처 스타일

의료영역에서의 소프트웨어 아키텍처도 동일한 구성을 가진다.

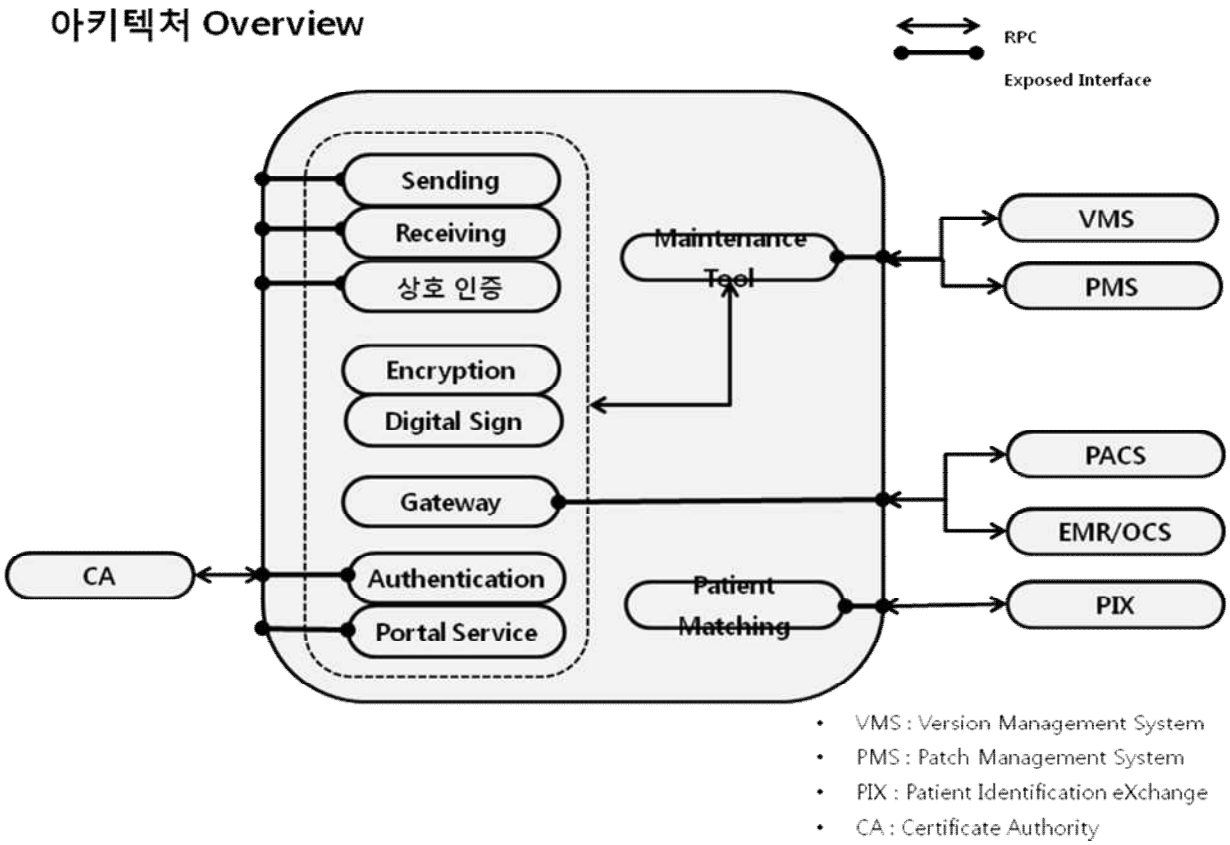
	아키텍처 스타일	관계	도움이 되는 영역
모듈뷰	분할	~의 하위모듈인, ~와 비밀을 공유하는	자원 할당과 프로젝트 구조화 및 계획, 정보 은폐, 캡슐화, 형상 통제
	사용	~의 정확한 존재를 요구하는	엔지니어링 하위집합, 엔지니어링 확장
	레이어	~ 정확한 존재를 요구하는, ~의 서비스를 사용하는, ~에 대한 추상화를 제공하는	"가상 기계" 이식성 위에 점증적인 개발을 통해 시스템 구현
	클래스	~의 인스턴스인, ~의 접근 메쏘드를 공유하는	공통 템플릿으로부터 신속하게 구현하는 객체지향 설계 시스템
C&C	클라이언트-서버	~와 통신하는, ~에 의존하는	분산 운영, 관심사의 분리, 수행성능 분석, 부하 균형
	프로세스	~와 동시에 실행되는 ~와 동시에 실행될 수도 있는 ~을 배제 또는 선행하는	스케줄링 분석, 수행성능 분석
	동시성	동일한 논리 쓰레드에서 실행되는	자원 다툼이 있는 곳, 쓰레드가 분기, 병합, 생성, 삭제 되는 곳을 식별
	공유 데이터	데이터를 생산하는, 데이터를 소비하는	수행성능; 데이터 통합; 변경용이성
할당뷰	배치	~에 할당하는, ~로 이전하는	수행성능, 가용성, 보안 분석
	구현	~에 저장되는	형상 통제, 통합, 테스트 활동
	작업 배정	~에 작업을 배정하는	프로젝트 관리, 전문 기술 사용, 공통성 관리

그림 15 아키텍처 스타일

각각의 아키텍처 스타일과 관계를 통하여 모듈뷰, C&C, 할당뷰를 정의할 수 있다.

22. 시스템 컨텍스트

의료도메인에서 정의되어진 시스템 컨텍스트를 도식화하였다.



이 시스템 컨텍스트 또한 한눈에 전체 아키텍처를 참조할 수 있게 한다.

22.1. 구성요소

도식화된 시스템 컨텍스트를 참조하여, 세부적인 구성요소들을 도출하고 정의한다. 리퍼런스 아키텍처에서는 다음과 같이 정의할 수 있다.

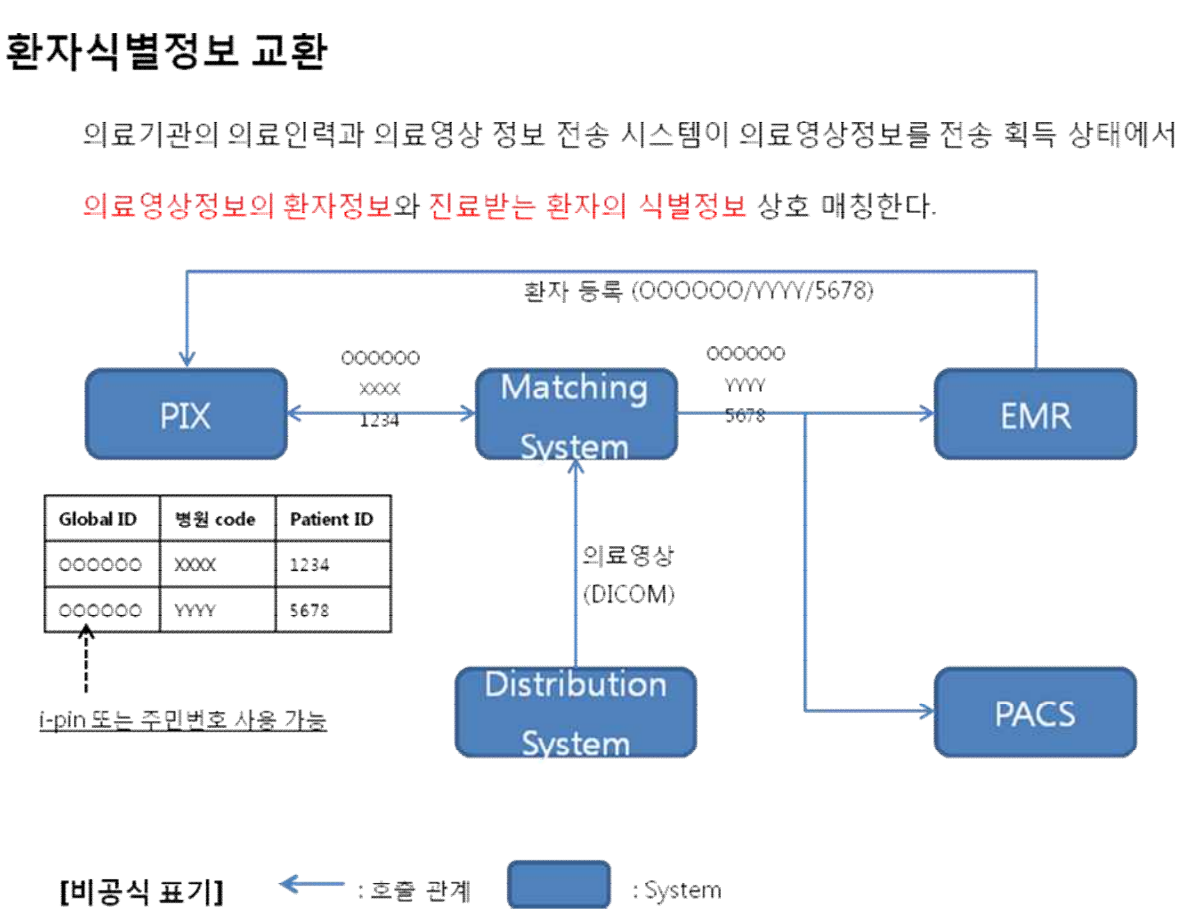
System	SubSystem	설명
Distribution	Sending	의료영상 정보를 target 병원으로 전송하는 system

System	Receiving	의료영상 정보를 source 병원으로부터 전송 받는 system
	상호 인증	의료영상 전송 시 유효한 병원인지를 상호 확인하는 system (양방향 인증)
Security System	Encryption	암/복호화를 수행하는 system
	Digital Sign	부인방지 및 불법적인 변경을 감지하기 위한 system
	Authentication	환자/의료인이 정당한 사용자 인지를 확인 하는 system
Interface System	Gateway	병원 내 system과 정보 교환을 담당하는 system
	Portal Service	의료영상 전송 요청/취소/이력을 조회 할 수 있는 system
Exchange System	Patient Matching	Patient 정보를 상호 교환/matching 을 수행하는 system
	PIX	병원 별 유효한 patient ID로 교환해주는 system
Maintenance	Maintenance Tool	System을 관리/모니터링 하기 위한 System
	VMS	Version Management System
	PMS	Patch Management System

23. 주요아키텍처 뷰

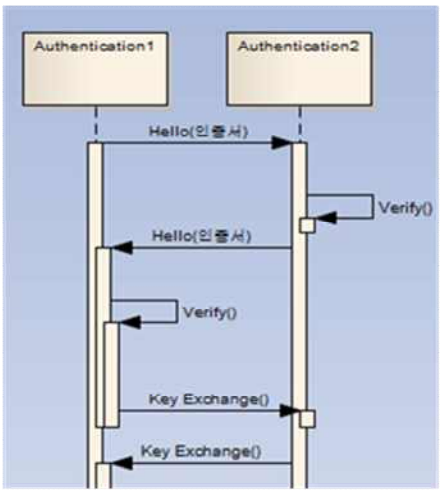
리퍼런스 아키텍처에서 정의된 주요 아키텍처 뷰에 대해서 설명한다. 정의된 뷰에는 '환자식별정보 교환', '의료기관 인증', '전송구간보호', '유지보수 비용'의 측면에서 주요 아키텍처 뷰를 정의하였다.

23.1. 환자식별정보 교환



24. 의료기관 인증

의료영상정보 전송 시스템이 의료영상정보를 전송 준비 상태에서 영상정보 이동 상태에 따라 영상정보를 전송할 의료기관의 상호인증을 수행한다.



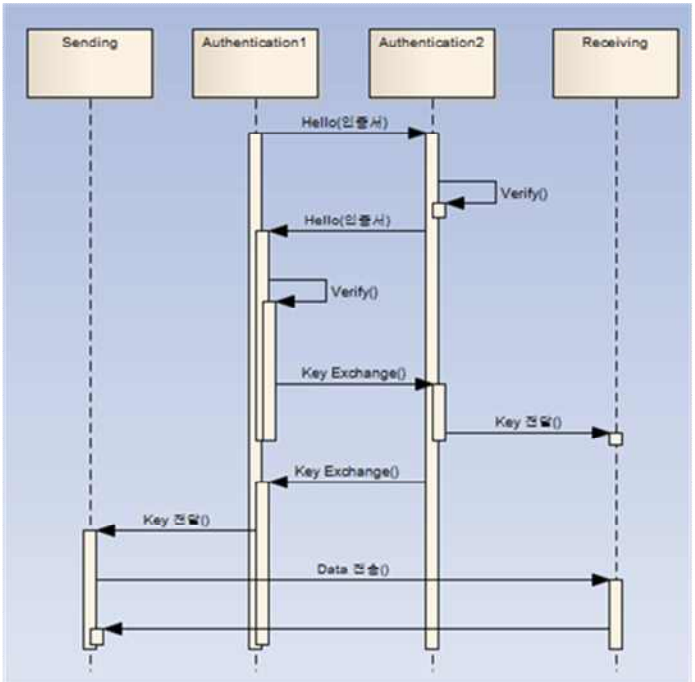
- 의료 기관의 공인인증서(Public key)를 전송
- 신뢰 할 수 있는 CA(Certificate Authority)에서 인증서의 유효성 확인 (Verify)

[Option – security 통신 준비]

- 의료기관의 session key를 랜덤으로 생성
- session key를 상대방 의료기관의 Public key로 암호화

24.1. 전송구간 보호

의료영상정보 전송 시스템이 의료영상정보를 전송 준비 또는 전송 상태에서 영상정보 전송구간 정보를 암호화하고 전송주체 간의 상호 인증을 수행한다.



① 인증서 교환(Public Key)

② 인증서의 유무 확인 (Verify)

③ Session Key를 랜덤으로 생성

④ 상대방의 public key로 session key 암호화

⑤ 상호 Key 교환

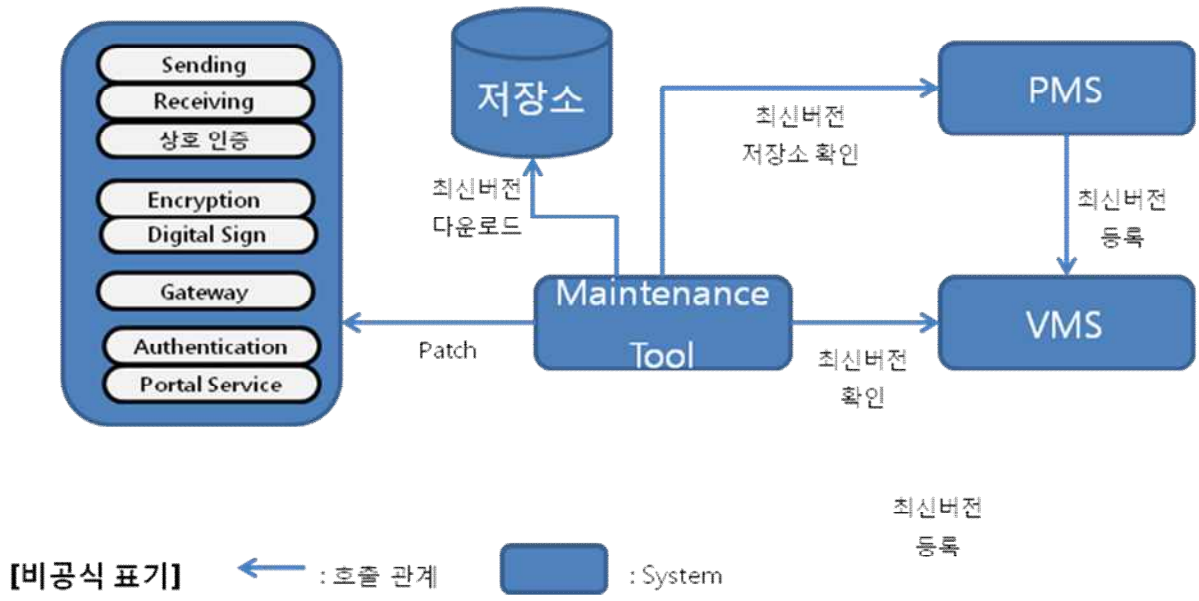
-------- Handshake ----------

① 교환된 Key로 Data 교환

② Data 전송 완료 시 자동 key 삭제

24.2. 유지보수 비용

의료영상정보 전송 시스템이 시스템의 구동/종료, 대기상태 또는 영상정보의 전송 준비 상태 또는 시스템의 사용자 인증 상태에서 의료영상정보 전송 시스템의 최신버전 유무를 자동 확인하여 시스템의 업데이트를 수행한다.



25. 아키텍처 검증

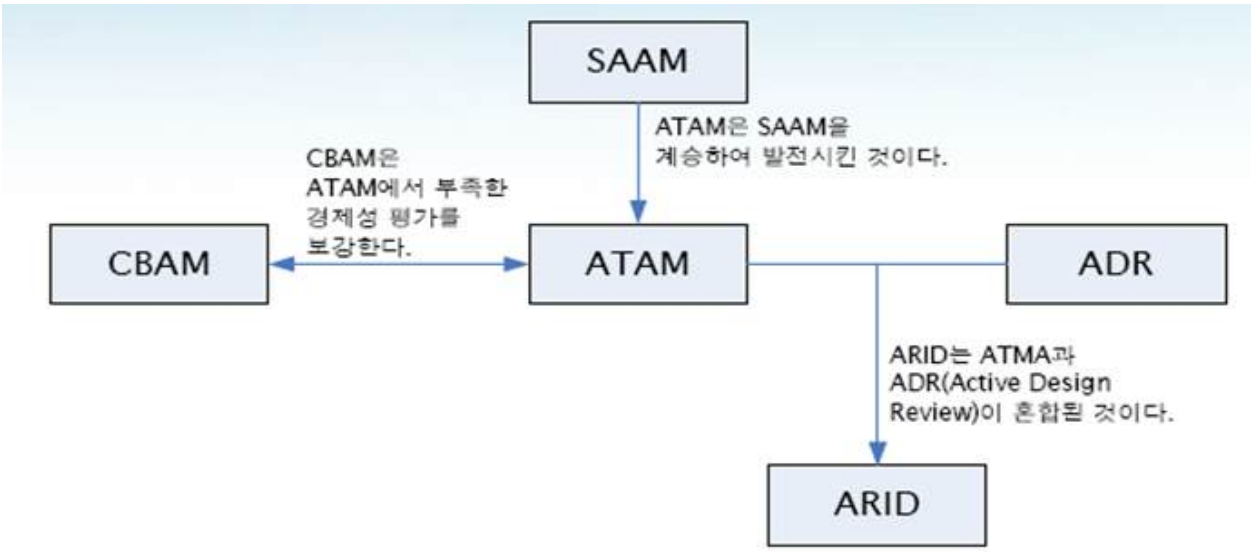
의료정보시스템의 아키텍처가 정의되고 이를 검증할 수 있는 방법들에 대해서 정의한다. 의료영역의 경우에는 의료서비스의 검증방법이 선행되어지고, 별도로 진행되어지는 영역들이 있기 때문에, 의료시스템의 영역에는 ATAM의 형태의 접근법이 좀더 일반적이다.

이는 '시나리오'를 많이 사용하여, 의료서비스의 형태와 연계하여 검증하는 방법이 매우 효과적이기 때문이다.

25.1. 아키텍처 검증(평가)모델

아키텍처 평가는 시스템의 위험 요소를 찾아서 제거하는 방법으로써 다음과 같은 방법들이 있다.

SAAM	최초로 정리된 아키텍처 평가방법-수정가능성, 기능성
ATAM	시스템품질에 초점, 시나리오기반 아키텍처 평가방법-성능, 가용성, 보안, 수정용이성, 사용성 등
CBAM	ATAM에 비용 및 이익 등의 경제성을 고려, 이해관계자의 경험기반-아키텍처 접근법의 ROI계산
ARID	설계초기에 부분 아키텍처 평가-모듈, 컴포넌트



각각의 형태에 맞추어서 평가 방법을 선정하는 것이 좋으며, 리퍼런스 아키텍처에서는 기본적인 시나리오에 충실한 ATAM방법을 사용하도록 한다.

25.2. ATAM (Architecture Trade-off Analysis Method)

아키텍처 품질 속성 만족 판단, 서로 상충하면서 상호작용하는지(trade-off)까지 분석하는 아키텍처 분석 평가기법이다.

Phase	활동	목표 / 참여자	기간 (일)
0	협력과 준비	검증 방법 및 아키텍처 이해 1. ATAM의 소개 2. 비즈니스/아키텍처 목표 3. 작성된 아키텍처 소개 참여자 : 검증팀리더, 주요 결정권자	1
1	평가	아키텍처 분석 및 평가 1. 아키텍처 접근법 식별 2. 품질속성 유틸리티 트리 생성 3. 아키텍처 접근법 분석 참여자 : 검증팀, 프로젝트 결정권자	1
2	평가	평가 및 아키텍처 정제 1. 브레인스토밍, 우선순위 결정 2. 아키텍처 접근법 분석 참여자 : 검증팀, 프로젝트 결정권자, 이해관계자	0.5 (Phase 1 이후 1주일 후)
3	후속 조치	후속 조치 1. 결과 발표 참여자 : 검증팀, 평가 의뢰인	0.5

25.3. ATAM (Architecture Trade-off Analysis Method) 9단계 평가

단계	주제	내용
1	ATAM 소개	검증팀 리더가 ATAM의 단계와 산출물들에 대해 설명
2	비즈니스/아키텍처 목표	프로젝트 결정권자가 업무 관점에서 시스템 전반을 설명하고 업무 목표, 제약 사항등을 설명.
3	작성된 아키텍처 소개	아키텍트는 비즈니스 드라이버를 어떻게 반영하고 있는지를 중점으로 설명.
4	아키텍처 접근법 식별	아키텍처 접근법과 스타일을 이해하고, 이를 이용해 아키텍처를 분석하는 데 초점을 둠. 아키텍트는 명확한 아키텍처 패턴에 대한 설명과 사용된 패턴 접근법을 목록으로 정리하고 공개.
5	품질속성 유틸리티트리 생성	품질 속성 목표를 자세하게 기술함. 도출된 품질 인자로부터 정제를 거쳐 품질 속성을 도출하고 시나리오를 작성함. 아키텍트는 그 시나리오를 달성하려면 얼마나 어려울지(구현성)와 중요도를 기술함.
6	아키텍처 접근법 분석	우선순위가 높은 시나리오부터 조사하여 위험요소, 비위험요소, 민감성 지점, trade off point를 찾아내고 아키텍처 결정사항을 문서화함.
7	브레인스토밍, 우선순위 결정	이해관계자가 브레인스토밍을 통해 역할과 관련된 의미 있는 시나리오를 도출하고 전체 이해관계자 그룹의 투표를 통해 우선순위를 결정.
8	아키텍처 접근법 분석	조정된 우선순위를 가지고 아키텍처 접근법 분석 활동을 반복함.

9	후속 조치	종합적인 형태의 본 보고서

25.4. Utility Tree

기호	내용	중요도	구현성	합계
T2	병원 의료인력의 투입없이 의료영상정보를 전송 처리할 수 있어야 한다.	5	5	10
D5	전송받은 환자식별정보를 병원에서 매칭할 수 있어야 한다.	5	5	10
S3	의료영상정보의 식별된 환자정보와 진료받는 환자의 식별정보가 동일한지 확인할 수 있어야 한다.	5	4	9
T3	시스템의 유지보수(버전관리, 업데이트 등) 비용 최소화를 지원해야 한다.	4	5	9
S7	의료영상정보의 복호화를 검증할 수 있어야 한다.	4	5	9
D3	의료영상정보의 메타정보를 병원에서 매칭할 수 있어야 한다.	5	2	7
D4	전송받은 이미지 정보와 원본을 일치시켜야 한다.	5	2	7
T5	기존 장비로 대체 활용할 수 있어야 한다.	4	3	7
D6	전송받은 개인식별정보를 병원에서 매칭할 수 있어야 한다.	3	4	7
S5	공인된 방식으로 전송 데이터를 보호할 수 있도록 암호화해야 한다.	5	1	6
S4	전송 데이터는 네트워크 상의 End to End에서 보호되어야 한다.	4	2	6

25.5. 아키텍처 접근법 분석서 작성

4단계 : 아키텍처 접근법을 찾아 냄

5단계 : 우선순위가 정해진 상세 품질 요구사항이 만들어 짐 (Utility Tree)

6단계 : 아키텍처 접근법 분석을 통해 아키텍처 판단을 내림

7단계 : 이해관계자와의 브레인스토밍을 통해 우선 순위를 조정함

8단계 : 6단계 반복

아키텍처 판단 (Architectural Decision)	민감점 (Sensitivity)	절충점 (Trade-off)	위험 (Risk)	무위험 (Nonrisk)
Distribution System에서 Sending 모듈과 Receiving 모듈을 분리	S1	T1	R1	NR1
.Net Framework 사용	S2	T2		

이상과 같이 리퍼런스 아키텍처를 사용하여 정의하고 이를 통하여 검증까지 수행하는 방법을 서술하였다.

26. 실무아키텍트 포럼의 결과물에 대한 의견

초기의 원대한 계획과 다루고자 했던 문제들에 대해서 접근하여 보았으나, 시간적인 문제과 공간적인 문제등으로 인하여, 용두사미의 결과가 도출된 것에 대해서는 애석함을 금치못한다. 다만, 이러한 과정을 통하여, 대한민국의 의료정보시스템의 구축에 도움이 될 수 있는 실무아키텍트 포럼의

취지에 부합되는 결과를 얻기 위하여 해당 연구를 시작하였고, 그 목적에 맞도록 방향성을 설립하는 것에는 어느 정도 목적을 달성하였다고 내부결론을 내어본다.

향후, 이러한 관련자료를 바탕으로, 의료정보시스템의 디자인에 있어서 보다 효율적이고 참조할 만한 의료정보시스템의 아키텍처 디자인에 도움이 되는 자료가 만들어졌으면 하는 것이 이번 2012년 연구포럼의 결과라 하겠다.