서비스 재사용성 중심의 Software-as-a-Service 개발 프로세스

(A Service Reusability-Centric Process for Developing Software-as-a-Service)

이 정 우 † 라 현 정 † 김 수 동 ††
(Jung Woo Lee) (Hyun Jung La) (Soo Dong Kim)

요 약 클라우드 컴퓨팅은 하드웨어와 소프트웨어의 자원을 서비스 제공자 서버에 운영하고, 소비자가 인터넷을 통하여 필요한 서비스를 활용하는 재사용 기반의 컴퓨팅 방식이다. 클라우드 서비스의 한 형태인 SaaS(Software-as-a-Service)는 소프트웨어를 하나의 서비스로 설계 개발한 후, 다양한 소비자들이 재사용하도록 하는 컴퓨팅을 지향한다. 기존의 소프트웨어 어플리케이션은 특정한 하나의 조직을 대상으로 개발하였으나, SaaS는 서비스의 형태로 소프트웨어의 전체 기능을 필요로 하는 다양한 조직에 소속된 다양한 소비자들이 사용할 수 있게 개발된다. 이것은 소비자가 자신이 원하는 기능의 수행을 위해 직접 SaaS를 설정할 수 없게 하여 SaaS 개발에 재사용성이 더욱 강조되게 한다. 하지만 기존의 객체지향 개발 방법론, 컴포넌트 기반 개발 기법, SOA 개발 기법들은 전통적인 소프트웨어 어플리케이션과 다른 SaaS가 가지는 재사용성의 특징을 반영한 설계와 구현을 지원하는 지침과 장치가 거의 없다. 따라서 본 논문에서는 이러한 문제를 해결하기 위해 기존 재사용성의 정의를 확장하여 SaaS의 재사용성을 적용성, 적응성, 확장성의 부 특성으로 구분하고 이를 반영한 재사용성 중심의 개발 프로세스를 제시한다. SaaS 재사용성의 각 부 특성은 제시하는 개발 프로세스의 분석과 설계를 위한 각 활동에 직·간접적으로 반영되어 보다 효과적으로 SaaS 재사용성을 향상시킨다. 제시된 프로세스를 적용하면 보다 체계적이고 효과적으로 재사용성 중심의 SaaS 개발을 유도할 수 있다.

키워드 : 클라우드 컴퓨팅, Software-as-a-Service, SaaS, 재사용성, 개발 프로세스

Abstract Cloud Computing is emerged as an effective reuse paradigm, where service providers operate hardware and software and as a service, and service consumers invoke the service through Internet. Software-as-a-Service (SaaS) is a type of cloud services, where the whole software is designed as a service so that several consumers can reuse the SaaS. While tradition software applications are developed for a specific organization, SaaS is developed for multiple users in the various organizations. Hence, reusability is very essential characteristic of SaaS. Reusability is defined as a metric of how effective and efficient software functionalities can be used by various users. Reusability in SaaS is evaluated by considering three sub-characteristics; applicability, adaptability, and scalability. Since such a SaaS has considerable differences and characteristics from traditional software applications, conventional methods including object-oriented modeling, component-based development method, and service-oriented architecture (SOA) service development method would be limited in developing services which can fulfill these three sub-characteristics related to reusability as well as SaaS-intrinsic characteristics. Hence, there is a great demand for effective processes for

· 본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었습니다(UD060048AD).
† 학생회원 : 숭실대학교 컴퓨터학과
jwlee@otlab.ssu.ac.kr
hjla@otlab.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim777@gmail.com
논문접수 : 2010년 3월 24일
심사완료 : 2010년 5월 10일

정보과학회논문지: 소프트웨어 및 응용 제37권 제7호(2010.7)

developing SaaS cloud services. In this paper, we present a practical process for developing SaaS, which focuses on ensuring reusability. And by performing a case study with our proposed SaaS development process, we evaluate applicability of our proposed process and explain how the process is used in a real domain. Then, we compare our proposed process with others for verifying our study.Through the proposed process, cloud services with high quality can be more effectively developed.

Key words : Cloud Computing, Software-as-a-Service, SaaS, Reusability, Development Process

1. 서 론

클라우드 컴퓨팅(Cloud Computing, CC)은 하드웨어와 소프트웨어의 자원을 서비스 제공자 서버에 운영하고, 소비자가 인터넷을 통하여 필요한 서비스를 활용하는 재사용 기반의 컴퓨팅 방식이다[1]. 대표적인 클라우드 서비스의 유형으로 Software-as-a-Service(SaaS), Component-as-a-Service(CaaS), Platform-as-a-Service (PaaS) 및 Infrastructure-as-a-Service (IaaS)가 있다. CaaS는 서비스 지향 아키텍처(Service-Oriented Architecture, SOA)의 메쉬업(Mash-up) 서비스와 동일하고, PaaS는 미들웨어 및 운영체제의 서비스를 다루고 있으며, IaaS는 하드웨어 자원의 재사용을 제공한다. 소프트웨어 서비스로는 SaaS와 CaaS가 있는데, CaaS는 SOA 연구에서 그 개발 기법이 상당히 연구가 되었으나, SaaS의 개발 기법은 매우 미흡하고, 본 논문에서는 SaaS의 설계 기법을 다루고 있다.

SaaS는 소프트웨어를 하나의 서비스로 설계 개발한 후, 다양한 소비자들이 재사용하도록 하는 컴퓨팅을 지향한다[2]. 기존의 소프트웨어 어플리케이션은 특정한 하나의 조직을 대상으로 개발되였으나, SaaS는 서비스의 형태로 소프트웨어의 전체 기능을 필요로 하는 다양한 조직에 소속된 다양한 소비자들을 위해 개발된다. 이것은 소비자가 원하는 기능의 수행을 위해 직접 SaaS를 설정할 수 없게 하여 SaaS 개발에 재사용성이 더욱 강조되게 한다.

재사용성이란 소프트웨어 기능모듈 등 특정한 자원이 다양한 소비자들에게 얼마나 효율적이고 효과적으로 사용될 수 있는 가의 척도이다[3]. 이를 기반으로 SaaS에서의 재사용성을 정의하면 다음과 같다.

> 요구되는 소프트웨어 기능을 해당 SaaS가 정해진 서비스 품질(Quality of Service, QoS)을 만족하면서 얼마나 많고 다양한 소비자들에게 효과적으로 재사용될 수 있는가의 척도를 나타낸다.

이처럼 SaaS 재사용성을 광역의 의미로 정의하면, 세 가지 부 특성(Sub-Characteristic)에 큰 영향을 받는다. 적용성(Applicability)은 SaaS가 얼마나 많은 소비자들에게 적용될 수 있는가의 정도를 나타낸다. 즉, 그 서비스의 기능을 필요로 하는 소비자 그룹이 얼마나 큰 가의 척도이다. 적응성(Adaptability)은 소비자가 특정 기능을 사용할 때, 얼마나 효과적이고 효율적으로 사용할 수 있는가의 정도를 나타낸다. 확장성(Scalability)은 SaaS를 이용하는 동시 소비자 수의 급증과 이에 따른 소요 자원의 증가에도 성능, 반응시간 등 다양한 QoS가 일정 수준이상을 유지하며 서비스를 제공할 수 있는 정도이다.

이렇게 전통적인 소프트웨어 어플리케이션보다 확장된 재사용성이 요구되는 SaaS의 개발에는 SaaS 재사용성의 세 가지 부 특성이 개발 프로세스에 반영되어야 한다. 하지만 기존의 객체지향 개발 방법론, 컴포넌트 기반 개발 기법, SOA 개발 기법들은 SaaS 재사용성의 부 특성을 반영한 서비스의 설계 및 구현을 지원하는 지침과 장치가 거의 없어 적용에 한계를 가진다. 따라서, SaaS를 개발하기 위해서는 SaaS 재사용성의 부 특성이 고려된 재사용성 중심의 개발 프로세스와 세부 지침이 요구된다.

본 논문에서는 SaaS 개발을 위한 프로세스와 그 주요 활동에 대한 수행 지침을 제공하는 방법론의 프레임워크를 제시한다. 본 논문에서 제시하는 개발 프로세스는 SaaS 재사용성에 중심을 둔 것으로써, 정의된 SaaS 재사용성의 세 가지 부 특성인 적용성, 적응성, 확장성을 개발 프로세스의 각 주요 활동에 직·간접적으로 반영하여 보다 효과적으로 SaaS 재사용성을 향상시킨다. 이를 위하여, 3장에서는 우선 SaaS 재사용성에 영향을 미치는 세가지 부 특성을 상세히 정의하고, 4장에서는 SaaS의 주요 구성을 나타내는 메타모델을 제시한다. 5장에서는 이를 기반으로 SaaS 개발 프로세스와 지침을 정의한다. 6장에서는 제안된 개발 프로세스를 적용한 사례연구를 통하여, SaaS 재사용성의 각 부 특성이 프로세스에 반영되어 SaaS 개발에 적용됨을 보인다. 7장에서는 기존 방법론들과의 비교분석을 통해 제안된 SaaS 개발 프로세스에 대한 평가를 수행한다. 제시된 프로세스를 적용하면 보다 체계적이고 효과적으로 재사용성이 높은 SaaS를 개발을 유도할 수 있다.

2. 관련 연구

Espadas의 연구는 SaaS를 개발하기 위한 체계적인 개발 프로세스의 필요성을 제기하고 이를 제안한다[4].

이를 위해 먼저 기존 소프트웨어와 SaaS의 특징을 구분하고 그 특징을 기존 소프트웨어 방법론에서 제시한 요구사항, 분석, 설계, 구현, 테스팅의 5단계로 구성된 개발 프로세스에 반영한다. 요구사항에서는 다수의 소비자들을 고려하여 비즈니스 측면의 요구사항을 정의한다. 분석 단계에서는 비즈니스적 관점에서 다수의 소비자들을 만족시켜야 하기 때문에 비즈니스 절차를 분석하고 유즈케이스 다이어그램을 작성한다. 설계 단계에서는 기술연구, 기술평가, 서비스 지향 아키텍처, 비즈니스 절차 공학 등을 고려한 설계를 한다. 구현에서는 위의 사항들을 바탕으로 어플리케이션과 프레임워크 환경을 구현한다. 마지막으로 테스팅 단계에서는 유닛 테스트, 통합 테스트, 성능 테스트, 테넌트(tenant) 측정 테스트를 한다. 하지만 각 단계별로 목적과 간단한 지침만을 제시하고 있어서 SaaS 개발을 위해서는 실제 개발에 필요한 각 단계별 구체적 활동과 보다 상세한 개발 지침을 필요로 한다.

Kwok의 연구는 다중 테넌트(multi-tenant)를 지원하는 전자 계약 관리 SaaS 어플리케이션의 아키텍처 프레임워크를 제시한다[5]. SaaS 성숙도 모델을 테넌트, 인스턴트, 코드의 구성 형태에 따라 네 가지 수준으로 구분한다. 이 중 다수의 테넌트들을 위한 변경 가능한 메타 데이터를 가지는 하나의 인스턴스와 코드로 구성된 수준 3에 기초하여 SaaS 아키텍처 프레임워크를 제시한다. 이 프레임워크는 웹 서버, 어플리케이션 서버, DB2 서버로 구성되며 각 서버에는 전자 계약 관리 서비스를 제공하는데 필요한 여러 서비스 모듈들이 설치되어 있다. 특히, 각 테넌트들에게 특화된 서비스를 제공하기 위해 여덟개의 모듈을 각 서버에 배치하였다. 뿐만 아니라 다중 테넌트를 위한 시스템 보안과 데이터베이스 관리 기법도 제시한다. SaaS를 이용하는 각 테넌트들에게 특화된 서비스를 제공하기 위한 실천적인 아키텍처 프레임워크를 제시하였지만 SaaS 아키텍처 설계에 다중 테넌트를 위한 확장성이 고려되지 않아 이에 대한 추가적인 연구가 요구된다.

Mietzner의 연구는 다중 테넌트를 지원하는 SaaS 어플리케이션이 다수의 소비자들과 제공자들이 만족할 만한 서비스가 되기 위해서는 두 가지 목표가 달성되어야 한다고 주장한다[6]. 첫째는 테넌트 종속적인 적응과 설치를 제공함으로써 잠재적인 테넌트들의 다양한 요구사항을 만족해야 한다는 것이다. 두번째는 각 테넌트들이 가지고 있는 공통성을 SaaS 어플리케이션에 잘 반영하여 경제적인 이익을 크게 하는 것이다. 이를 위해서는 효과적인 가변성 모델링 기법이 필요한데 이 연구에서는 기존 프로덕트 라인 공학(Product Line Engineering, PLE)의 외부와 내부 가변성 개념을 적용한 가변성 모델링 방법으로 SaaS 어플리케이션의 가변성을 모델링 한다. 그러나 이 연구는 SaaS 어플리케이션 내의 가변성 모델링 기법만을 다루고 있기 때문에 SaaS 개발을 위한 측면에서 개발 프로세스를 구성하는 각 활동들과 지침들과의 관계가 언급되지 않아 SaaS를 개발하기 위해서는 추가적인 연구가 필요하다.

3. SaaS의 재사용성

재사용성은 SaaS의 주요한 특성 중 하나로, SaaS 개발 프로세스는 재사용성을 보장할 수 있는 지침 및 기법을 제시해야 한다. 본 장에서는 그림 1에서 보는 것처럼 SaaS의 재사용성에 직접적으로 영향을 미치는 부 특성인 적용성, 적응성, 확장성에 대하여 상세히 설명한다.

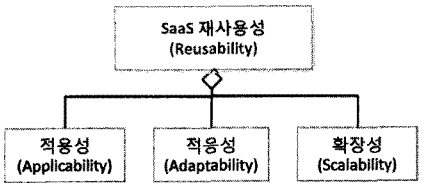


그림 1 SaaS 재사용성의 세 가지 부 특성

3.1 적용성(Applicability)

SaaS가 얼마나 많은 소비자들에게 적용될 수 있는가의 정도를 나타낸다. SaaS의 재사용성을 높이기 위해서는 다양한 요구사항들을 가능한 많이 SaaS에 반영하여야 한다. 그러므로, 적용성은 SaaS 재사용성의 주요 부특성 중의 하나로 정의될 수 있다. 적용성은 CBD에서 컴포넌트가 다수의 사용자가 원하는 공통적인 기능을 제공하도록 하는 것과 같은 의미를 가지기 때문에, 공통성(commonality)과 직접적인 연관이 있다[7]. 그림 2는 SaaS 재사용성의 적용성을 보여준다. 다수의 소비자들은 각자 자신의 시스템 상황에 따라 다른 요구사항을 가지며, SaaS는 여러 소비자가 원하는 공통 기능을 제공해야 한다.

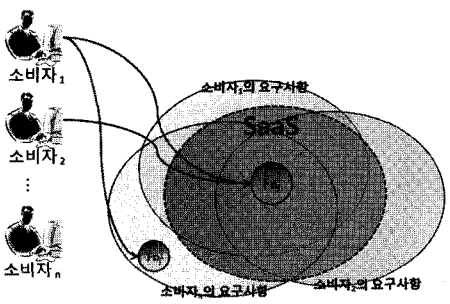


그림 2 SaaS에서의 적용성

만약 적용성이 충분히 고려되지 않아 그림 2에서의 Fn_j 같은 기능이 SaaS에 포함된다면, 적은 수의 소비자만이 SaaS를 사용할 수 있게 된다. 그로 인해, 개발된 SaaS의 재사용성은 떨어지게 된다. 반면에, SaaS 개발 프로세스에서 다수의 소비자들의 요구사항을 분석하여 공통적으로 원하는 Fn_j와 같은 기능을 SaaS에 포함한다면, SaaS는 가능한 많은 소비자들이 사용할 수 있게 되어 재사용성이 높아진다. 그러므로, 보다 많은 소비자들의 공통적인 요구사항을 반영하여 그들이 원하는 공통적인 서비스를 SaaS로 제공하여 적용성을 높여야 한다.

3.2 적응성(Adaptability)

적응성은 소비자가 특정 기능들을 사용할 때, 얼마나 효율적으로 특화하여 사용할 수 있는가의 정도를 나타낸다. 즉, SaaS가 부분적으로 소비자의 요구사항에 맞는다면, 이 서비스를 개인 요구사항에 맞게 특화해서 사용할 수 있는지를 나타낸다. 공통적인 기능을 제공하는 SaaS는 모든 소비자의 요구사항을 100% 만족하기는 어렵다. 그러므로, 더 많은 소비자가 원하는 기능을 완전히 만족하기 위하여, 개개인의 요구사항에 맞게 특화시킬 수 있는 높은 적응성이 요구된다. 이는 CBD에서의 가변성(Variability) 개념과 연관이 있다[8]. 그림 3은 SaaS 재사용성의 적응성을 보여준다. SaaS는 단위 서비스의 형태로 여러 소비자가 공통으로 요구하는 기능인 Fn_1, Fn_2, ... Fn_n을 제공하지만, 이들은 소비자를 완전히 만족시키 힘들다. 그러므로, SaaS는 소비자에 따라 부분적으로 수정이 되어, 조금씩 다른 형태로 제공된다.

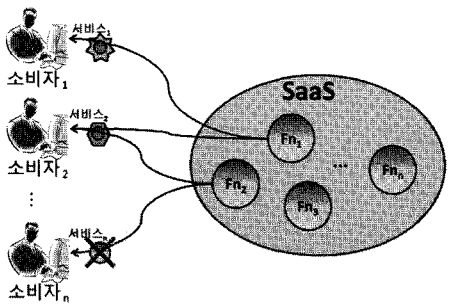


그림 3 SaaS에서의 적응성

Fn_1은 소비자에 따라 서비스$_1$과 서비스$_2$의 형태로 수정될 수 있으므로, 소비자를 완전히 만족시킬 수 있게 된다. 즉, Fn_j와 같이, 여러 소비자의 요구사항에 특화될 수 있도록 높은 적응성의 SaaS를 제공하면, 많은 소비자가 SaaS를 사용할 수 있어 재사용성을 높일 수 있다. 반면에, Fn_2는 소비자2를 만족시킬 수 있는 서비스$_2$ 형태로 제공되지만, 소비자$_n$을 만족시키지는 못한다. 즉, Fn_2와 같이 소비자가 원하는 서비스의 형태를 고려하지 않고 SaaS를 제공하게 되면, SaaS를 이용하는 소비자의 수는 제한적이게 된다. 이것은 SaaS의 이용을 저하시키고 결국 SaaS의 재사용성은 떨어지게 된다. 그러므로 많은 소비자들에게 만족할 수 있는 SaaS를 제공하기 위해서 SaaS 개발 프로세스에서 적응성이 반드시 고려되어야 한다.

3.3 확장성(Scalability)

확장성은 기존 시스템에서 원하는 목적을 위해 기능이나 데이터를 추가할 수 있는 능력의 정도를 나타낸다[9]. SaaS에서의 확장성은 많은 소비자들이 동시에 해당 서비스를 사용할 때 소비자들에게 영향을 미치지 않고 얼마나 제공할 수 있는가로 정의한다. 그림 4는 SaaS 재사용성의 확장성을 보여준다. 동시 소비자의 수의 급증과 이에 따른 소요 자원의 증가로 성능, 반응시간 등 SaaS의 여러 QoS가 낮아질 수 있다. 그림에서와 같이 SaaS의 Fn_1은 현재 n명의 소비자에게 수용할 만한 QoS로 제공되었지만 추가적인 소비자인 소비자$_{n+1}$이 동일 Fn_1을 요청하자마자 전체적인 QoS가 떨어지게 된다면, 소비자$_{n+1}$ 뿐 아니라 기존 소비자도 SaaS을 만족하지 못하고 사용하지 않을 수 있다.

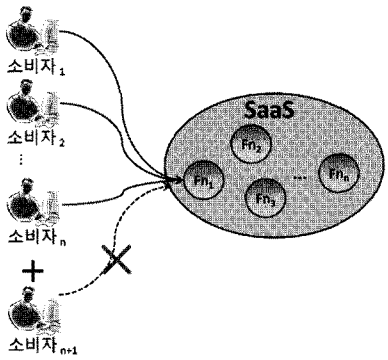


그림 4 SaaS에서의 확장성

만약 SaaS 개발 프로세스에서 확장성이 고려되지 않으면, SaaS가 허용할 수 있는 소비자가 제한적이게 되며, 이는 곧 SaaS 재사용성에 악영향을 미치게 된다. 즉, SaaS에 대한 소비자의 신뢰를 떨어뜨리게 되어 SaaS를 보다 덜 사용하게 함으로써 결국 SaaS의 재사용성을 떨어뜨리게 된다. 그러므로, SaaS의 재사용성을 높이기 위하여, SaaS 개발 프로세스는 높은 확장성을 보장할 수 있는 SaaS를 개발할 수 있는 지침을 제공해야 한다.

4. SaaS 메타모델

클라우드 서비스의 형태 중에서 SaaS는 가장 널리

보급되었지만, 그 구성요소와 이들 간의 연관 관계에 대한 표준이나 공감이 크게 결여된 상태이다. 본 장에서는 SaaS 개발 프로세스 정의에 근간이 되는 SaaS의 주요 구성 요소를 식별하고, 이들 간의 관계를 명확히 정의함으로써, 5장에서의 개발 프로세스를 보다 명확히 정의하고 이해할 수 있도록 한다. 그림 5에서 제시하는 SaaS의 메타모델은 관련된 문헌 및 표준을 조사 및 분석하여 작성된 것이다[2,10].

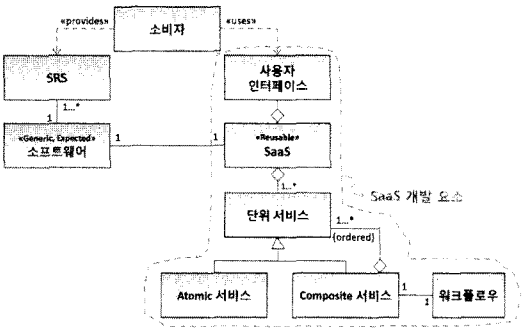


그림 5 SaaS 메타모델

SaaS는 소프트웨어의 기능을 제공는 동시에 서비스의 한 형태이기 때문에, 여러 소비자들이 필요로 하는 공통적인 기능을 제공하는 범용적(«generic») 소프트웨어가 SaaS로 제공된다. 그러므로, 그림 5와 같이 범용적인 소프트웨어와 SaaS는 1:1 관계로 정의된다.

범용적인 소프트웨어의 개발을 위해 하나 이상의 소프트웨어 요구사항(Software Requirement Specification, SRS)이 필요하며, 이는 다수의 소비자로부터 제공(«provides»)된다. 소비자는 각자가 원하는 기능 및 비기능 요구사항을 SRS에 명세하게 되고, SaaS 개발자는 이를 고려하여 SaaS를 개발하는 것이다. 그리하여, 소비자는 각자만의 요구사항 또는 기능을 만족하기 위하여 SaaS를 사용«uses»하게 된다.

SaaS는 소프트웨어에 준하는 기능을 제공하기 때문에, 응집도가 높고 결합도가 낮은 독립적으로 실행 가능한 기능 단위인 하나 이상의 단위 서비스(Unit Service)로 구성된다. 독립적인 기능을 제공한다는 의미 관점에서는 SOA 서비스와 비슷한 의미를 가지지만, SOA 서비스처럼 클라이언트 어플리케이션에서 호출되어 매쉬업 가능한 기능을 의미하는 것은 아니기 때문에 SOA 서비스와는 다른 개념으로 본 논문에서 사용된다. 각 단위 서비스는 더 이상 분할 되지 않는 단일(Atomic) 서비스와 다른 서비스들을 포함하면서 정의되는 복합(Composite) 서비스로 나누어진다. 복합 서비스는 여러 개의 단일 서비스를 이용하여 하나의 워크플로우를 나타내므로, 그림 5의 메타 모델에서는 재귀적 복합 구조를 이용하여 나타내었다. 복합 서비스는 비즈니스 프로세스와 같이 워크플로우를 포함하지만, 재사용 가능한 워크플로우를 제공하기 때문에 기존의 비즈니스 프로세스와는 다른 의미를 가진다.

SaaS는 SOA 서비스처럼 재사용 기능만 제공하는 것이 아니라 재사용 가능한 단위 서비스들을 이용할 수 있는 사용자 인터페이스(User Interface)를 같이 제공한다. 그리하여, 소비자는 사용자 인터페이스를 통하여 SaaS에서 제공되는 여러 단위 서비스를 사용할 수 있게 된다.

다음 장부터는 메타 모델 중 붉은 박스 안에 포함되는 소비자 인터페이스와 단위 서비스를 설계하는 프로세스 및 지침을 제공한다.

5. 프로세스 및 지침

본 장에서는 재사용성 중심의 SaaS를 개발하기 위한 설계 프로세스를 제안한다. 이 프로세스는 3장에서 언급한 적용성, 적응성, 확장성을 반영하여 각 단계 및 활동을 정의하였다. SaaS 설계 프로세스는 그림 6에서 보는 것처럼 서비스 요구사항 정의, SaaS 분석, SaaS 설계, SaaS 시스템 구현의 네 단계로 구성된다.

5.1 단계 100. 서비스 요구사항 정의

서비스 요구사항은 기존의 소프트웨어 요구사항과는 달리 여러 소비자들이 공통으로 필요로 하는 요구사항을 정의한 것이다. 이 단계에서는 SaaS 요구사항을 정의하기 위하여, 먼저 여러 소비자들로부터 소프트웨어 요구사항을 수집하고, 공통성 분석을 통해 공통 요구사항을 식별한다.

5.1.1 활동 110. 소프트웨어 요구사항 수집

이 활동에서는 SaaS 개발 도메인에 속하는 잠재적인 여러 소비자들로부터 SRS를 수집한다. SaaS는 특정 소비자가 아니라 잠재적으로 많은 소비자가 사용할 수 있도록 설계되어야 하므로, SaaS 설계에 여러 소비자들의 다른 요구사항을 반영하는 것은 반드시 필요하다. 그러나, 여러 소비자들로부터 직접적으로 요구사항을 받는 것은 쉽지가 않다. 이런 경우에는, 잠재적인 소비자 또는 도메인 전문가와의 인터뷰를 통해 요구사항을 작성해야 한다.

5.1.2 활동 120. 소프트웨어 요구사항 정규화

이 활동에서는 도메인에서 사용하는 표준 용어를 이용하여 수집된 SRS를 정규화한다. SRS 정규화는 각 소비자마다 사용하는 용어를 통일하고, SRS의 기능 단위를 동일하게 하기 위해서 수행한다.

다른 소비자로부터 수집된 요구사항은 동일한 의미를 뜻하는 용어를 다른 단어로 사용할 수 있고, 같은 단어

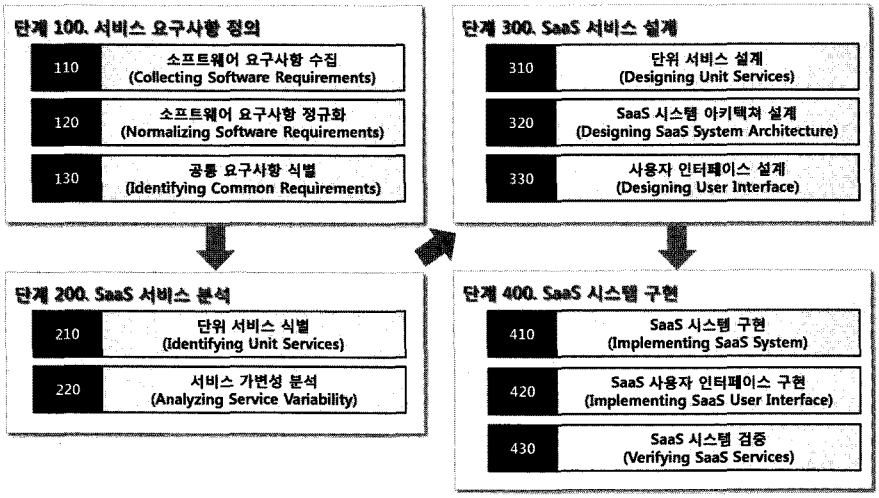


그림 6 SaaS의 설계 프로세스

표 1 용어 비교표

소비자 / 기능영역	소비자1	소비자2	…	소비자n	표준용어
기능영역$_1$	용어(1,1)	용어(2,1)	…	용어(n,2)	표준용어1
	:	:	…	:	:
기능영역$_n$	용어(1,j)	용어(2,k)	…	용어(n,l)	표준용어n

를 다른 의미로 사용하는 경우가 있다. 표 1을 이용하여 각 용어들을 분석하고 표준 용어를 정의한다[7].

각 소비자는 기능 또는 비 기능 요구사항을 다른 크기(Granularity)로 작성하면, 요구사항 비교 시에 어려움이 생기게 된다. 그러므로, 각 요구사항에 정의된 기능성 크기를 일관성있게 작성해야 한다. 용어와 기능 단위의 정규화 과정을 마친 후에는, 선택된 표준 용어와 기능 크기를 이용하여 이전 활동에서 수집한 요구사항을 재 작성한다.

5.1.3 활동 130. 공통 요구사항 식별

이 활동에서는 각각 정규화된 SRS 집합을 비교/분석하여 공통 요구사항을 식별한다. SaaS의 재사용성을 향상시키기 위해서는, 그림 2에서 제시된 높은 적용성이 보장되어야 한다. 그러기 위해 이 활동의 수행 과정에서 산출된 결과는 3.1장에서 제시한 적용성의 정의를 직접적으로 반영하여야 한다.

첫째, 각 SRS별로 기능성 분석을 실시한다. 유즈케이스와 비즈니스 프로세스 모두 소비자가 인지할 수 있는 기능을 분석하는데 사용되는 주요 기법이므로, 본 논문에서는 유즈케이스 모델링 또는 비즈니스 프로세스 모델링을 사용하는 것을 모두 허용한다. 비즈니스 프로세스 모델링을 수행할 때에는 [11]에서 제시된 유즈케이스 모델을 기반으로 비즈니스 프로세스를 분석하는 기법을 이용할 수 있다. 여기서 주의할 점은 각 요구사항 별로 모든 기능성에 대해 기능성 분석을 실시할 경우, 많은 시간과 노력이 소요된다는 점이다. 이전 단계에서 정규화 과정을 통하여, SaaS 시스템 분석가는 특정 기능성이 완전히 일치하는지는 파악이 가능하다. 그러므로, 약간의 차이점이 발생할 수 있는 기능성에 대해서만 선별적으로 기능성을 분석하여 보다 경제적으로 해당 활동을 수행하도록 한다.

둘째, 유즈케이스 모델 또는 비즈니스 프로세스 모델에 포함된 각 기능별로 얼마나 많은 소비자에게 요구되는지의 정도인 공통성을 분석하여 공통 요구사항을 식별한다. 표 2는 공통 기능 비교 분석표를 보여준다. '기능 이름' 열에는 유즈케이스 이름 또는 비즈니스 프로세스의 '액티비티' 이름을 기술하고, '기능 설명'열에는 해당 기능에 대한 간략한 설명을 기술한다. 그리고, 해당 기능을 소비자 n이 원하면 '소비자 집합' 열에 'O'표시를 한다. 마지막으로, '공통성' 열에는 전체 소비자 중 얼마나 많은 소비자가 해당 기능을 요구하는지를 백분율로 나타낸다.

공통성을 구하는 공식은 다음과 같으며, 0과 1사이의 값이 나온다.

$$\text{공통성} = \frac{\text{해당 기능을 요구하는 소비자 수}}{\text{전체 소비자 수}}$$

표 2 공통 기능 비교 분석표

기능 휘처 이름	기능 설명	소비자 집합				공통성
		소비자$_1$	소비자$_2$	…	소비자$_n$	
휘처$_1$						
…						

위의 공식으로 구해진 공통성 수치 중 1/n 이상의 값을 가지는 기능 휘처는 공통성을 어느 정도 가진 것으로 해석될 수 있다. 즉, 1/n 이상의 값을 가지는 휘처는 단위 서비스의 주요 후보가 되는 것이다.

셋째, 공통 요구사항들을 포함하는 SaaS 요구사항 명세서를 작성하도록 한다. SaaS 요구사항 명세서는 유즈케이스 명세서와 비슷하게 작성되며, 각 공통 휘처에 대한 기능 설명, 입/출력물, 사전/사후 조건, 간단한 기능 흐름을 포함한다.

5.2 단계 200. SaaS 분석

이 단계에서는 전 단계에서 식별한 공통 요구사항을 이용하여, 가변성 관점에서 SaaS를 분석한다. 먼저, 공통성 분석 결과를 이용하여 SaaS를 이루는 단위 서비스를 식별하고, 각 단위 서비스 별 가변성을 분석하여 적용성과 적응성이 높은 서비스를 개발하도록 한다. 이 단계를 통해서 SaaS 분석서를 만든다. SaaS 분석서는 단위 서비스 명세서와 가변성 분석서로 구성된다.

5.2.1 활동 210. 단위 서비스 식별

이 활동에서는 SaaS의 주요 구성 요소인 단위 서비스를 식별한다. 그림 2에서 보았듯이, SaaS에 포함되는 단위 서비스는 여러 소비자가 원하는 공통적인 기능을 제공하여 높은 적용성을 보장해야 SaaS의 재사용성도 증가하게 된다. 단위 서비스는 응집도가 높고 결합도가 낮은 독립적으로 실행 가능한 기능 단위이기 때문에, 재사용성뿐 아니라 응집도 역시 서비스 식별에 중요한 기준이 된다. 그러므로, 이 활동에서는 응집성 등의 기준을 사용하여 단위 서비스를 식별한다.

먼저, SaaS 요구사항 범위에 속할 기능 군들을 식별한다. SaaS는 고유한 특성상 SOA 서비스처럼 단위 기능이 아니라 소비자가 원하는 전체 기능을 제공해야 하므로, 기능 휘처 간의 기능/데이터 의존성을 기반으로 하는 다음의 가이드라인을 사용한다.

- 가이드라인 #1) 기능 휘처의 공통성이 0이거나 0에 가깝다면, 해당 휘처는 SaaS 개발 범위에 속할 가능성이 극히 적다. 다만 공통성이 높은 기능 휘처와 의존성이 높으면, 이 휘처가 SaaS 개발 범위에 속할 수 있다. 즉, 유즈케이스 다이어그램에서 공통성이 높은 유즈케이스와 include 또는 extend 관계를 가지는 유즈케이스는 SaaS 개발 범위에 속할 가능성이 높게 되는 것이다(붉은 색만 칠해진 영역, 즉 단위 서비스 영역에서 공통기능 군 영역을 제외한 부분).
- 가이드라인 #2) 기능 휘처의 공통성이 1이거나 1에 가깝다면, 해당 휘처는 SaaS 개발 범위에 속할 가능성이 높다(노란색 '공통기능 군' 영역).
- 가이드라인 #3) 기능 휘처의 공통성이 0과 1사이의 값을 가진다면, ROI, 시장성, 재정지원성 등의 도메인 또는 SaaS 제공자에 특화된 사항들을 고려하여 SaaS 개발 범위에 속할 수 있게' 된다.

그림 7은 가이드라인을 사용해서 결정된 SaaS 개발 범위를 보여주며, 이 가이드라인을 적용하여 표 3으로 작성함으로써, SaaS 개발 영역을 선정한다. 기타 기준으로 선택된 의존성이 높은 기능 휘처는 가이드라인 #1)에서 제시한 것처럼 유즈케이스 다이어그램에서 include와 extend 관계에 있는 유즈케이스의 기능휘처를 의미한다. 그 외의 기타 기준은 도메인과 SaaS 제공자에 특화된 사항들로써 개발하려는 SaaS마다 다르게 적용된다. 여기서 예로든 '시장성'은 해당 기능을 요구하는 소비자가 도메인 내에서 상대적으로 큰 비중을 차지하고 있어 그 기능 휘처를 개발 영역에 포함할 경우 SaaS의

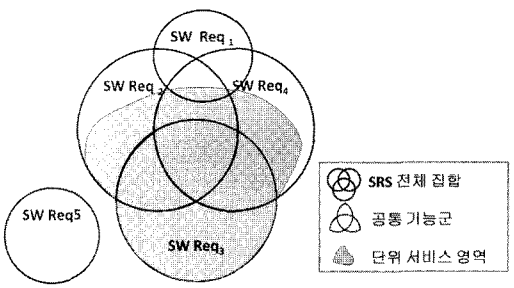


그림 7 SaaS 개발 범위

표 3 SaaS 개발 영역 결정표

기능 휘처 이름	공통성	기타 기준					SaaS 개발영역 (Y/N)	근거
		의존성이 높은 기능 휘처	시장성	…	…	C_n		
휘처$_1$								
…	…	…	…	…	…	…	…	…

표 4 단위 서비스 목록

단위 서비스 이름	관련 기능 휘처 이름	기능 설명
서비스$_1$	휘처$_1$, 휘처$_2$	….
…	…	…

시장성이 커짐을 체크하기 위한 것이다.

둘째, SaaS 개발 범위에 속한 기능 휘처를 대상으로 단위 서비스를 식별한다. 이 때에는 기능 의존성과 데이터 의존성을 고려하여, 결합도는 낮고 응집성이 높은 단위 서비스를 식별한다. 표 4는 단위 서비스 목록을 보여준다. '단위 서비스 이름' 열에는 식별된 서비스의 이름을 작성하고, '관련 기능 휘처 이름; 열에는 단위 서비스와 관련 있는 휘처 이름을 작성한다. 그리고, '기능 설명' 열에는 단위 서비스의 기능에 대해 간략히 기술한다.

셋째, 식별된 단위 서비스에 대한 명세서를 작성한다. 명세서에는 기능 개요, 입력 매개변수, 출력 매개변수, 사전/사후 조건 등을 포함한다.

5.2.2 활동 220. 서비스 가변성 분석

이 활동에서는 이전 활동에서 식별된 단위 서비스에 대하여 가변성을 분석한다. 현재까지 식별된 서비스는 적용성만을 고려하였기 때문에, 이 활동은 가변성 분석을 통해서 적응성이 높은 서비스가 개발되는 것을 보장해야 한다. 따라서 이 활동의 수행 과정에서 산출되는 결과는 3.2장에서 제시한 적응성의 정의를 직접적으로 반영하여야 한다.

SaaS는 소프트웨어를 서비스 형태로 제공하는 것이기 때문에, 기존의 소프트웨어 특징과 SOA 서비스 특징을 모두 가지게 된다[3,12]. 그러므로, CBD에서의 가변성 종류와 SOA에서의 가변성 종류 모두가 SaaS의 가변성 종류가 될 수 있다.

가변성 종류 집합 = {속성, 로직, 인터페이스,
워크플로우, 영속성, 컴포지션, QoS}

가변성을 분석하기 위해서, 먼저 공통 기능을 대상으로 발생 가능한 가변점 별로 가변치를 식별한다. 즉, 위에서 설명한 가변성 종류 중 어느 가변성이 발생하는지를 찾는다.

속성 가변성과 인터페이스 가변성은 유즈케이스 명세서 중 소비자 입력값을 분석하여 식별할 수 있으며, 로직 가변성과 워크플로우 가변성은 유즈케이스 명세서 중 기능 흐름(flow)을 분석하여 식별할 수 있다. 그림 8과 같이 기능 흐름 중 한 활동만 소비자마다 다르면 로직 가변성에 해당하지만, 기능 흐름의 전체 활동 개수가 다르면 워크플로우 가변성에 해당된다. 하나의 유즈케이스 A에 대하여 세 개의 가변 흐름을 분석하였다. <흐름 1>과 <흐름 2>는 활동 3에서만 가변성이 발생하므로, 로직 가변성이 발생한 것이다. 반면에, <흐름 2>와

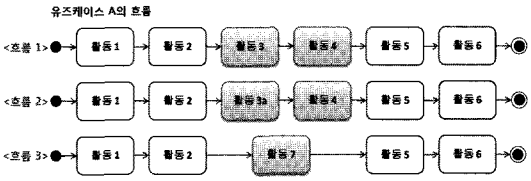


그림 8 로직 가변성과 워크플로우 가변성

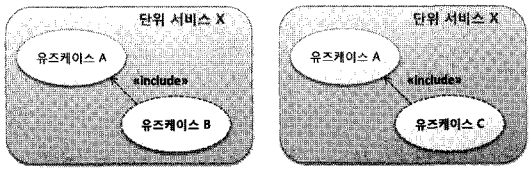


그림 9 컴포지션 가변성의 예제

<흐름 3> 또는 <흐름 1>과 <흐름 3>은 동일한 기능을 수행하는 활동의 개수가 다르기 때문에, 워크플로우 가변성이 발생한다.

컴포지션 가변성은 유즈케이스 다이어그램을 분석하여 식별할 수 있으며, 그림 9와 같이 단위 서비스에 여러 개의 유즈케이스만 포함되어 있는 경우에 발생할 수 있다. 예를 들어, 단위 서비스 X는 소비자에 따라 include 관계가 있는 유즈케이스 A와 유즈케이스 B가 포함되거나, include 관계가 있는 유즈케이스 A와 유즈케이스 C가 포함된다. 그러므로, 소비자에 따라 유즈케이스 B 또는 유즈케이스 C가 적용되므로, 컴포지션 가변성이 되는 것이다. 즉, 컴포지션 가변성은 유즈케이스 간의 조합에 가변성이 발생하는 것이며, 이는 다른 컴포넌트의 오퍼레이션 또는 서비스를 호출하는 것으로 설계된다.

QoS 가변성은 동일한 기능을 여러 소비자가 공통으로 필요로 하지만, 각 소비자가 다른 QoS 값으로 서비스를 요청할 때 발생하는 가변성이다. QoS 가변성은 SRS의 비기능적 요구사항을 분석함으로써 도출된다.

둘째, 가변성 범위에 대하여 분석을 한다. 가변성 범위는 한 가변점에 할당될 수 있는 가변치의 개수에 따라 결정되며, 해당 가변점에 대해 선택 가능한 가변치가 2개가 있으면 Binary, 3개 이상이 있으면 Selection이 된다. 그리고, 분석 시점에 가변치가 추가될 가능성이 있으면 가변성 범위가 Open이 된다[3]. Open 기능은 소비자가 원하는 가변치를 재사용 기능 단위에 추가하여 사용할 수 있게 하지만, 모든 기능이 인터넷에 존재하고 소비자는 해당 기능을 호출하여 사용한다는 SaaS의 특징 때문에 적용할 수 없다. 그러므로, SaaS 가변성 범위로는 다음과 같이 Binary와 Open만 고려한다.

가변성 범위 집합 = {Binary, Selection}

표 5 가변성 분석표

단위서비스 이름	가변점	가변성 종류	소비자 별 가변치				가변성 범위
			SRS_1	SRS_2	…	SRS_n	
…	…	…	…	…	…	…	…

이제, SaaS 고유의 가변성 특징을 고려하여 가변점, 가변치, 가변성 종류, 가변성 범위 관점으로, 단위 서비스의 가변성을 분석하여 표 5를 작성한다.

"단위서비스 이름" 열에는 이전 단계에서 식별된 단위서비스의 이름을 기술하고, "가변점" 열에는 해당 가변점을 식별할 수 있는 고유한 아이디 또는 이름을 작성한다. "가변성 종류" 열에는 가변점에서 발생한 SaaS의 가변성 종류를 기술하며, "소비자 별 가변치" 열에는 소비자 별 가변점에 대한 가변치를 기술한다. 가변치의 개수를 기준으로 "가변성 범위" 열을 작성한다.

5.3 단계 300. SaaS 설계

이 단계에서는 공통성과 가변성 분석 결과를 기반으로 SaaS를 설계한다. SaaS는 소프트웨어를 서비스 형태로 제공하므로, 단위 서비스 외에 시스템 아키텍처와 사용자 인터페이스 설계도 필요하다. 이 단계를 통하여, 단위 서비스 설계서, 아키텍처 명세서, 소비자 인터페이스 설계서 등으로 구성된 SaaS 설계서를 만든다.

5.3.1 활동 310. 단위 서비스 설계

이 활동에서는 공통성과 가변성 분석 결과를 이용하여, 적용성과 적응성이 높은 단위 서비스를 설계한다. 단위 서비스는 SOA 서비스와 비슷한 특성을 가지고 있으므로, 서비스 인터페이스와 서비스 구현체 관점에서 설계하여야 하며, 단위 서비스들과 관련된 데이터베이스 설계도 수행해야 한다.

먼저, 서비스 인터페이스를 설계한다. 서비스 인터페이스는 SaaS 사용자 인터페이스(즉, SaaS를 사용할 수 있게 하는 웹 페이지)에서 호출되는 오퍼레이션들의 집합이고, 각 오퍼레이션은 오퍼레이션 이름, 입력 매개변수, 출력 매개변수, 사전 조건, 사후 조건과 그 외 시맨틱 정보를 이용하여 정의된다. 서비스 인터페이스의 오퍼레이션은 대개 액터가 유즈케이스를 호출하는 부분에서 유도된다.

둘째, 단위 서비스 구현체를 설계한다. 단위 서비스 구현체는 객체 또는 컴포넌트이다. SaaS의 단위 서비스는 그림 10과 같이 새로운 컴포넌트를 개발, 배포된 서비스를 이용하여 컴포넌트를 개발, 레거시 어플리케이션을 이용하여 컴포넌트를 개발하는 세 가지 방법으로 구현될 수 있다.

그리고, 배포된 서비스나 레거시 어플리케이션을 이용하는 방법은 크게 세 가지로 분류가 된다.

그림 10 단위 서비스 구현 방법

- 경우 1) 배포된 서비스나 레거시 어플리케이션이 제공하는 기능과 단위 서비스의 기능이 완전히 일치하는 경우로, 이 기능을 그대로 사용하면 된다. 이 때, Façade 패턴을 이용하여 제공하는 기능없이 재사용 기능을 호출하는 역할만을 하는 컴포넌트를 설계가 가능하다[13].
- 경우 2) 배포된 서비스나 레거시 어플리케이션이 제공하는 기능이 단위 서비스의 기능 중 일부를 제공하는 경우이다. 이 경우에는 배포된 서비스나 레거시 어플리케이션을 래핑하여 추가적인 기능을 구현하도록 한다. 이 때, Decorator 패턴을 이용하여 설계해야 한다[13].
- 경우 3) 배포된 서비스나 레거시 어플리케이션이 제공하는 기능이 필요로 하는 기능보다 더 많은 것을 제공하는 경우이다. 이 경우에는 ROI를 분석하여, 새로 컴포넌트를 개발하거나 추가적인 기능을 예외 처리를 하여 사용할 수 없도록 해야 한다.

구현 방법을 결정하기 위하여, 먼저 서비스 분석서에 기술된 각 단위 서비스 기능성을 고려하여 이용 가능한 서비스나 레거시 어플리케이션이 존재하는지를 확인한다. 예를 들어, 지도 상의 거리를 나타내는 기능을 제공하는 단위 서비스가 식별되었다면, Google 등에서 제공하는 지도 서비스를 사용할 수 있는 것이다.

이용 가능한 서비스 또는 레거시 어플리케이션을 검색한 후, ROI 분석 등을 통하여 표 6을 작성함으로써, 각 단위 서비스를 개발하는 방법을 결정한다.

표 6 단위 서비스 설계 결정표

서비스	이용 가능한 자산	구현 방법
…	…	…

그리고, 결정한 구현방법에 따라 다음과 같이 서비스 컴포넌트를 설계한다. 이 때에는 공통성 기능 외에 가변성 정보도 반드시 설계해야 한다.

새 서비스 컴포넌트 설계: 서비스 컴포넌트는 대개 객

체나 컴포넌트로 구현한다. 따라서 기존의 객체지향설계 또는 컴포넌트 기반 설계 방법을 이용하여 서비스 컴포넌트의 정적 모델과 동적 모델을 설계한다. 여기서 주의할 점은, 각 모델에 가변성이 발생하는 곳을 명시하기 위해서 가변성을 나타내는 스테레오 타입인 «*variability*»를 표시하도록 한다[14].

배포된 서비스를 이용하여 서비스 컴포넌트 설계: 기존의 배포된 서비스를 앞서 언급한 경우에 따라서 그대로 이용하거나 래핑하는 서비스 컴포넌트를 설계한다. 먼저, 단위 서비스 중 어느 기능이 배포된 서비스로 대체될 수 있는지를 확인한다. 그리고, 단위 서비스와 배포된 서비스 간의 기능성 관계를 분석하여 앞서 언급한 배포된 서비스를 이용하는 방법 중 하나를 선택한다. 즉, Façade 패턴이나 Decorator 패턴을 이용하여 컴포넌트를 구성한다. 따라서 래퍼 컴포넌트는 WSDL 인터페이스를 구현하고, 기존의 배포된 서비스 정보와 WSDL 인터페이스가 서로 일치하게 매핑하는 역할을 하는 오퍼레이션으로 정의한다. 모든 설계와 관련된 결정사항은 서비스 설계 명세서에 기술한다.

레거시 어플리케이션을 이용하여 서비스 컴포넌트 설계: 레거시 어플리케이션을 래핑하는 서비스 컴포넌트를 설계한다. 소비자가 WSDL 인터페이스를 통해 SaaS를 호출할 때, 레거시 어플리케이션을 래핑하는 컴포넌트는 내부적으로 레거시 어플리케이션을 호출한다. 레거시 어플리케이션을 래핑하는 서비스 컴포넌트의 설계는 배포된 서비스를 래핑하는 서비스 컴포넌트와 동일하게 시행한다. 모든 설계와 관련된 결정사항은 서비스 설계 명세서에 기술한다.

셋째, 구현된 단위 서비스에 내포된 가변성에 대하여, 소비자가 특정 가변치를 선택할 수 있도록 가변성 설정 관련 인터페이스, 클래스 및 메소드를 설계한다. 앞서 설계한 서비스 인터페이스는 *Provided* 인터페이스에 해당이 되며, 가변성을 위해 설계되는 인터페이스는 *Required* 인터페이스에 해당된다[3]. 가변성을 설계하기 위해서는 활동 220에서 만든 가변성 분석서 내용 중 "가변성 범위"를 참고하여 오퍼레이션을 정의한다.

넷째, SaaS에 대한 데이터 베이스를 설계하여 ER 다이어그램을 작성한다. 전통적인 소프트웨어 어플리케이션과 다르게 SaaS를 위한 어플리케이션의 설계 시에 주의할 점은 SaaS가 멀티 테넌트를 지원한다는 것이다. 멀티 테넌트를 고려한 데이터 베이스 설계방법은 일반적으로 세 가지로 구분한다[15]. SaaS 개발자는 이 중 개발하려는 SaaS의 도메인과 개발 여건에 따라서 데이터 베이스 설계 방법을 선택한다.

5.3.2 활동 320. SaaS 시스템 아키텍처 설계

이 활동에서는 4장에서 언급한 SaaS의 재사용성을 높이기 위한 요구사항 중에서 높은 확장성을 보장하기 위한 SaaS 아키텍처를 설계한다. 소프트웨어 아키텍처는 목표 소프트웨어 시스템의 주요 컴포넌트와 그들의 관계를 정의하는 것으로, 요구사항 명세서 중 비기능적(Non-Functional) 요구사항과 품질 요구사항을 반영하는 역할을 한다[16]. SaaS의 확장성은 기존의 소프트웨어와 구별되는 특징이기 때문에, 이 활동에서 다른 비기능적 요구사항보다 확장성에 초점을 맞춘다.

확장성 및 기타 비기능적 요구사항을 만족하는 SaaS 시스템 아키텍처를 설계하기 위하여, 전통적인 아키텍처 설계 프로세스를 이용하여 그림 11과 같이 세 가지의 단계로 수행한다[16].

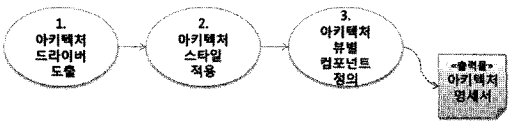


그림 11 SaaS 아키텍처 설계의 주요 활동

먼저, SaaS 시스템 아키텍처가 반드시 만족해야 하는 아키텍처 드라이버를 도출한다. 이 때에는 확장성 외에 SaaS 개발 도메인에 특화된 다양한 비기능적 요구사항을 식별한다. 본 논문에서는 확장성 높은 SaaS 아키텍처 설계에 중점을 두고 있으므로, 다음과 같은 특징을 고려하여야 한다.

서비스 자원 확장의 용이성: SaaS의 확장성을 높이는 가장 손쉬운 방법은 부족한 자원을 추가하는 것이다. 즉, 서비스의 보유 자원 량을 확장할 수 있는 구조를 가질수록 SaaS의 확장성은 좋아진다. 이를 위해, 서비스 제공자는 서비스의 자원에 대한 직접적인 접근을 지양하고, 참조를 통하여 접근해야 한다. 서비스 자원에 대한 직접적인 접근은 서비스와 소요 자원의 단단한 연결(closely coupling)을 의미하고, 소요 자원에 대한 변경이 어렵다. 반면에, 참조를 통한 서비스와 소요 자원의 느슨한 연결(loosely coupling)은 참조 주소의 변경 혹은 추가를 통하여, 소요 자원의 변경 또는 추가에 용이하다. 소요자원의 변경과 추가를 통한 자원의 확장으로 해당 서비스의 확장성은 좋아진다.

높은 자원 사용 효율성: 한번의 서비스 호출에 의해 소요되는 자원 량은 미리 정의되어 있기 때문에, 서비스가 보유한 자원 중 휴면 상태의 자원이 없도록 관리하는 것이 서비스의 확장성 측면에서 중요하다. 특히, 분산 환경으로 구성된 자원의 경우에 발생된 총 부하 량을 각각의 분산된 자원에 균등하게 분배되어야 한다. 예를 들어, 발생된 부하 량이 균등히 분배되지 않는 시스템에서는 특정 자원에 부하가 집중될 수 있고, 이는 총

분한 여유 자원을 보유한 상태에서도 서비스의 전체 품질을 떨어뜨리는 원인이 된다.

공용 자원의 사용 지양: 공용 자원이란 각각의 서비스 인스턴스가 실행이 될 때, 공통으로 사용하게 되는 자원이다. 공용 자원이 한번에 처리할 수 있는 처리량에는 한계가 존재한다. 따라서 서비스 사용량이 급증하였을 때, 동시 처리량이 증가함으로써 공용 자원에 의한 병목 현상이 발생할 수 있다. 즉, 공용 자원에 의한 병목 현상으로 전체 서비스의 성능 및 품질이 하락한다. 이런 현상을 줄이기 위해서는 최대한 공용 자원을 줄여야 한다.

위의 특징을 아키텍처 설계에 반영하기 위해서는 대부분 서버 측에 운영되는 기능을 한 곳에 집중시키기보다는 분산시키는 것이 효율적이다. 한 서비스 제공자 서버나 이 서버의 기능모듈이 기능을 수행하지 못할 경우에 다른 곳에서 그 기능을 대신 수행할 수 있기 때문이다. 이 경우, 마스터 슬래이브(Master-Slave) 아키텍처 스타일은 이들 간의 로드 밸런싱이나 실시간 업무이관(Task Migration)을 지원할 수 있다. 또 다른 방법으로는 하나의 기능 또는 자원에 대해 여러 Replica를 생성하여, 한 노드가 기능 호출을 허용하지 못할 경우 다른 복제된 노드(Replica)로 업무 이관을 하는 것이다[17]. 이외에 SaaS 시스템은 기본적으로 웹 상에서 운영되는 소프트웨어 형태의 서비스이므로, 클라이언트-서버 아키텍처 스타일 또는 MVC 아키텍처 스타일을 적용하여 아키텍처를 설계해야 한다[16].

마지막으로, 아키텍처를 설계하는 동안에 결정된 모든 사항을 아키텍처 설계서에 작성한다. 아키텍처 설계서는 선정한 아키텍처 스타일과 아키텍처 뷰에 대한 정보를 기술한다.

5.3.3 활동 330. 소비자 인터페이스 설계

이 활동에서는 SasaS 소비자가 웹 브라우저를 통해 SaaS를 사용할 수 있게 하는 사용자 인터페이스를 설계한다. SaaS는 브라우저를 통해 소비자가 기능을 사용할 수 있기 때문에, 기존의 웹 어플리케이션의 사용자 인터페이스 설계 기법 이용하도록 한다[18].

5.4 단계 400. SaaS 구현

이 단계에서는 이전 단계에서 정의된 서비스에 대한 설계를 수행하고 결정된 구현방법에 따라 SaaS를 구현한다. SaaS 구현은 크게 공통성 및 가변성을 포함하는 SaaS 시스템과 사용자 인터페이스 구현으로 구분이 되며, 이를 기반으로 SaaS 시스템을 검증해야 한다.

5.4.1 활동 410. SaaS 시스템 구현

이 활동에서는 특정 플랫폼 및 프로그래밍 언어를 사용하여 SaaS 시스템을 구현한다. 이 단계는 객체 지향 개발 방법을 이용하여 객체 지향 시스템을 개발하는 방법을 이용하면 된다. 그러나, 주의해야 할 점은 개발 대상인 SaaS에서 제공되는 단위 서비스의 공통성뿐 아니라 가변성 설계 결과도 특정 프로그래밍 언어를 사용하여 구현해야 한다. SaaS 시스템이 제대로 설계되었다면, 이 활동은 비교적 수월하게 진행된다. 즉, 설계 모델을 기반으로 단위 서비스 및 SaaS에 해당되는 클래스 및 클래스 간의 연관 관계를 구현한다.

5.4.2 활동 420. 소비자 인터페이스 구현

이 활동에서는 소비자 인터페이스 설계 결과를 이용하여 소비자 인터페이스를 구현한다. SaaS는 일반 웹 어플리케이션과 같이 웹 브라우저를 통해 소비자가 소프트웨어 기능을 제공받기 때문에, 웹 어플리케이션의 소비자 인터페이스 구현 방법을 이용하여 소비자 인터페이스를 구현한다.

5.4.3 활동 430. SaaS 시스템 검증

이 활동에서는 활동 410과 활동 420에서 구현한 시스템을 검증한다. SaaS 시스템 검증 역시 전형적인 소프트웨어를 검증하는 기법을 이용하여 수행한다. 그러나, 전형적인 소프트웨어와는 다르게 SaaS는 인터넷을 통하여 실행되기 때문에, 3절에서 언급한 재사용성에 관련된 3가지 부 특성과 웹 상에서 발생 가능한 문제들을 해결할 수 있는지에 초점을 맞추어 SaaS를 검증해야 한다.

6. 사례연구

본 장에서는 5장에서 제안한 개발 프로세스가 SaaS 재사용성의 세 가지 부 특성인 적용성, 적응성, 확장성을 실제로 반영하는지를 검증하기 위한 사례연구를 수행한다. 사례연구에서는 제안된 개발 프로세스의 각 활동 중 SaaS 재사용성의 부 특성을 반영하는 활동에 초점을 맞춰서 개발 프로세스의 유효성을 검증한다.

6.1 '단계 100. 서비스 요구사항 정의' 적용

• 활동 110. 소프트웨어 요구사항 수집

사례연구를 위해서 수집된 요구사항은 해외 여행자, 영화 동호회원, 초등학생을 둔 부모의 각기 다른 요구사항을 가지고 있는 세 명의 소비자들에게서 얻어진 것이다. 해외 여행자는 해외 여행지에서 찾을 회원의 검색, 검색된 회원들에 대한 친구 등록, 친구와의 만남을 위한 길 안내 서비스 등을 요구한다. 영화 동호회원은 영화에 관심이 있는 회원들 간의 친구 맺기, 회원들의 관심사항을 검색할 수 있는 기능, 영화를 보기 위한 극장정보의 제공 등을 요구한다. 마지막으로 초등학생을 둔 부모는 아이의 현재 위치를 검색하고 아이가 이동하는 경로를 실시간으로 표시해주는 기능 등을 요구한다.

• 활동 120. 소프트웨어 요구사항 정규화

이 활동에서는 각 소비자들에게서 받은 요구사항을 정규화하여 SRS를 재 작성한다.

• 활동 130. 공통 요구사항 식별

이 활동에서는 정규화된 각 소비자들의 요구사항에서 기능성을 추출하여 공통 기능성을 식별하기 위해 UML의 기능성 모델링 방법인 유즈케이스 다이어그램을 이용한다. 정규화된 각각의 요구사항들은 각각 14개, 13개, 14개의 유즈케이스로 구성된다. 그 중 8개의 유즈케이스에서 공통적인 기능 휘처가 사용되었으며 그 외의 나머지 유즈케이스는 각 SRS에 고유한 기능 휘처를 나타내는 유즈케이스이다. 정규화된 요구사항들에서 도출된 공통 기능 휘처와 공통성의 정도를 식별하기 위해 친구 찾기 SaaS를 위한 공통 기능 비교 분석표를 작성한다.

'회원 검색'과 '친구 등록'은 친구 찾기 SaaS에 종속적인 기능휘처이기 때문에 모든 소비자들이 공통적으로 이용하는 기능이다. 그러나 '현재상태 설정'은 해외 여행자에게만 종속적인 기능이므로 상대적으로 공통성이 낮게 된다. 얻어진 공통성 값은 활동 130에서 제시된 공통성을 구하는 공식을 적용한 결과이다. '회원 검색'과 '친구 등록'의 공통성 값인 1.0은 해당 기능을 요구하는 소비자 수가 3명이고 전체 소비자 수가 3명이기 때문에 공통성을 구하는 공식에 적용하여 1.0이라는 공통성 값을 얻게 된다. 반면에 현재상태 설정은 오직 해외 여행자만이 이용하는 기능 휘처이므로 공통성 값이 0.33이 된다.

6.2 '단계 200. SaaS 분석' 적용

• 활동 210. 단위 서비스 식별

이 활동은 표 7에서와 같이 친구 찾기 SaaS의 단위 서비스 식별을 위한 SaaS 개발 영역 결정표에서 각 기능 휘처들의 공통성과 휘처들 간의 의존성, 시장성 등의 기타 기준을 고려하여 친구 찾기 SaaS에서 제공될 수 있는 기능인지 여부를 분석한다.

또 표 8에서 '최단경로 탐색', '최소비용 경로탐색', '실시간 경로탐색'은 모두 경로 탐색의 한 방법들이므로 '경로 탐색'과의 의존성이 높다. 그 중에서 '최소비용 경로탐색'과 '실시간 경로탐색'은 세 명의 소비자 중 한 명에 의해서만 요구되는 기능이므로 공통성 값이 모두 0.33이다. 하지만 실시간 경로탐색은 초등학생을 둔 부모가 주로 이용하는 기능이므로 시장성이 높은 반면 '최소비용 경로 탐색'은 해외 여행자가 주로 이용하는 기능이 아니기 때문에 시장성이 낮다. 따라서 5.2.1 절의 활동 210에서 제시한 가이드 라인에 의해서 친구 찾기 SaaS의 개발 영역에 포함되지 않는다.

친구 찾기 SaaS에서 개발될 기능 휘처들이 결정되면 단위 서비스를 식별하고 이 결과를 단위 서비스 목록에 기술한다. 이렇게 모든 단위 서비스들이 식별되면 단위 서비스 명세서를 작성하여 단위 서비스의 구체적인 내용을 명확히 한다.

• 활동 220. 서비스 가변성 분석

이 활동에서는 이전 활동에서 식별된 친구 찾기 SaaS의 단위 서비스들에 대한 가변성 분석을 수행한다. 표 9는 친구 찾기 SaaS가 제공하는 서비스 중 회원검색 서비스에 대한 가변성 분석을 수행하고 정리한 가변성 분석표이다.

회원검색 서비스는 QoS와 인터페이스 가변성을 갖는다. 인터페이스 가변성의 경우는 회원검색 서비스를 이용하기 위해서 입력하는 매개변수와 그 매개변수의 데이터 타입에 따라 다른 로직으로 회원검색 서비스가 제공하기 때문에 가변성이 발생한다.

표 7 친구 찾기 SaaS를 위한 공통 기능 비교 분석표

기능 휘처 이름	기능 설명	소비자 집합			공통성
		해외 여행자	영화 동호회원	초등학생을 둔 부모	
회원 검색	소비자가 원하는 조건에 맞는 회원을 검색한다.	○	○	○	1.0
친구 등록	검색된 회원들 중 소비자가 원하는 회원을 선택하여 친구로 등록한다.	○	○	○	1.0
...					
현재상태 설정	현재 서비스의 이용여부를 설정한다.	○			0.33

표 8 친구 찾기 SaaS를 위한 개발 영역 결정표

기능 휘처 이름	공통성	기타 기준			SaaS 개발영역 (Y/N)	근거
		의존성이 높은 기능 휘처	시장성	...		
회원 검색	1.0				Y	가이드라인 #2)에 해당
...						
경로 탐색	0.67		높음		Y	가이드라인 #2)과 #3)에 해당
최단경로 탐색	0.67	경로탐색	높음		Y	가이드라인 #2)과 #3)에 해당
최소비용 경로탐색	0.33	경로탐색	낮음		N	
실시간 경로탐색	0.33	경로탐색	높음		Y	가이드라인 #1)의 예외경우와 #3)에 해당

표 9 친구 찾기 SaaS의 가변성 분석표

단위서비스 이름	가변점	가변성 종류	소비자 별 가변치			가변성 범위
			해외 여행자	영화 동호회원	초등학생을 둔 부모	
회원검색	VP01	QoS	…	…	…	Selection
	VP02	인터페이스	{검색범위}	{관심사항}	{주민등록번호}	Selection
…	…	…	…	…		

6.3 '단계 300. SaaS 설계' 적용

• 활동 310. 단위 서비스 설계

이 활동에서는 친구 찾기 SaaS의 단위 서비스에 대한 설계를 수행한다. 먼저 단위 서비스 구현체의 설계를 위해서는 정적 모델과 동적 모델링이 필요한데 여기에서는 이것들을 위해 UML의 클래스 다이어그램과 시퀀스 다이어그램을 이용한다. 그림 12는 회원검색 서비스를 위한 클래스 다이어그램의 일부를 보여준다.

Contoller 클래스는 일반적인 클래스 다이어그램과는 다르게 «*variability*» 스테레오 타입과 {type=interface} 제약사항을 볼 수 있다. 이것들은 가변성의 표현을 위한 것으로, «*variability*» 스테레오 타입은 회원검색 서비스에서 발생하는 가변점을 의미하고 {type=interface}은 그 가변점에서 발생하는 가변성 타입을 설명하기 위한 것이다. 그리고 Map 클래스는 Google에서 제공하는 맵 서비스를 래핑하는 클래스이므로 «use»스테레오 타입을 사용하여 표현하였다. 회원검색 서비스의 동적 모델링을 위한 시퀀스 다이어그램은 지면상의 이유로 본 논문에서는 기술하지 않는다.

그림 13에서 보는 것과 같이 회원검색 서비스의 데이터베이스는 세 개의 테이블로 구성된다. Multi-tenant를 위한 다양한 데이터베이스 설계방법이 있지만 본 논문에서는 서비스 제공자의 경제성 등을 고려하여 각 tenant들을 하나의 데이터베이스에 3개의 스키마로 따로 분리/저장한다. 이것은 multi-tenant 환경에서 야기될 수 있는 데이터 베이스 무결성 등의 문제점을 고려한 것이다.

Member 테이블은 친구 찾기 SaaS 회원들의 개인정보를 저장하기 위한 것이다. 여기서 유의할 점은 회원은 친구로 등록된 다른 회원과의 관계가 성립되기 때문에 재귀적으로 회원 테이블과 관계를 맺는다는 것이다. 그리고 Variability 테이블은 회원이 친구 찾기 SaaS에서

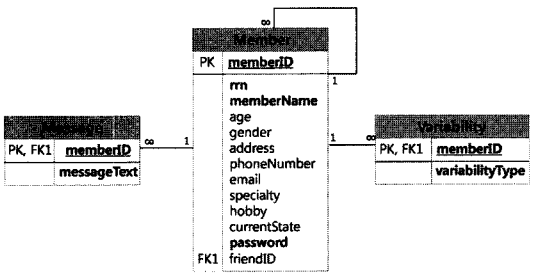


그림 13 회원검색 서비스의 데이터베이스 설계

설정한 가변점과 가변치에 대한 정보를 저장한다. 이것은 회원이 설정한 가변성 정보를 저장한 후에 다시 서비스 이용을 위해 그 회원이 친구 찾기 SaaS에 접속하면 설정한 가변성을 그대로 제공해주기 위한 것이다. 마지막으로 Message 테이블은 회원이 다른 회원들에게 보낸 메시지들을 저장하기 위한 테이블이다.

• 활동 320. SaaS 시스템 아키텍처 설계

이 활동에서는 친구 찾기 SaaS의 시스템 아키텍처를 설계한다. 공통 요구사항으로부터 확장성 외에 성능과 사용성(Usability)이 아키텍처 드라이버로 선정되었다. 표 10은 선정된 아키텍처 드라이버를 만족하기 위한 아키텍처 설계 결정사항을 보여준다.

위의 결정사항은 아키텍처 명세서의 일부로 포함이 되며, 이를 기반으로 그림 14의 아키텍처 설계 모델을 작성하였다. 그림의 왼쪽 부분은 MVC 아키텍처 스타일이 적용된 기능 뷰를 보여준 것이며, 오른쪽 부분은 Master-Slave 아키텍처 스타일이 적용된 배치뷰의 일부를 보여준다.

• 활동 330. 소비자 인터페이스 설계

이 활동에서는 친구 찾기 SaaS의 소비자 인터페이스를 설계한다. SaaS의 인터페이스는 일반적인 웹 어플리

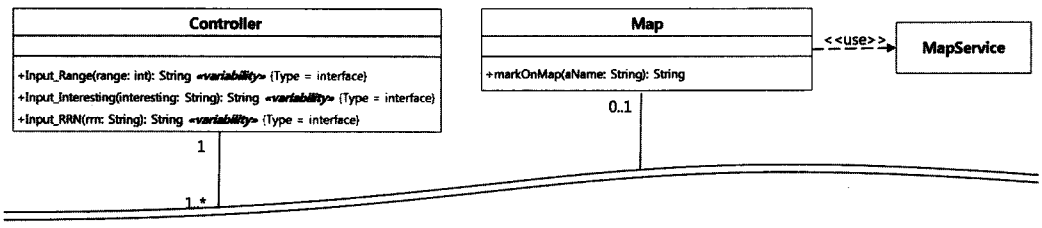
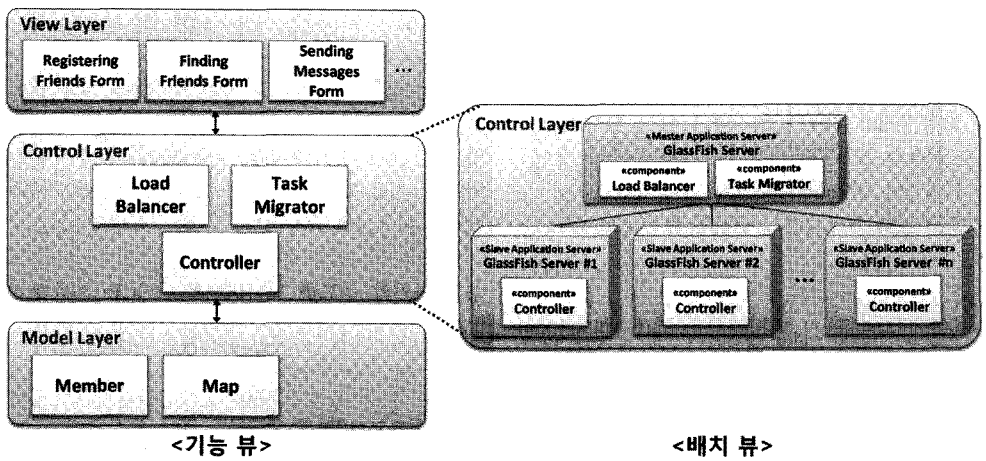


그림 12 회원검색 서비스를 위한 클래스 다이어그램

표 10 친구 찾기 SaaS를 위한 아키텍처 결정사항(Design Decisions)

아키텍처 드라이버	아키텍처 스타일	적용한 뷰	근거
확장성과 성능	Master-Slave 아키텍처 스타일	·기능뷰를 통해 Model, Control, View 계층에 할당되는 컴포넌트를 결정한다. ·배치뷰를 통해, Master 노드와 Slave 노드를 결정한다. ·동시뷰를 통해 Model, Control, View가 어떻게 상호작용하는지를 보여준다.	한 노드로 집중되는 기능호출을 여러 곳으로 분산하여 피크타임 시에 확장성을 보장할 수 있도록 한다.
성능	Master-Slave과 MVC 아키텍처 스타일		확장성이 보장이 되면, 어느 정도의 성능 역시 보장이 된다. 이 외에 성능이 보장이 되기 위해서는 각 계층별로 상호작용의 횟수가 적어야 하므로, MVC 아키텍처를 적용하여 각 계층 간의 상호작용 횟수를 최소화시키도록 한다.
사용성	MVC 아키텍처 스타일		Model-Controller-View를 분리시킴으로써, 소비자는 내부 흐름을 관여하지 않고, 잘 설계된 View 계층만 접근함으로써 SaaS 기능을 쉽게 사용할 수 있게 한다.



<기능 뷰> <배치 뷰>

그림 14 친구 찾기 SaaS 시스템 아키텍처 설계 모델 - 기능 뷰

케이션의 인터페이스와 동일하기 때문에 웹 어플리케이션의 소비자 인터페이스 설계를 위한 절차를 따른다. 이 논문에서는 지면상의 제약으로 더 이상의 기술은 하지 않기로 한다.

6.4 '단계 400. SaaS 구현' 적용

• 활동 410. SaaS 시스템 구현

이 활동에서는 앞에서 설계된 내용을 바탕으로 친구 찾기 SaaS의 회원검색 서비스에서 발생하는 가변성에 대한 구현을 수행한다. 회원검색 서비스의 가변성이 구현됨으로 인해서 각 소비자는 원하는 목적에 따라 가변치를 설정하여 회원검색 서비스를 이용하게 된다.

그림 15는 친구 찾기 SaaS 서비스의 구현 코드 중 가변성 설정 클래스를 보여준다. 가변성 설정 클래스는 Variability 클래스의 일부로써 소비자가 선택한 가변성을 관리하기 위한 부분이다. 소비자가 선택한 가변성에 따라 다르게 입력된 가변치는 다른 Variability 객체를 생성함으로써 소비자가 선택한 가변성을 시스템에서 관리할 수 있게 한다. 회원 검색 조건에 대한 가변치를 설정하는 부분은 각 소비자가 선택한 가변성과 이에 따라

```
// 회원 검색 서비스 이용을 위한 가변성 설정 클래스
public class Variability {
  ...
// 회원 검색 조건 (검색 범위)에 설정된 가변치를 관리하기 위한 Variability 생성자
   public Variability(int aRange, String aMemberID) {
      range = aRange;
      memberID = aMemberID;
      ...
   }
   ...

// 회원 검색 조건 (검색 범위)에 대한 가변치를 설정하는 오퍼레이션 - '해외여행자'를 위한 가변치 설정 메소드
   public String setLocationVariability() {
      return Member.searchLocation(this.range);
      // 가변치를 지정함.
   }
   ...
}
```

그림 15 회원 검색 서비스 이용을 위한 가변치 입력 클래스와 가변성 설정 클래스

입력한 가변치들에 의해서 다른 종류의 검색 방법으로 회원을 검색하는 것을 보장해준다. setLocationVariability 오퍼레이션은 소비자가 입력한 범위를 이용하여 회원검색을 하므로 다른 값을 입력하는 회원 검색 오퍼레이션과는 다른 검색 과정과 결과를 가진다.

그림 16은 해외 여행자가 회원검색 서비스를 실행시킨 결과이다. 이 결과는 소비자가 입력한 범위 내에 있는 회원들에 대한 정보와 위치를 보여준다. 소비자가 가

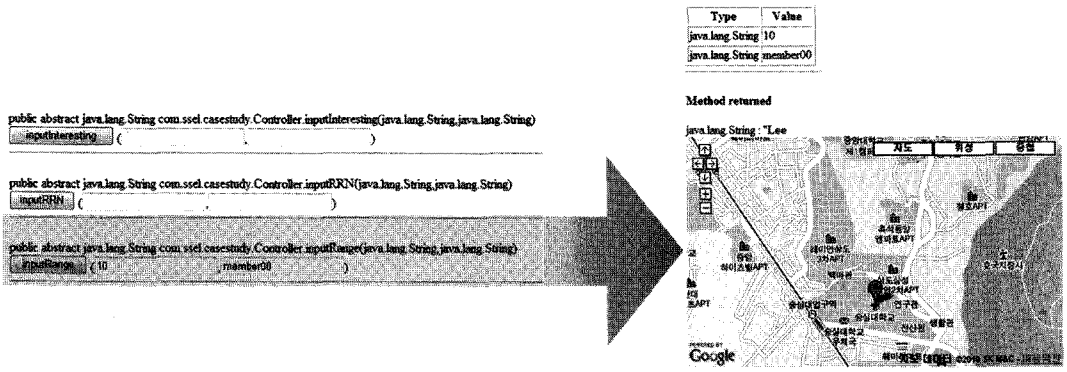


그림 16 가변성 설정을 위한 인터페이스와 해외 여행자를 위한 서비스 실행결과

변치로써 10을 입력하면 친구 찾기 SaaS는 소비자의 위치에서 반경 10km 이내에 위치하고 있는 회원을 검색하여 그 이름을 출력하고 회원의 위치는 지도 상에 아이콘을 이용하여 보여준다.

• 활동 420. 소비자 인터페이스 구현과 활동 430. SaaS 시스템 검증

설계한 친구 찾기 SaaS의 소비자 인터페이스를 구현하고, SaaS의 전체적인 시스템 품질을 검증한다. 소비자 인터페이스는 구글 Map 서비스를 이용하여 지도 관련 화면을 구현하였으며, 블랙박스와 화이트박스 테스팅을 시행하여, SaaS가 제대로 동작하는지를 확인하였다. 지면의 제약상 상세한 기술은 하지 않는다.

6.5 사례연구를 통해서 본 프로세스 평가

본 장에서는 SaaS 재사용성의 세 가지 부 특성인 적용성, 적응성, 확장성이 사례연구를 통해서 수행된 각 활동들이 실제로 어떻게 반영되었는지를 평가한다.

그림 17은 SaaS 재사용성에 대한 세 가지 부 특성과 이와 연관이 있는 SaaS 개발 프로세스의 활동들과의 관계를 보여준다. 그림의 왼쪽과 오른쪽에는 SaaS 재사용성의 부 특성과 SaaS 개발 프로세스의 활동들을 각각 나열하였으며, 이들 간에는 «reflects on» 관계로 연결된다. «reflects on»은 특정 SaaS 재사용성의 부 특성이 특정 활동에 반영되어 상세 지침을 정의하였음을 보여주는 것으로 직접적으로 반영된 활동과 간접적으로 반영된 활동으로 구분된다.

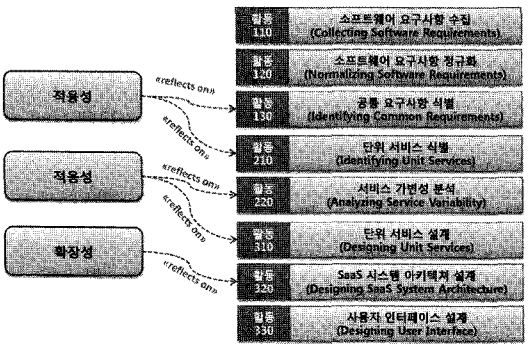


그림 17 SaaS 재사용성의 부 특성과 SaaS 개발 프로세스 간의 연관관계

적용성은 SaaS가 얼마나 많은 소비자들에게 적용될 수 있는가의 정도를 나타내므로, 공통성 분석과 연관이 있다. 그러므로 적용성은 소비자가 필요로 하는 기능을 분석하여 서비스를 식별하는 활동 130과 활동 210에 직·간접적으로 반영되었다.

• 직접적으로 반영된 활동

활동 130. 공통 요구사항 식별: 회원 검색 기능 휘처는 공통성 값이 1이기 때문에 세 명의 소비자들이 모두 사용하기를 원하는 기능성이다. 따라서 회원 검색 기능 휘처를 제공하는 것은 세 명의 소비자들이 모두 사용하는 기능성을 제공하여 친구 찾기 SaaS의 이용을 증가시킨다. 이 결과는 일반적인 소비자들의 경우에도 해당되므로 공통 요구사항을 식별하는 것은 보다 많은 소비스 서비자들에게 SaaS를 이용하게 하여 적용성을 직접적으로 증가시킨다.

• 간접적으로 반영된 활동

활동 210. 단위 서비스 식별: 소비자들이 요구한 경로탐색 방법 중 최소비용 경로탐색과 실시간 경로탐색은 모두 공통성 값이 0.33이지만 실시간 경로탐색 방법은 초등학생을 둔 부모가 친구 찾기 SaaS를 이용하는 주목적 중에 하나이므로 서비스로 식별된다. 하지만 최소비용 경로탐색은 해외 여행자가 이용하는 경로탐색 방법 중 최단경로 탐색보다 상대적으로 이용횟수가 적어 시장성이 낮을 것으로 분석되기 때문에 친구찾기 SaaS의 서비스로 포함되지 못했다. 이 결과는 일반적인 경우에도 해당되므로 보다 많이 이용될 수 있는 기능성을 식별하고 SaaS에서 제공하는 서비스로 포함함으로써 적용성이 증가될 수 있다.

적응성은 특정 기능을 소비자의 요구사항에 맞게 특화하여 얼마나 효과적 또는 효율적으로 SaaS를 사용할 수 있는 가의 정도를 나타내므로, 가변성 분석과 연관이 있다. 그러므로 가변성을 모델링 하는 활동 220과 활동 310에 직·간접적으로 반영되었다.

• 직접적으로 반영된 활동

활동 220. 서비스 가변성 분석: 회원 검색 서비스에 대한 가변성 분석을 통해 검색범위, 관심사항, 주민등록번호의 세 가지 가변치를 가지는 가변점이 식별된다. 그리고 해외 여행자, 영화 동호회원, 초등학생을 둔 부모가 각각 자신에게 맞는 가변치를 설정하여 친구 찾기 SaaS를 사용하는 것은 각 소비자들에게 맞는 특화된 서비스를 이용할 수 있게 한다. 회원 검색 서비스의 예를 일반화해 볼 때, 다수의 소비자들의 요구사항을 분석하고 그들이 가지는 가변성을 식별함으로써 보다 많은 사람들에게 특화된 서비스를 제공하는 것은 SaaS의 적응성을 증가시킨다.

• 간접적으로 반영된 활동

활동 310. 단위 서비스 설계: 친구 찾기 SaaS는 소비자들이 가변치를 설정할 수 있는 Input_Range(), Input_Interesting(), Input_RRN()의 인터페이스를 단위 서비스 설계에 반영함으로써 서비스 분석 시에 식별된 가변성을 지원하는 장치를 제공한다. 일반적인 경우에도 서비스 가변성 분석 시에 식별된 가변성을 정확하게 설계에 반영하는 것은 적응성을 증가시키는 것에 도움을 준다.

확장성은 SaaS가 제공자 서버에 설치 운영되고, 동시에 많은 수의 소비자들이 이를 접근 사용할 수 있음을 나타내므로, 비기능적 요구사항과 연관이 있다. 그러므로, 아키텍처 설계와 관련된 활동 320에 직접적으로 반영되었다.

• 직접적으로 반영된 활동

활동 320. SaaS 시스템 아키텍처 설계: 친구 찾기 SaaS의 아키텍처에 포함되는 컨트롤 층은 로드 밸런서와 테스크 마이그레이터를 두어 다수의 서버를 관리하는 컨트롤러와 실시간으로 상호작용할 수 있기 때문에 서비스 사용을 원하는 소비자들을 보다 효과적이고 효율적으로 관리하는 것이 가능하다. 따라서 친구 찾기 SaaS에서 설계된 아키텍처는 소비자들의 증가에 따른 확장성의 관리에도 보다 효율적으로 대처하는 것이 가능하게 된다. 친구 찾기 SaaS의 경우처럼 SaaS 시스템 아키텍처에 확장성을 고려하는 것은 보다 많은 소비자들에게 SaaS의 이용을 가능하게 해준다.

7. 평 가

본 논문에서 제안한 프로세스의 검증을 위하여 표 11에서 보는 것과 같이 여섯 가지의 평가 기준을 기반으로 비교평가하였다. 평가 기준은 [19][20]에서 제안한 항목을 기반으로 추출하였으며, 평가기준에 적합한 지원을 하는지 여부에 따라 지원(O), 불완전 지원(Δ), 지원불가(X)로 분류하였다.

'프로세스 지원'은 SaaS 개발을 위한 프로세스 모델의 지원여부를 나타내는 것으로 다음의 세부 평가 기준으로 구성된다. '프로세스의 체계적 구성'은 제안된 프로세스가 단계 및 절차에 따라 일정한 업무단위의 순서로 이루어졌는지를 평가한다. '지침의 상세화'는 제안된 지침과 활동을 다음 단계에 일관성있게 전달하기 위해 상세히 기술되었지를 평가하며 '산출물 명세'는 산출물에 핵심 항목과 양식, 예제, 관련된 가공물과 활동내역을 상세히 기술하였는지를 평가한다. 그리고 '재사용성 지원'은 SaaS 재사용성의 세 가지 부특성인 적용성, 적응성, 확장성의 반영 정도를 평가하기 위한 항목이다.

Espada의 연구에서 제시된 프로세스는 요구사항, 분석, 설계, 구현, 테스팅의 다섯 단계로 구성되며 각 단계별 지침과 산출물을 명세한다. 하지만 각 단계를 구성하는 구체적인 활동을 기술하지 않고 산출물도 간략하게 언급했을 뿐 구체적인 양식이나 예제 등을 제시하지 않았다. 그리고 적용성의 지원도 분석 단계에서 사용자들의 비즈니스 프로세스에 대한 공통성 고려의 필요성만을 언급하는데 그친다.

Kwok의 연구에서 제시된 SaaS 어플리케이션을 위한 아키텍처 프레임워크는 SaaS를 이용하는 다중 테넌트들과 이에 포함된 소비자들의 공통 요구사항과 특화된 요구사항을 지원하기 위한 여덟개의 서비스 모듈을 제공한

표 11 다른 연구와의 비교 평가

평가 기준 \ 연구		Espada	Kwok	Mietzer	본 논문의 프로세스
프로세스 지원	프로세스의 체계적 구성	O	×	×	O
	지침의 상세화	Δ	×	×	O
	산출물 명세	Δ	×	×	O
SaaS 재사용성 지원	적용성 지원	Δ	O	O	O
	적응성 지원	×	O	O	O
	확장성 지원	×	×	×	O

다. 하지만 멀티 테넌트의 증가에 따른 확장성의 필요는 언급하였음에도 제안한 아키텍처에 반영하지는 못했다.

Mietzer의 연구에서는 기존 PLE의 외부와 내부 가변성 개념을 적용한 가변성 모델링 방법을 이용하여 SaaS 어플리케이션의 가변성을 모델링을 한다. 하지만 가변성 모델링에만 초점을 맞추고 SaaS 재사용성의 부 특성 중 하나인 확장성은 다루지 못했다.

본 논문에서 제시한 프로세스는 요구사항의 분석부터 테스팅에 이르기 까지 SaaS 개발의 전 단계 및 활동들과 이에 적용되는 구체적인 지침을 제시한다. 또한 각 활동의 수행에 따른 산출물의 구체적인 작성방법도 함께 제시한다. 그리고 SaaS 재사용성을 세 가지 부 특성으로 분류하고 각 부 특성을 개발 프로세스에 반영함으로써 재사용성과의 밀접한 연관성을 보였다.

8. 결 론

CC는 하드웨어와 소프트웨어 등 다양한 형태의 자원을 서비스 형태로 제공하는 재사용 기반의 컴퓨팅 방식이다. 클라우드 서비스의 한 형태인 SaaS는 소프트웨어를 서비스 형태로 제공하여, 다양한 소비자들이 재사용하게 한다. SaaS는 제공되는 기능을 필요로 하는 다양한 조직들의 다양한 소비자들을 위해 개발되므로, 재사용성이 기존의 소프트웨어에서 언급하는 재사용성과 다른 의미를 가진다. 이렇듯, SaaS는 전통적인 소프트웨어 어플리케이션과 차이점을 가지고 있으므로, 기존의 객체지향 개발 방법론, 컴포넌트 기반 개발 기법, SOA 개발 기법으로는 SaaS를 개발하는 것이 어렵다.

따라서, 본 논문에서는 클라우드 서비스의 한 종류인 SaaS의 재사용성에 초점을 맞춘 실용적인 설계 기법을 제안한다. 먼저, 소프트웨어 기능모듈 등 특정한 자원이 다양한 소비자들에게 얼마나 효율적으로 효과적으로 사용될 수 있는가를 나타내는 척도로 "재사용성"을 정의하였으며, 세 가지 부 특성인 적용성(Applicability), 적응성(Adaptability), 확장성(Scalability)을 도출하였다. 그리고, SaaS 개발 대상을 식별하기 위하여, 여러 문헌을 분석하여 여러 개의 단위 서비스와 서비스 인터페이스로 구성된 SaaS 메타 모델을 정의하였다.

SaaS 재사용성의 세 가지 부 특성을 만족하도록 단계와 세부 활동을 유도하여, SaaS 개발 프로세스를 정의하였다. 제안된 SaaS 개발 프로세스는 크게 서비스 요구사항 정의 단계, SaaS 분석 단계, SaaS 설계 단계, SaaS 구현 단계로 구성되며, 각 단계는 복수 개의 세부 활동으로 나뉘어진다. 마지막으로, 제안된 개발 프로세스의 적용 가능성을 보여주기 위하여 '친구 찾기(Mate SaaS Service)' 도메인을 대상으로 사례연구를 실행하였다. 제시된 프로세스를 적용하면 보다 체계적이고 효과적으로 재사용성이 높은 SaaS를 개발을 유도할 수 있다.

SaaS는 제공자의 서버에서 어플리케이션이 운영된다는 특징을 가지기 때문에 현재 IT분야의 큰 이슈인 MID(Mobile Internet Device)의 물리적인 컴퓨팅 자원의 한계를 극복할 수 있는 좋은 대안으로 언급되고 있다. 따라서 향후 논문에서는 제시한 SaaS 개발 프로세스가 MID를 위한 어플리케이션의 개발에도 적용될 수 있는 보다 확장된 형태의 연구가 요구된다.

참 고 문 헌

[1] Gillett, F.E., "Future View: New Tech Ecosystems of Cloud, Cloud Services, and Cloud Computing," *Forrester Research Paper*, 2008.

[2] Turner, M., Budgen, D., and Brereton, P., "Turning Software into a Service," *IEEE Computer*, vol.36, no.10, pp.38-44, 2003.

[3] Kim, S.D., "Software Reusability," *Wiley Encyclopedia of Computer Science and Engineering*, vol. 4, pp.2679-2689, 2009.

[4] Espadas, J., Concha, D., and Molina, A., "Application Development over Software-as-a-Service platforms," *In Proceedings of Software Engineering Advances(ICSEA 2008)*, IEEE, pp.97-104, October, 2008.

[5] Kwok, T., Nguyen, T., Lam, L., "A Software as a Service with Multi-tenancy Support for an Electronic Contract Management Application," *In Proceedings of IEEE International Conference on Services Computing (SCC 2008)*, pp.179-186, 2008.

[6] Mietzner, R., Metzger, A., Leymann, F., and Pohl, K., "Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications," *In Proceedings of 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems (PESOS 2009)*, pp.18-25, 2009.

[7] Choi, S. W., Chang, S. H., and Kim, S. D., "A Systematic Methodology for Developing Component Frameworks," *In Proceedings of 7th International Conference on Fundamental Approaches to Software Engineering (FASE 2004), Lecture Notes in Computer Science 2984*, pp.359-373, 2004.

[8] Kim, S.D., Her, J.S., and Chang, S.H., "A Theoretical Foundation of Variability in Component-Based Development," *Information and Software Technology(IST)*, vol.47, pp.663-673, July, 2005.

[9] *Software Engineering - Product Quality - Part 1: Quality Model. ISO/IEC 9126-1*, June, 2001.

[10] Laplante, P.A., Zhang, J., and Voas, J., "What's in a Name? Distinguishing between SaaS and SOA," *IT Professional*, vol.10, no.3, pp.46-50, May/June 2008.

[11] Her, J.S., La, H.J., and Kim, S.D., "A Formal

Approach to Devising a Practical Method for Modeling Reusable Service," *In Proceedings of 2008 IEEE International Conference on e-Business Engineering (ICEBE 2008)*, pp.221-228, October 22-24, 2008.

[12] Chang, S.H. and Kim, S.D., "A SOAD Approach to Developing Adaptable Services," *In Proceedings of IEEE International Conference on Services Computing (SCC 2007)*, pp.713-714, 2007.

[13] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley Professional, 1994.

[14] Gomma, H., *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*, Addison Wesley Professional, 2004.

[15] Chong, F., Carraro G., and Wolter, R., "Multi-Tenant Data Architecture," MSDN Architecture Center, 2006. http://msdn2.microsoft.com/

[16] Rozanski, N. and Woods, E., *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*, Addison-Wesley, 2005.

[17] Jimenez-Peris, R., Patiño-Martinez, M., Kemme, B., Perez-Sorrosal, F., and Serrano, D., "A System of Architectural Patterns for Scalable, Consistent and Highly Available Mult-Tier Service-Oriented Infrastructure," *Architecting Dependable Systems VI, Lecture Notes in Computer Science 5835*, pp. 1-23, 2009.

[18] Hennicker, R. and Koch, N., "Modeling the User Interface of Web Applications with UML," *In Proceedings of Workshop of the pUML-Group held together with the «UML» 2001 on Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists*, pp. 158-172, 2001.

[19] IEEE Computer Society and ACM, Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE Computer Socity, 2004.

[20] Pressman, R., Software Engineering: A Practitioner's Approach 6th edition, McGraw-Hill, 2005.

이 정 우

2009년 숭실대학교 컴퓨터학과 학사. 2009년~현재 숭실대학교 컴퓨터학과 석사과정. 관심분야는 객체지향 S/W공학, 서비스 지향 아키텍처(SOA), 클라우드 컴퓨팅(Cloud Computing)

라 현 정

2003년 경희대학교 우주과학과 이학사 2006년 숭실대학교 컴퓨터학과 공학석사 2006년~현재 숭실대학교 컴퓨터학과 박사수료. 관심분야는 서비스 지향 아키텍처(SOA), 컴포넌트 기반 개발(CBD)

김 수 동

1984년 Northeast Missouri State University 전산학 학사. 1988년/1991년 The University of Iowa 전산학 석사/박사 1991년~1993년 한국통신 연구개발단 선임연구원. 1994년~1995년 현대전자 소프트웨어연구소 책임연구원. 1995년 9월~현재 숭실대학교 컴퓨터학부 교수. 관심분야는 서비스 지향 아키텍처(SOA), 클라우드 컴퓨팅(Cloud Computing), 모바일 서비스(Mobile Service), 객체지향 S/W공학, 컴포넌트 기반 개발(CBD), 소프트웨어 아키텍처