사례연구: 기능확장성을 위한 소프트웨어 아키텍처 재설계

김정호, 강성원

한국정보통신대학교 공학부
대전광역시 유성구 문지로 119번지(문지동 103-6)
aplymath@icu.ac.kr, kangsw@icu.ac.kr

요약: 본 논문은 소프트웨어의 기능확장성을 높이기 위해 소프트웨어 아키텍처를 재설계한 실제 사례를 소개한다. 본 논문에서는 이미 구현된 소프트웨어 시스템에서 기능확장을 시도할 때 소요되는 노력과, 기능확장을 위해 소프트웨어 아키텍처를 재설계 하고 동일한 기능확장을 시도할 때 소요되는 노력을 비교하여 소프트웨어 아키텍처 재설계에 의해 기능확장성이 얼마나 많은 도움을 주는지 추정해 본다. 또한 이러한 실제 사례 조사를 통하여 계층화, 관심의 분리, 모듈화와 같이 기능확장을 위한 아키텍처 설계 방법이 얼마나 기능확장성을 높이는 지 확인할 수 있다.

핵심어: 소프트웨어 아키텍처, 기능확장성, 재설계

1. 서 론

소프트웨어 개발 이후의 유지보수 비용이 소프트웨어 생명주기 전체 비용의 90% 이상이라는 것은 널리 알려진 사실이다[1][2]. 이런 유지보수 비용이 발생하는 이유는 시스템 개발 시에는 미처 고려하지 못했던 변화가 시스템 개발 이후에 필요해졌기 때문이고, 이를 효과적으로 적용할 수 있는 아키텍처가 부족하였기 때문이다[4]. 최근의 소프트웨어 개발 프로젝트에서 고객이 내세우는 가장 큰 이슈는 급변하는 경영환경에 얼마나 빠르고 효과적으로 적응하는 소프트웨어 시스템을 개발하느냐 하는 것이다. 따라서, 유지보수 중에서도, 고객의 요구와 환경변화에 따라 빠른 시스템 적용이 가능한 기능확장성(Extensibility)이 매우 중요한 역할을 하게 된다.

이 논문에서는 사례를 통해 기능확장성을 높일 수 있는 소프트웨어 아키텍처 재설계 방법론을 제시한다.

2. 기능확장성 정의

품질 속성은 모호하게 정의되어 있기도 하고 어떤 경우에는 서로 중복된 의미로 설명되기도 한다[6]. '예를 들어 신뢰성', '안정성', '성능'과 관련된 정의는 일반적으로 잘 설명이 되어있는 반면, '유지보수성', '사용성'과 같은 용어는 모호하다[6]. 따라서 본 논문은 기능확장성에 대한 정의를 명확히 하기 위해 IEEE 및 ISO 표준화 단체에서 기능확장성과 관련된 품질 속성들의 정의를 표 1에 요약하였다.

표 1. 기능확장과 관련된 품질 속성 정의

품질 속성	설명
유연성 (Flexibility)	- Application이나 특정 환경을 사용할 때 시스템이나 컴포넌트를 원하는 방향대로 쉽게 수정할 수 있는 능력[7]
기능확장성 (Extensibility)	- 기존 기능을 확장하거나 새로운 기능 혹은 데이터를 추가하여 소프트웨어의 성능이나 능력을 확장시키는데 투입되는 상대적인 노력 [7]
유지보수성 (Maintainability)	- 특정 장비에서 유지보수를 쉽고 빠르게 수행할 수 있는 능력. [7] - 소프트웨어 개발이 완료된 이후에 성능 혹은 다른 소프트웨어 품질 속성을 높이기 위해서거나 수정된 환경에 적응하기 위해 혹은 잘못된 부분을 수정하기 위해 들어가는 노력 [9]
진화성 (Evolvability)	- 고객 입장에서 소프트웨어 제품이 지속적으로 비용 대비 효율이 높은 쪽으로 진화하는 능력 [10] - 환경이나 요구사항, 혹은 구현 기술의 변화에 쉽게 대처할 수 있는 시스템의 능력 [11]
수정용이성 (Modifiability)	- 시스템의 변화를 빠른 시간 내에 적은 비용으로 고칠 수 있는 능력 [8]

표 1의 내용을 참조하여 본 논문에서는 기능확장성을 다음의 정의를 채택한다: *"기능확장성은 기존 기능을 확장하거나 새로운 기능 혹은 데이터를 추가하여 소프트웨어의 성능이나 능력을 확장시키는데 투입되는 상대적인 노력 [7]."* 이 정의에서 보듯이 본 논문은 소프트웨어 아키텍처에 새로운 서비스가 추가될 때의 상황에 초점을 맞춘다. 즉, 본 논문은 소프트웨어 아키텍처의 모듈뷰를 기준으로 모듈 내부의 변화와 관련된 기능확장성은 제외하고 새로운 모

듈이 추가되는 형태의 기능확장성 만을 고려한다[3]. 또한 ATAM에서 제시하는 소프트웨어 아키텍처와 관련된 Tradeoff와 관련된 설명은 제외한다[5].

3. 재설계의 실제 사례

이 절에서는 기능확장성을 높이기 위해 기존의 소프트웨어 아키텍처를 재설계한 사례를 소개한다. 우선 각 사례에 해당하는 소프트웨어 시스템의 개요를 설명한다. 두 번째로 실제 사례에서 제시된 기능확장성의 품질 시나리오를 소개한다. 세 번째로는 기존의 소프트웨어 아키텍처(재설계 대상)를 수정한 새로운 소프트웨어 아키텍처(재설계 결과)를 소개한다. 아키텍처를 수정한 이유와 수정하는 방법에 대해서도 간략히 언급한다. 마지막으로, 수정된 소프트웨어 아키텍처로 개발된 소프트웨어 시스템이 품질 시나리오를 만족하는지 평가한다. 사례 소개의 두 번째 단계에서 품질 시나리오를 제시하는 이유는 기능확장을 위해 추가되는 서비스를 기술하기 위해서이며, 추후 마지막 단계에서 소프트웨어 시스템 개발 이후에 새롭게 도입된 소프트웨어 아키텍처가 처음 목표했던 품질 시나리오를 얼마나 만족하는지 확인하기 위해서이다. 이런 품질 시나리오는 QAW(Quality Attribute Workshop) 에서 제시한 방법대로 작성되었다[12].

3.1 LBS 시스템

첫 번째 사례로 '위치 기반 서비스 플랫폼 개발 프로젝트를 소개한다. 본 사례는 기존에 제공되고 있는 LBS 서비스를 통합하여 하나의 플랫폼에서 작동하게 하고 신규 LBS 서비스 추가를 용이하게 하는 소프트웨어 시스템 이다.

3.1.1 개요

LBS 시스템은 사용자가 원하는 다양한 위치기반 서비스(LBS, Location Based Services)를 무선 네트워크 사업자가 쉽게 제공할 수 있도록 개발된 오픈 플랫폼이다. 위치기반 서비스는 사용자의 위치 정보를 위도, 경도와 같은 값으로 변환하여 각종 편의를 사용자에게 제공하는 서비스를 말한다. LBS 시스템은 네트워크 사업자가 신규 위치기반 서비스를 제공하려 할 때 저비용으로 보다 빠르게 서비스를 개발할 수 있도록 도와준다. 상위 수준의 소프트웨어 아키텍처는 그림 1 과 같다.

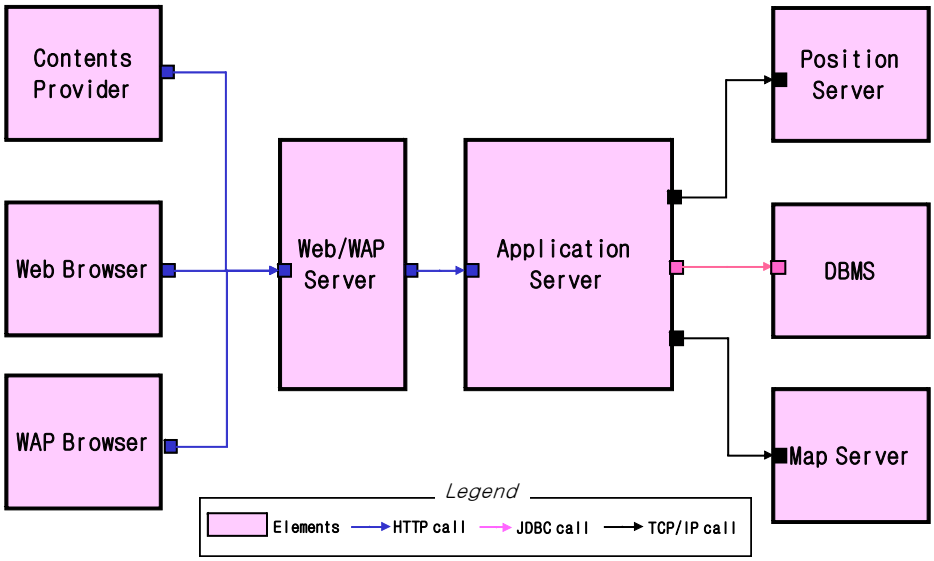

그림 1. LBS 시스템의 상위 수준 소프트웨어 아키텍처

LBS 시스템은 일반적인 웹 사용자뿐 아니라 모바일 폰 사용자를 위한 왑(WAP: Wireless Application Protocol) 사용자에게도 서비스가 가능하다. 또한 CP(Contents Provider)에게 위치 정보를 제공하기 위한 기능을 가지고 있다. 이와 같이 다양한 사용자와 연동을 지원하기 위한 웹, 왑 서버를 채널 역할로 활용한다. 또한 사용자 관리, billing, privacy 관리를 위한 자체 DBMS 와 위치 정보를 가져오기 위한 Position server 와 연동하고 다양한 지도 서버와 연동하여 위치 정보에 지도 정보를 맵핑하여 제공한다.

3.1.2 기능확장성 시나리오

기존의 LBS 시스템은 위치기반 서비스를 담을 수 있는 플랫폼의 형태가 아니고 단위 Application 서비스를 개발하여 독자적으로 서비스를 하는 시스템이었다. 따라서 새로운 위치기반 서비스를 개발하여 서비스하려고 할 때 많은 비용이 추가로 소요되었다. 고객은 신규 위치기반서비스의 개발비용을 줄이고 개발 기간도 단축할 수 있는 시스템을 요구하였다. 이러한 요구를 해결하고자, 기존 시스템의 아키텍처를 기능확장성을 높일 수

있는 LBS 시스템의 소프트웨어 아키텍처로 재설계 했다. 이와 관련된 품질 속성 시나리오는 QAW 에 기반하여 표 2 와 같이 작성되었다[12].

표 2. LBS 시스템의 기능확장성 품질 시나리오

Scenario	LBS 시스템을 이용하여 신규 위치기반 서비스를 개발할 때 서비스 개발자가 LBS 로직만을 구현함으로써 사용자가 3개월 이내에 위치기반 서비스를 사용할 수 있어야 한다.	
Scenario Refinement	Stimulus	신규 위치기반 서비스
	Source	신규 위치기반 서비스 개발자
	Environment	신규 위치기반 서비스를 사용자에게 서비스 할 때
	Artifact	LBS 시스템
	Response	신규 위치기반 서비스를 구현하는 기간
	Response Measure	3개월 이내

3.1.3 소프트웨어 아키텍처 재설계

기존의 LBS 시스템은 위치기반 서비스의 고유의 특징과 LBS 의 공통적인 특징 모두를 하나의 Application 에 가지고 있었다. 따라서 새로운 서비스를 추가하려고 할 때 LBS 시스템은 서비스 고유의 로직 뿐 아니라 위치기반 서비스의 공통적인 기능도 같이 개발해야 했다.

기능확장성을 고려한 재설계에서는 LBS 시스템의 아키텍처를 계층 패턴을 이용하여 위치기반 서비스 계층(service layer)과 서비스 공통 기능 계층(Facilities layer)으로 구분하였다.

서비스 계층은 위치기반 서비스 고유의 로직만 가지고 있고 서비스와 관련된 공통적 기능은 모두 서비스 공통 기능 계층에 두었다. 이는 다시 말해 새로운 위치기반 서비스를 LBS 시스템에 제공하려 할 때 신규 서비스 로직만을 구현하고 서비스의 공통 기능은 호출하여 재사용 하기만 하면 새로운 서비스를 제공할 수 있다는 것이다.

기존 LBS 시스템의 소프트웨어 아키텍처와 신규 위치기반 서비스의 추가를 쉽게 하기 위해, 즉 LBS 시스템의 기능확장성을 높이기 위해서 새롭게 설계된 소프트웨어 아키텍처는 그림 2 와 같다

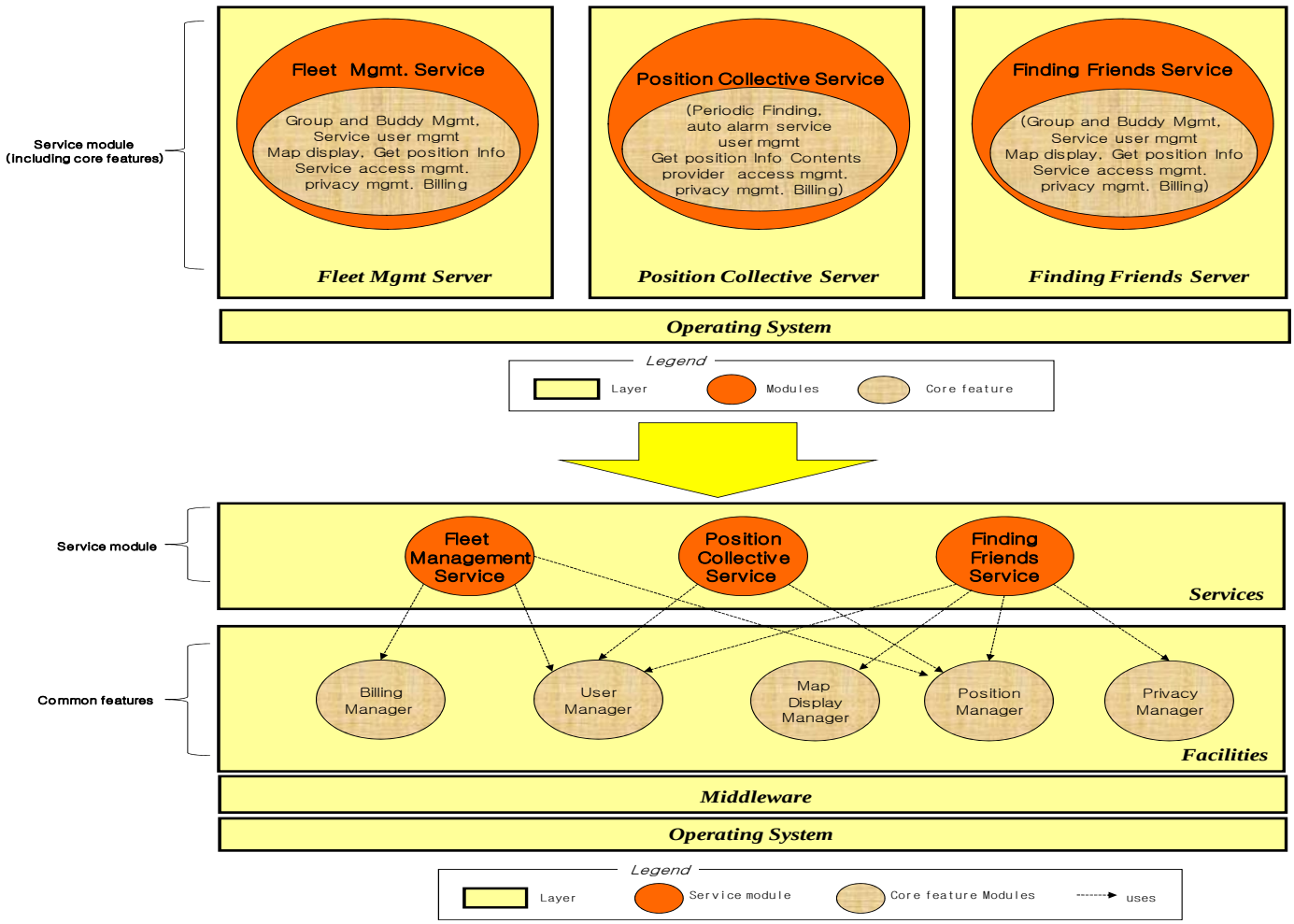

그림 2. 기능확장성을 고려한 계층 소프트웨어 아키텍처

그림 2 의 위쪽 그림은 기존의 LBS 시스템의 소프트웨어 아키텍처에서 위치기반 서비스가 존재하는 모습이다. 그림 2 의 아래 그림은 기존의 서비스는 그대로 유지하였으나 서비스가 가지고 있는 서비스 공통 기능을 Billing, User, Privacy, Position, Map display 으로 구분하여 하위 계층의 모듈로 구분하였다. 서비스 계층에는 위치기반 서비스의 고유 기능, 예를 들어 Fleet management 에서 주기적인 위치 수집, 그룹 사용자 관리 등은 서비스 계층에 서비스 고유 기능을 두어 서비스 계층의 Fleet management 모듈에 두었다. 이들 서비스 계층의 서비스 모듈은 서비스 공통 기능 계층의 모듈을 재사용할 수 있어 새로운 위치기반 서비스 개발이 편리하고 빠르게 진행될 수 있게 한다.

3.1.4 소프트웨어 아키텍처 평가

그림 3 은 LBS 시스템 개발 이후 기존의 LBS 소프트웨어 아키텍처에 신규 서비스(모듈)이 추가된 모습을 보여준다.

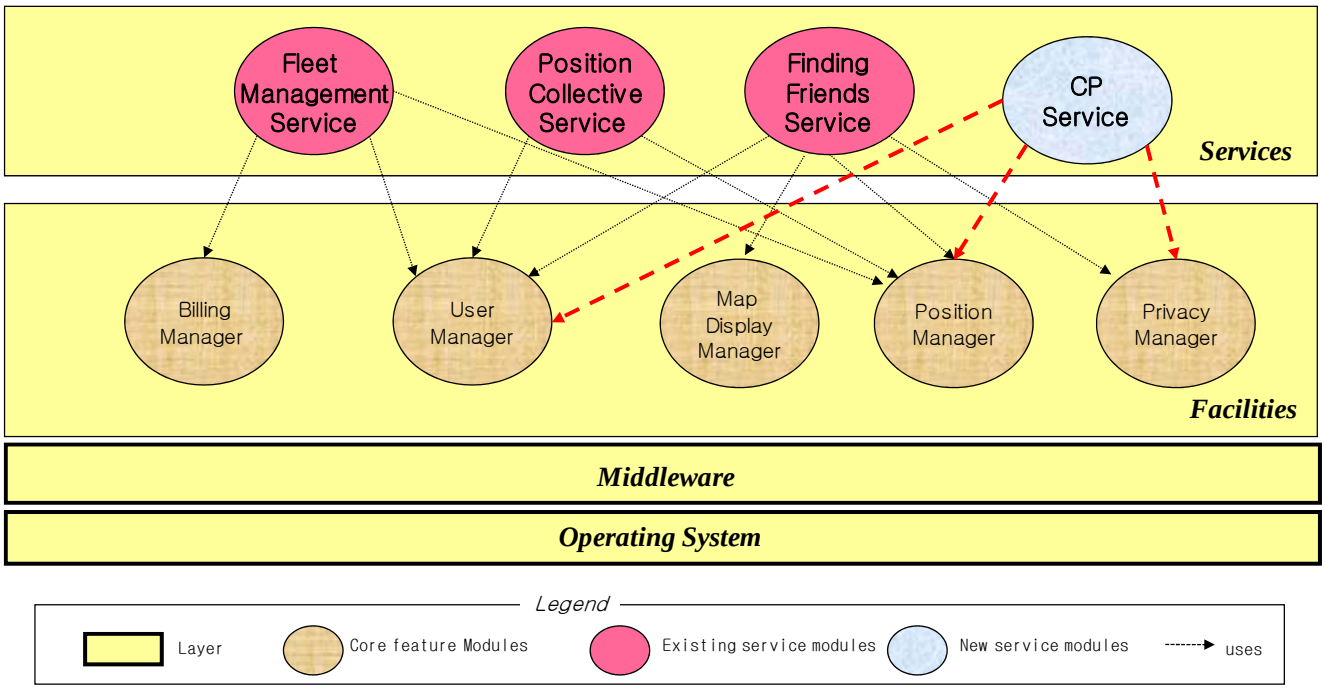

그림 3. 신규 서비스가 추가된 LBS 시스템의 소프트웨어 아키텍처

LBS 시스템 위에 개발된 서비스는 외부의 콘텐츠 제공자(CP, contents provider)를 인증하기 위해 가입자 관리 모듈에게 인증을 요청하여 인증 확인을 진행한다. 인증 확인 후 콘텐츠 가입자의 위치 정보를 제공할 수 있는 지 여부를 확인하기 위해 privacy 관리 모듈에게 privacy 체크를 의뢰한다. Privacy 체크가 성공하면 위치 관리 모듈에게 서비스 요청자의 위치를 제공받아 contents provider 에게 제공한다. 이를 입증하기 위해 실제 서비스인 외부 CP 연동 서비스를 직접 LBS Platform 에 개발하여 porting 해보았다. 사용하는 대부분의 feature 들은 facility 계층에 존재하는 관련 모듈에게 요청하고 CP 연동 서비스는 이들을 호출하고 판단하기만 하면 되었다. 외부 CP 연동 서비스는 단지 CP 에게 전달하는 서비스 로직만 구현하면 되었다.

수정된 소프트웨어 아키텍처를 기반으로 LBS 시스템을 개발하고 신규 위치기반 서비스를 추가한 결과 개발 기간이 4 개월 정도 단축되었다.

이는 서비스 계층과 서비스 공통 기능 계층을 구분하여 서비스 계층의 서비스 모듈이 facility 계층의 모듈, 즉 위치 관리 모듈, 가입자 관리 모듈, privacy 관리 모듈 등을 재사용하기 때문에 그 전 보다 쉽고 빠르게 신규 서비스를 추가할 수 있었기 때문이다.

위의 실험 결과에서 보듯이 신규 서비스로 추가를 용이하게 하기 위해 재설계된 LBS 시스템의 소프트웨어 아키텍처는 그 기능확장성을 용이하게 하는데 효과가 있었음을 알 수 있었다.

3.2 전사 경영정보 시스템

다음 사례는 전사 경영정보시스템에서 기능확장성을 고려해 수립한 소프트웨어 아키텍처 사례이다. 본 사례는 SOA 구현 전 단계의 소프트웨어 아키텍처를 구성하여 추후 SOA 시스템으로 전사 경영정보 시스템이 신속하게 확장할 수 있도록 만든 실제 소프트웨어 시스템 개발 사례이다.

3.2.1 개요

본 시스템은 전사 경영정보 시스템 구축을 통해 전사 핵심 프로세스를 효과적으로 지원해서 업무 효율성을 극대화하는 것을 목표로 한다. 따라서 경영정보를 실시간으로 제공하고 이를 통해 신속하고 정확한 의사결정을 내릴 수 있도록 지원한다. 본 시스템은 SOA 를 적용하기 위한 전 단계의 시스템으로 SOA 로 쉽게 넘어갈 수 있는 아키텍처와 설계 구조를 가져야 한다. 그림 4 는 전사 경영정보 시스템을 표현한 상위 수준의 소프트웨어 아키텍처이다.

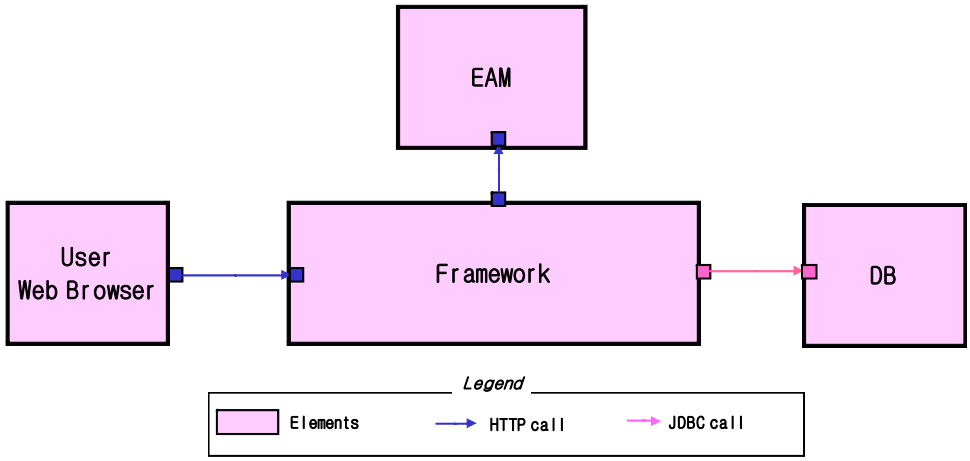

그림 4. 전사 경영정보 시스템의 상위 수준의 소프트웨어 아키텍처

그림 4 는 전사 경영정보 시스템의 상위 수준의 소프트웨어 아키텍처로써 사용자의 웹 브라우저를 이용하여 시스템의 초기 화면을 요청하고 Framework 과 EAM(전사 인증/권한 관리 시스템, Enterprise Access Management)을 통한 전사 인증 및 권한을 요청하여 인증 및 권한 처리를 한다. 인증이 완료되면 Framework 이 사용자가 요청한 비즈니스 로직을 전사 DB(Data Base)와 연동하여 완료한 후 사용자의 웹 브라우저에 전달하는 구조이다.

3.2.2 기능확장성 시나리오

표 3 의 품질 시나리오는 고객이 요청한 SOA 확장과 관련된 것을 QAW(Quality Attribute Workshop)에 6 가지 구조로 분석한 것이다[12].

3.2.3 기능확장성을 위한 소프트웨어 아키텍처

SOA 시스템으로 쉽게 확장하려면, Framework 요소에 SOA 와 관련된 기능이 추가되어야 하는 것이 핵심이다. 따라서 Framework 내부 요소가 SOA 표준에 맞도록 세분화된 계층 구조를 가져야 한다. 또한 SOA 가 지원하는 표준 Interaction interface 를 쉽게 받을 수 있어야 한다

그림 5 의 위쪽 그림은 기존에 존재한 Framework 요소 내부에 대한 아키텍처이다. 이 구조를 보면 컨트롤 로직과 클라이언트 연동 로직이 구분되지 않음을 볼 수 있다. 또한 Business service 계층 또한 비즈니스 기능과 Application 기능이 혼합된 모듈로 되어 있다. 반면에 그림 5 의 아래 그림은 왼쪽 그림의 Active service 계층을 클라이언트와 연동하는 Interaction service 계층과 비즈니스의 로직을 혼합하는 orchestration 계층으로 구분하였다. 또한 서비스와 비즈니스의 기능이 혼재된 Business service 계층을 비즈니스 기능을 가지고 있는 비즈니스 계층과 Application 기능을 가지고 있는 Application 계층으로 구분하였다.

표 3. 전사경영정보시스템의 기능확장성 품질 시나리오

Scenario	시스템 유지보수자가 SOA 시스템으로 전사 경영정보 시스템을 확장하려 할 때, 기존 시스템은 변동 없이 서비스를 진행할 수 있어야 하며 SOA 시스템은 3개월 이내에 적용될 수 있어야 한다.	
Scenario Refinement	Stimulus	SOA 시스템으로 확장
	Source	시스템 유지보수자가
	Environment	기존 시스템의 변동 없이 SOA 시스템으로 확장하려고 할 때
	Artifact	전사 경영정보 시스템
	Response	SOA 시스템으로 확장되는 기간
	Response Measure	3개월 이내

3.2.4 소프트웨어 아키텍처 평가

그림 5 의 Framework 내부를 Orchestration, Business service, Application service 계층 패턴으로 새롭게 구현하게 되면 SOA 로 가기 위한 Framework 내부의

기능은 이미 구현이 되었고 새로운 SOA interaction module 만 추가하면 된다. 다만 현재 서비스는 내부 고객을 위한 것이어서 내부 클라이언트 모듈만 연결되어 있다. 하지만 추후 웹 서비스 등으로 Enterprise MIS 시스템의 기능이 제공되어야 할 경우 그림 6 과 같이 Interaction service 계층에 Web service 와 관련된 연동 모듈만 추가하면 바로 외부에 service 를 제공할 수 있는 SOA 시스템으로 확장이 가능하다.

현재 이 전사 MIS 시스템은 수정된 소프트웨어 아키텍처로 개발이 진행되고 있다. 실제 개발이 완료되더라도 SOA 를 적용하기에는 시간이 더 필요할 것이다. 따라서 실제 SOA 연동이 품질 시나리오를 만족하는지 확인할 수는 없지만 소프트웨어 아키텍처 수준에서 판단하기에 충분한 효과를 보일 것으로 예상한다.

4. 결론 및 향후 연구방향

본 논문에서는 기능확장성을 높이기 위해 기존 소프트웨어 아키텍처를 수정하여 재설계한 사례를 소개하였다. 이 사례를 통해 기능확장성을 고려한 소프트웨어 아키텍처 재설계에 사용된 설계패턴이 아키텍처의 기능확장성을 향상시킨다는 것을 확인할 수 있었다. 즉 Parnas가 [Parnas 79]에서 제시한 계층 구조, 관심의 분리("separation of concerns"), 모듈화 와 같은 설계 패턴이 기능확장성을 높인다는 것을 실제 사례를 통해 확인하였다[13].

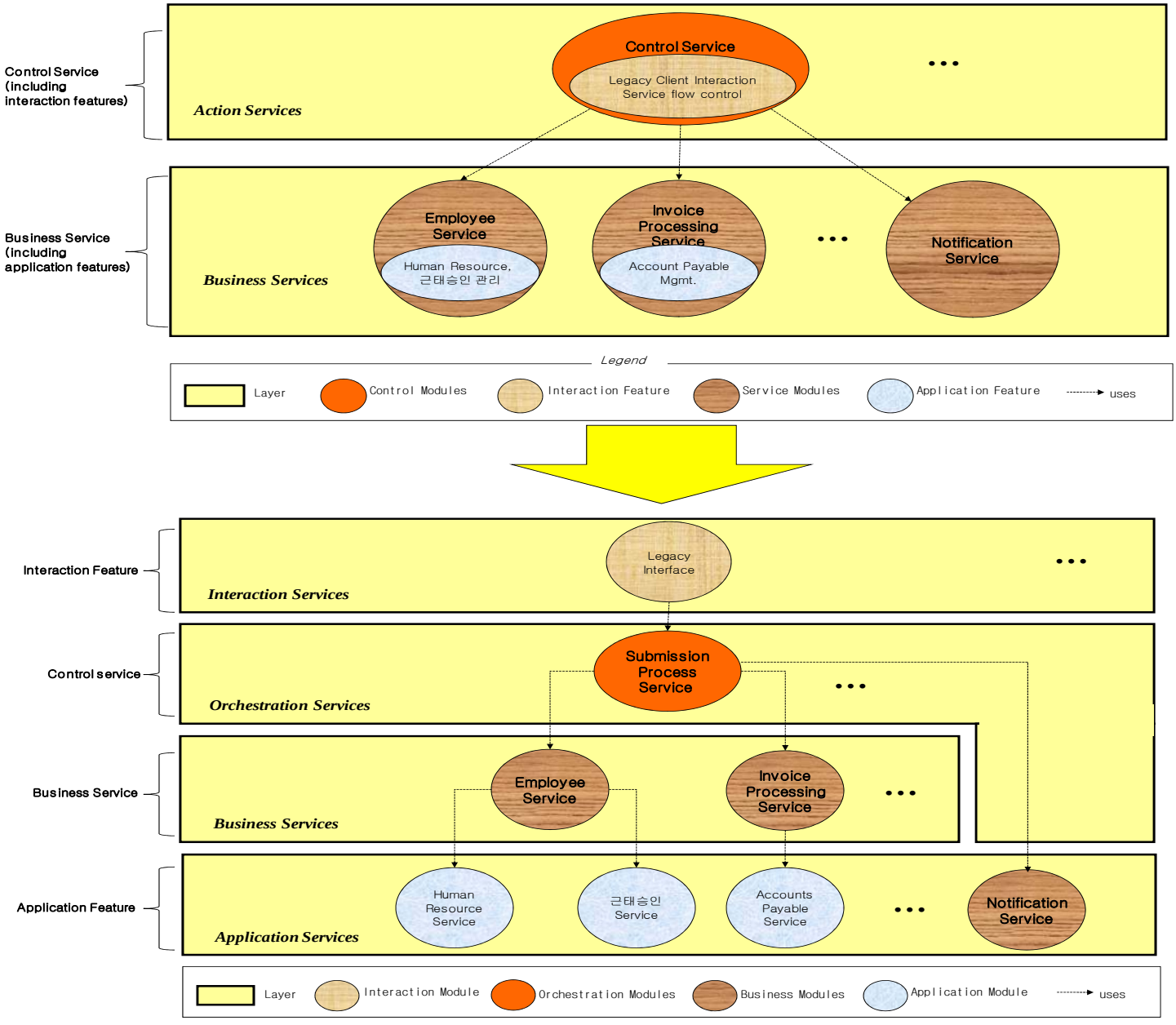

그림 5. Framework 내부의 소프트웨어 아키텍처

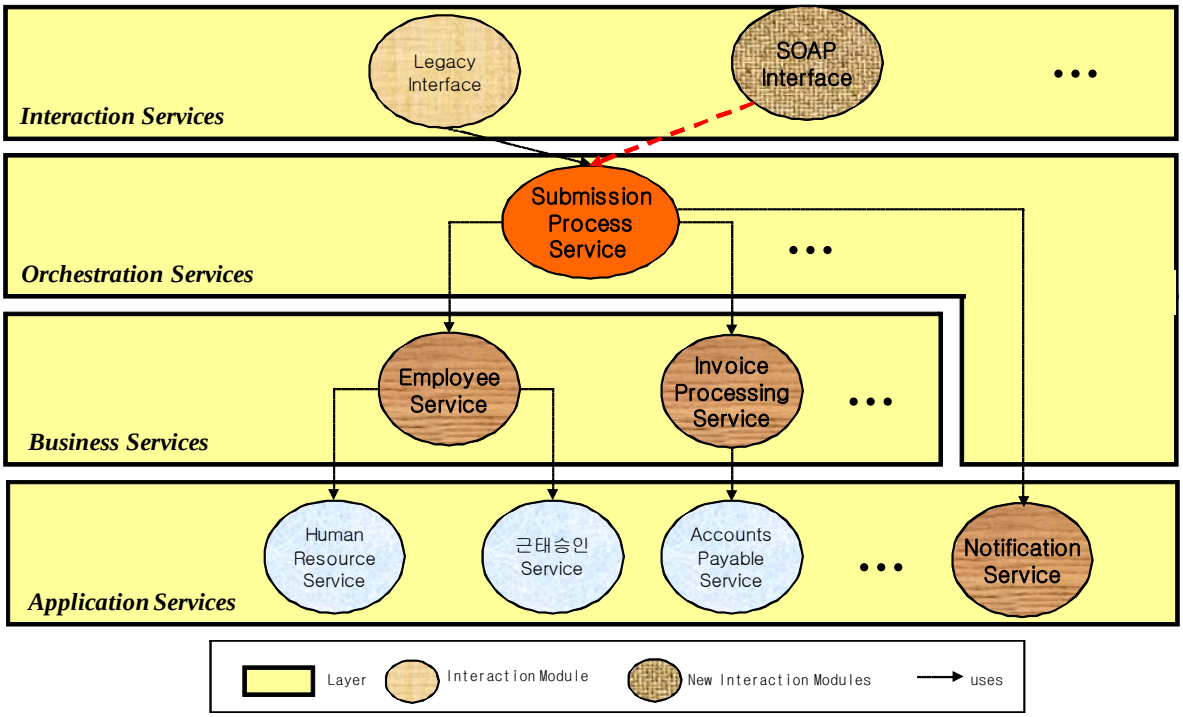

그림 6. SOA 시스템으로 전이하기 위해 SOAP 프로토콜을 지원하도록 변경된 Framework 의 아키텍처

향후 연구에서는 기능확장을 위해 본 논문에서 사용되었던 아키텍처 계층화와 application의 특징 분류 기법 이외에 일반적으로 제시되는 아키텍처 재구성 기법을 개발하겠다. 또한 소프트웨어 아키텍처 수준에서 기능확장성을 측정할 수 있는 메트릭(Metric)을 제시하여 기능확장성 평가 부분을 보강하고자 한다.

참고문헌

[1]. Smith, T.F., Waterman, M.S., "Identification of Common Molecular Subsequences", J. Mol. Biol. 147, pp. 195-197, 1980.

[2]. Lientz, Swanson, "Impact of development productivity aids on application system maintenance" ACM SIGMIS, Volume 11, Issue 3(Winter-Spring 1980), pp. 114-120, 1980.

[3]. Bass, L. Clements, P. Kazman, R., *Software Architecture in Practice*, 2nd Edition, Addison Wesley Longman, 2003.

[4]. Nosek, J. Palvia, P., " Software maintenance management: changes in the last decade" , Journal of Software Maintenance: Research and Practice 2 (3), pp. 157-174, 1988.

[5]. R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere, "The Architecture Tradeoff Analysis Method", Proceedings of ICECCS'98, 8-1, 1998.

[6]. Leonard J. Bass, Mark Klein, Felix Bachmann, "Quality Attribute Design Primitives and the Attribute Driven Design Method", Lecture notes in Computer Science, Vol. 2290, pp. 169-186, 2001.

[7]. IEEE Std. 610.12, *Standard for Software maintenance*, IEEE Computer Society Press, 1990.

[8]. ISO/IEC 9126-1, *Software engineering - Product quality - Part 1: Quality model*, 2001.

[9]. IEEE. Std. 1219, *Standard for Software maintenance*, IEEE Computer Society Press, 1998.

[10]. S. ook, H. Ji and R. Harrison, "Software evolution and evolvability", Technical Report, University of Reading, 2000.

[11]. Selim Ciraci, Pim van den Broek, "Evolvability as a Quality Attribute of Software Architectures", The International ERCIM Workshop on Software Evolution 2006, 6--7, pp. 29—31, 2006.

[12]. M. R. Barbacci, *Quality Attribute Workshops (QAWs)*, Tech Report CMU/SEI-2003-TR-016, 2003

[13]. Parnas, D. L., "Designing software for ease of extension and contraction", IEEE Trans. Software. Eng. SE-5(2), 128, 1979.