서비스 시스템 구축을 위한 효율적 아키텍처 설계

(A Design of Effective Architecture for Constructing Services Systems)

라 현 정 † 김 성 안 †† 김 수 동 †††
(Hyun Jung La) (Sung Ahn Kim) (Soo Dong Kim)

요 약 서비스 시스템은 서비스를 시스템 개발의 기본 구성 단위로 이용하는 시스템으로, 새로운 응용 분야로 등장하고 있다. 서비스 시스템은 웹 환경 외에 움직이는 여러 장치들이 유선 또는 무선으로 연결되어 있는 유비쿼터스 환경에서도 운영되며, 사용자가 필요로 하는 서비스를 제공한다. 서비스는 사용자가 수행하는 작업을 수월하게 하기 위해 제공 받는 기능의 단위로, 사용자의 요구뿐만 아니라 사용자 주변의 여러 컨텍스트 정보를 고려해서 가장 적절한 서비스가 실행되어야 한다. 그러므로, 사용자의 요구에 충분히 만족하는 서비스를 제공하기 위해서 서비스 시스템은 컨텍스트 인지와 컨텍스트에 맞게 서비스를 적응시키는 것이 매우 중요하다. 이러한 특성 때문에 동일한 서비스는 사용자의 컨텍스트에 따라 다수의 사용자에게 다르게 제공될 수 있으며 이는 서비스 시스템의 가변성이 생기는 주요한 원인이 된다. 컨텍스트 인지와 관련된 많은 연구에서는 컨텍스트에 따라 시스템을 적응시키는 것을 다소 정형화되지 않은 임시적인 방법을 이용해서 해결하려고 시도했었다. 본 논문에서는 컨텍스트에 따라 다양하게 서비스가 제공되는 것을 가변성으로 간주하고, 가변성을 체계적인 방법으로 다루기 위해서 대표적인 재사용 방법론 중의 하나의 프로덕트 라인 공학 개념을 적용함으로써 컨텍스트 기반의 동적으로 적응 가능한 아키텍처를 제안한다.

키워드 : 소프트웨어 아키텍처, 서비스, 프로덕트 라인 공학

Abstract Services system which has been emerging as a new way of application development utilizes services as fundamental units for developing a system. Services system can offer services within web environment as well as the ubiquitous environment where mobile devices are connected to wired or wireless network. In order to provide the functionality that meets users' requirements, the most appropriate service should be selected among candidate services by taking requests and context information into account. Therefore, it is important that the service system should provide services to users by dynamically adapting to users' requirements and context information. Since different users request same functionality with different context information, one service can be differently offered to users so that variability can happen to the service systems. Most researches on context-aware systems have a tendency to solve dynamic adaptation by using more or less ad hoc manner. In this paper, we consider various types of services which are performed according to context information as variability and propose adaptable architecture by applying concepts of product line architecture in order to deal with variabilities systematically.

Key words : Software Architecture, Service, Product Line Engineering

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음

† 학생회원 : 숭실대학교 컴퓨터학과
hjla@otlab.ssu.ac.kr
†† 정 회 원 : 숭실대학교 컴퓨터학과
sakim@otlab.ssu.ac.kr
††† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@ssu.ac.kr

논문접수 : 2006년 9월 18일
심사완료 : 2008년 2월 1일

정보과학회논문지: 소프트웨어 및 응용 제35권 제3호(2008.3)

1. 서 론

새로운 응용분야로 등장하고 있는 서비스 시스템은 시스템 개발을 위한 기본 구성 단위로 서비스를 사용하는 시스템이다[1]. 서비스 시스템은 움직이는 여러 정치들이 유선 또는 무선으로 연결되어 있는 유비쿼터스 환경에서 운영되며, 사용자가 필요로 하는 서비스를 제공한다. 서비스는 사용자가 수행하는 작업을 수월하게 하기 위해 제공 받는 기능 단위로, 사용자의 요구뿐만 아니라 사용자 주변의 여러 컨텍스트 정보를 고려해서 가장 적절한 서비스가 실행되어야 한다. 그러므로, 사용자의 요구에 충분히 만족하는 서비스를 제공하기 위해서 서비스 시스템은 컨텍스트 인지와 컨텍스트에 맞게 서비스를 적응시키는 것이 매우 중요하다. 이러한 특성 때문에 동일한 서비스는 사용자의 컨텍스트에 따라 다수의 사용자에게 다르게 제공될 수 있으며 이는 서비스 시스템의 가변성이 생기는 주요한 원인이 된다.

프로덕트 라인 공학(Product Line Engineering, PLE)은 한 프로덕트 라인에 속하는 여러 어플리케이션 멤버들이 재사용할 수 있는 핵심자산을 이용하는 대표적인 소프트웨어 재사용 방법이다[2]. PLE에서의 재사용 단위인 핵심자산은 어플리케이션을 만드는데 기초가 되는 모든 자산을 의미하고, 프로덕트 라인 아키텍처(Product Line Architecture, PLA), 컴포넌트, 결정 모델이 이에 속하며, 프로덕트 라인에 속한 모든 어플리케이션들의 공통성과 가변성을 분석/설계하여 만든다. 핵심 자산 중에 PLA는 프로덕트 라인의 전체 구조를 표현하는 핵심 자산의 주요 요소이다.

기존의 컨텍스트 인지 시스템에 관한 많은 연구에서는 정형화되지 않은 임시적인 방식으로 동적 적응을 해결하려고 시도했었다. 본 논문에서는 컨텍스트에 따라 다양하게 서비스가 제공되는 것을 가변성으로 간주하고, 가변성을 체계적인 방법으로 다루기 위해서 대표적인 재사용 방법론 중의 하나의 프로덕트 라인 공학 개념을 적용함으로써 컨텍스트 기반의 동적으로 적응 가능한 아키텍처를 제안한다.

본 논문에서는 먼저 서비스 시스템의 아키텍처 요구사항과 아키텍처 드라이버를 기술한다. 다음으로 아키텍처 스타일을 선택하고 컴포넌트를 할당시킴으로써 아키텍처를 설계한 후 어플리케이션에 따른 가변성을 해결하는데 사용되는 의사결정 모델을 제안한다. 서비스 시스템에서 발생할 수 있는 가변성을 식별하는데 중점을 두고 의사결정 모델을 이용함으로써 가변성을 다루기 위한 체계적인 방법을 제안한다. 마지막으로 제안된 아키텍처의 적용성을 보이기 위해 사례연구를 수행한다.

2. 관련연구

Tao Gu와 동료들이 제안한 SOCAM(Service-Oriented Context-Aware Middleware)은 컨텍스트를 인지하는 서비스를 구축하기 위한 아키텍처이다[3]. SOCAM은 센서로부터 컨텍스트 정보를 획득하는 컨텍스트 감지 계층, 인지된 컨텍스트 정보를 해석/추론하는 컨텍스트 미들웨어 계층, 분석된 컨텍스트 결과에 따라 동적으로 적절한 서비스를 제공하는 컨텍스트 어플리케이션 계층으로 구성되어 있다. 감지된 컨텍스트 정보는 OWL과 사용자가 정의한 규칙을 이용하여 표현됨으로써 시맨틱 정보을 포함하여 기술하며, 추론도 가능하게 하였다. 본 논문은 온톨로지와 규칙을 이용하여 적절한 서비스를 제공하지만, 컨텍스트 정보에 따라 변해야 하는 서비스 기능이 다양해질수록 이를 해결하기 위한 체계적인 기법이 필요하다.

Hayes는 적응 가능한 지능형 시스템을 지원하기 위한 도메인 종속적인 소프트웨어 아키텍처(Domain-Specific Software Architecture, DSSA)를 제안한다[4]. DSSA는 환경에서 정보를 감지하는 물리 계층과 정보를 기반으로 사용자의 환경을 추론하며 재사용 가능한 컴포넌트와 어플리케이션에 국한된 컴포넌트를 이용하여 자동적으로 컴포넌트들의 구성도를 만드는 인지 계층으로 구성된다. 인지 계층에서 사용되는 컴포넌트도 재사용성을 높이기 위해 주제 도메인, 메소드, 태스크 기준으로 분류하였다. 그리고, 실행 시간에도 컴포넌트들의 구성을 동적으로 조정시키는 도구를 제공한다. 그러나, 아키텍처의 동적인 구성 변경은 재사용 가능한 컴포넌트를 이용하여 아키텍처 관점에서의 가변성은 해결될 수 있지만, 이 외에 컴포넌트의 내에서 발생하는 가변성을 해결 할 수 있는 체계적인 방법이 필요하다.

Garlan이 제안한 Rainbow 프레임웍은 자동적으로 적응될 수 있는 소프트웨어 시스템의 재사용 가능한 인프라스트럭처를 제안한다[5]. 여러 시스템에 의해 재사용되는 적응형 인프라스트럭처는 시스템에 접근하는 인터페이스를 제어하는 시스템 계층 인프라스트럭처, 적응 행위를 결정하고 이를 실행하는 아키텍처 계층 인프라스트럭처와 이 두 개의 계층 간의 차이를 매핑시켜주는 전환 인프라스트럭처로 구성된다. 적응에 사용되는 지식은 특정 시스템의 요구에 맞게 인프라스트럭처를 맞추는데 사용된다. 이 연구는 어플리케이션간의 공통성은 아키텍처 스타일을 이용해서 해결하려고 하였지만, 가변성은 고려하지 않았기 때문에 가변성을 다룸으로써 메커니즘을 향상시킬 수 있을 것이다. 또한, 제안된 프레임웍은 자원과 사용자의 변경에 대한 적응은 제공하지만 사용자의 컨텍스트에 대한 동적인 적응은 다루지 않

는다. 그리고, 또 다른 Garlan의 연구에서는 개인 작업을 보조하는 시스템 아키텍처인 RADAR를 제안하였다. 제안된 아키텍처는 새 기능들을 쉽게 플러그인함으로써 아키텍처를 확장할 수 있으며 사용자에 따른 적절한 서비스를 제공한다 [6]. 하위 계층에서는 사용자와의 상호작용을 담당하며, 상위 계층에서는 플러그인 기법을 이용하여 사용자의 작업을 도와주는 서비스를 실행한다. 그러나, 본 논문에서는 입력에 따라 시스템을 동적으로 적응시키는 메커니즘을 구체적으로 기술하지 않았다.

3. 기반연구

3.1 서비스 시스템

기존의 소프트웨어 시스템과는 달리, 서비스 시스템은 서비스가 시스템 개발을 위한 기본 단위로 활용되는 새롭게 부각되고 있는 컴퓨팅 패러다임이다[7]. 서비스 시스템의 주요 목표는 사용자의 필요에 따른 적절한 서비스를 제공함으로써 삶을 보조하는 것이다.

서비스 시스템을 이루는 서비스에는 제공자, 구독자, 사용자의 세 참여자가 관련된다[1]. 서비스 제공자는 서비스를 구현하고 서비스에 대한 명세서를 제공하며 기술적/비즈니스적인 지원을 하는 기관이다. 서비스 구독자는 제공자로부터 서비스를 구독하는 개인이나 기관이며, 서비스 사용자는 서비스의 목표에 해당하는 서비스가 제공하는 기능성의 대상인 사람이다. 서비스는 관련된 참여자에게 의미 있는 가치를 줄 수 있는 단위로 정의된다[8]. 따라서 서비스는 소프트웨어 관점에서 기능성을 제공하는 단위인 함수나 오퍼레이션과는 달리 서비스 참여자에게 가치를 줄 수 있는 단위이다.

그림 1은 서비스 시스템이 사용자에게 서비스를 제공하는 과정을 보여주는 서비스 시스템의 컴퓨팅 모델이다. 서비스 시스템은 사용자로부터의 입력/감지된 데이타, 사전에 등록된 사용자 프로파일에 기반해 서비스가 요구되는 컨텍스트 정보를 획득한다. 획득된 컨텍스트 정보와 분석규칙을 이용해 현재 사용자가 처한 상황을 판단한다. 시스템은 미리 정의된 규칙에 기반해서 사용자의 상황에 따른 적절한 행동을 결정하고 이를 수행한다. 서비스 시스템은 사용자에게 적절한 서비스를 제공하기 위하여 사용자나 환경에 영향을 줄 수 있는 행동을 수행한다. 서비스의 수행 결과는 차후의 서비스에도 피드백을 제공한다.

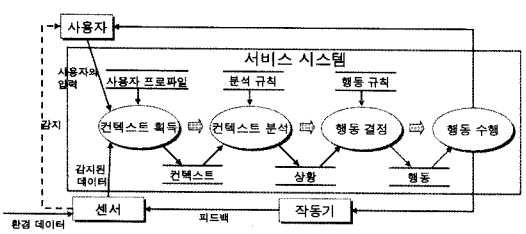

그림 1 서비스 시스템 컴퓨팅 모델

3.2 가변성 관리(Variability Management)

가변성(Variability)은 동일한 도메인에 속한 여러 어플리케이션에 따라 변할 수 있는 공통성 내에 존재하는 성질을 의미한다[2]. 많은 연구자들은 소프트웨어 위기에 대응하기 위해 재사용 방법론에 관심을 두었고, 컴포넌트 기반 개발(Component-based development, CBD)과 프로덕트 라인 공학(Product line engineering, PLE)에서 재사용성을 향상시키기 위해 가변성 개념을 고안하였다. CBD는 재사용 단위인 컴포넌트를 조립하여 효율적으로 어플리케이션을 개발하는 방법론이며, PLE는 재사용 단위인 미리 정의된 핵심 자산을 목적에 맞게 인스턴시에이션(Instantiation)하여 목표 어플리케이션을 개발하는 방법론이다.

가변성은 가변점(Variation Point)과 가변치(Variant)로 표현될 수 있으며, 가변점은 어플리케이션간에 서로 다른 부분인 가변성이 발생하는 지점이고, 이런 가변치는 가변점을 어플리케이션 고유의 요구사항을 위해 설정할 수 있는 값이다. 가변점에 가변치를 설정함으로써, 동일한 도메인에 속한 여러 어플리케이션 간의 가변성을 해결한다. 가변성의 타입은 가변점의 종류에 따라, 속성(Attribute) 가변성, 로직(Logic) 가변성, 워크플로우(Workflow) 가변성, 영속성(Persistency) 가변성, 인터페이스(Interface) 가변성으로 분류할 수 있다[9].

CBD와 PLE에서는 재사용 단위인 컴포넌트나 핵심자산을 개발하기 위해 도메인 공학 단계에서 공통성과 가변성(Commonality and Variability)을 분석하고 설계한다. 이 단계에서는 어플리케이션 간의 공통적인 기능 또는 휘처를 식별하고, 가변성이 어떻게 존재하는지를 찾아내어 그 가변성을 해결할 수 있는 방법을 설계한다. 그리고, 재사용 단위를 이용하여 목표 어플리케이션을 만드는 단계인 어플리케이션 공학 단계에서 설계된 가변성에 특정 가변치가 설정됨으로써 가변성이 해결된다.

3.3 프로덕트 라인 아키텍처

프로덕트 라인 아키텍처(Product Line Architecture, PLA)는 프로덕트 라인에 포함된 어플리케이션들이 공유 가능한 범용적인 시스템의 구조를 표현한다[2]. PLA는 프로덕트 라인에 포함된 다수의 어플리케이션에서 사용되므로 시스템들 간에 공통적으로 존재하는 공통성과 시스템과 시스템에 따라 변하는 가변성을 모두 다뤄야 한다. 이러한 특성은 전통적인 소프트웨어 아키텍처와 PLA의 가장 큰 차이점이다. 따라서 프로덕트 라인에서 어플리케이션 간의 재사용성을 높이기 위해서는 가변성을 다루는 것이 필수적인 요구조건이다. PLA에

는 아키텍처에 영향을 주는 요소인 아키텍처 드라이버의 가변성, 아키텍처를 정의하는 스타일의 가변성, 아키텍처를 구성하는 컴포넌트 간의 관계에서의 가변성, 세부 컴포넌트가 가지는 가변성이 존재한다[9].

다수의 연구[2,11,12]에서 제안한 것과 같이 PLA는 다음의 프로세스를 수행하면서 설계된다. 먼저 요구사항으로부터 아키텍처에 영향을 주는 요소인 아키텍처 드라이버를 식별한다. 식별된 아키텍처 드라이버로부터 아키텍처 스타일을 유도함으로써 개념적 아키텍처를 구축한다. 마지막으로 추상 컴포넌트를 구체적 컴포넌트로 분할하면서 개념적 아키텍처를 정제한다. PLA 설계 과정에서 아키텍처 의사결정 모델이 만들어진다. 의사결정 모델은 가변점, 가변치, 가변점에 알맞은 가변치를 채워서 인스턴시에이션(Instantiation)하기 위한 작업을 포함하며, PLA가 인스턴시에이션되어 특정 어플리케이션의 아키텍처를 생성하는데 사용된다.

4. 서비스 시스템의 아키텍처 요구사항과 드라이버

아키텍처 요구사항은 아키텍처를 설계하는데 있어서 필수 요소이다. 본 장에서는 서비스 시스템의 고유한 특징으로부터 아키텍처 요구사항을 명시하고 요구사항으로부터 아키텍처적 의사결정에 영향을 미치는 아키텍처 드라이버를 유도한다. 서비스 시스템은 기존의 IT 시스템과 비교해 눈에 띄는 차이점을 가지기 때문에 IT 시스템과 구별되는 서비스 시스템만의 특징에 기반한 아키텍처 요구사항을 작성한다.

서비스 시스템은 사용자 각자의 요구사항을 충실히 만족시킬 수 있는 서비스를 지원해야 하므로 서비스는 사용자 개개인에 맞게 개인화되어 제공되어야 한다. 사용자의 개입 없이 개인에게 알맞은 서비스를 제공하기 위해서 시스템은 서비스가 필요로 하는 사용자의 컨텍스트를 인지하고 사용자의 응답을 이해함으로써 사용자의 필요를 알 수 있어야 한다. 이를 위해, 사용자의 컨텍스트 정보를 모으고 이해하는 것은 사용자의 요구에 알맞은 서비스를 제공하기 위한 서비스 시스템의 주요 기능이다. 분석된 컨텍스트 정보에 기반한 알맞은 서비스를 제공하기 위해서 시스템은 동적으로 구성된 서비스를 제공할 수 있다. 시스템은 지속적인 서비스를 제공하기 위하여 운용중인 상태에서 새롭게 배포된 서비스가 갱신된 서비스를 감지할 수 있어야 한다. 시스템은 사용자에게 적절한 서비스를 제공하기 위하여 다양한 시스템과 장치와 상호작용한다. 사용자의 위치 변경에도 컨텍스트 정보는 시스템과 시스템 사이에서 단절되지 않고 지속적으로 관리되어야 하고, 다른 시스템이 가지고 있는 이전의 컨텍스트 정보와 같이 획득할 수 없는 컨텍스트 정보를 획득하고 공유하기 위해서는 사용자가 가지고 있는 다양한 장치와도 상호작용하여야 한다. 서비스 시스템은 다수의 사용자에게 동시에 서비스를 제공할 수 있으므로 동일한 서비스라 할지라도 동시에 사용자 개인에 특화된 서비스를 제공해야 한다.

위의 요구사항에 기반해 표 1과 같이 아키텍처 드라이버를 도출한다. 적응 가능한 동적 재구성은 시스템이 사용자에 맞는 서비스를 제공하기 위해서 유연하고 확장 가능해야 하는 요구사항으로부터 도출되었다. 컨텍스트 인지는 시스템이 사용자의 직접적인 입력 중심적이던 기존의 IT 시스템과는 다르게 사용자의 주변 컨텍스트를 이해하는 것이 핵심적이라는 요구사항으로부터 도출되었다. 다수의 사용자에게 개인화된 서비스 제공은 하나의 서비스라 할지라도 동시의 다수의 특화가 필요하다는 요구사항으로부터 도출되었으며, 매우 높은 분산 환경은 시스템이 분산된 다양한 시스템과 장치와 상호작용하여 단절되지 않은 서비스를 제공하여야 하는 요구사항으로부터 도출되었다.

5. 서비스 시스템의 아키텍처 스타일

하나의 아키텍처 드라이버는 여러 아키텍처 스타일을 이용해 아키텍처에 해결될 수 있으므로, 목표로 하는 어플리케이션의 요구사항과 특성을 고려함으로써 해당 드라이버에 가장 적절한 스타일을 선택해야 한다. 그러므로, 본 장에서는 4장에서 유도한 아키텍처 드라이버를 만족시키는 적절한 아키텍처 스타일을 선택하고, 어떻게 선택한 스타일이 아키텍처 드라이버를 만족시키는지에 대해 평가한다.

표 1 아카텍처 드라이버 명세

드라이버	설명
적응 가능한 동적 재구성	서비스를 제공하기 위해서 유연하게 구성하고 런타임에 새로운 기능을 추가할 수 있는 시스템의 능력.
컨텍스트 인지	사용자에게 적절한 서비스를 제공하기 위해 사용자의 컨텍스트를 획득하고 분석할 수 있는 시스템의 능력.
다수의 사용자에게 개인화된 서비스 제공	하나의 서비스라 할지라도 다수의 사용자에게 동시에 개인화된 서비스를 제공할 수 있는 시스템의 능력.
매우 높은 분산 환경	움직이는 사용자 중심적으로 다수의 시스템과 장치가 상호작용하여 정보의 접근, 의사소통, 비즈니스 트랜잭션을 처리할 수 있는 시스템의 능력.

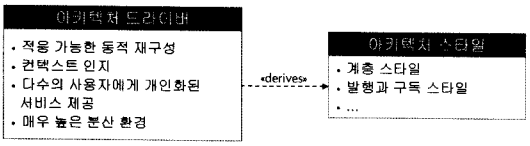

그림 2 선택된 아키텍처 스타일

아키텍처에 관한 표준 중 하나인 IEEE 1471에서는 아키텍처 뷰(Architecture View)에 따라 아키텍처를 설계/명세하는 것을 권장한다[13]. 본 논문에서도 역시 IEEE 1471의 권장사항을 수용하여 2가지 아키텍처 뷰를 이용하여 아키텍처를 설계한다. 그림 2와 같이 4가지의 아키텍처 드라이버를 만족시키기 위해 모듈(Module) 아키텍처 뷰를 위한 계층 스타일을 선택하고, 행위(Behavior) 아키텍처 뷰를 위한 발행과 구독 스타일을 선택하였다. 그러므로, 모듈 아키텍처 뷰를 이용하여 시스템의 정적인 구조를 보여주고, 행위 뷰를 통해 시스템의 동적인 구조를 보여줄 수 있게 된다.

그러나, 선택된 아키텍처 스타일은 본 논문에서 제안하는 아키텍처를 위해 선택되었기 때문에, 이 스타일 외에도 특정 서비스 시스템의 요구사항에 따라 아키텍처 스타일이 추가될 수 있다.

• 계층 스타일(Layered Style)

서비스 시스템은 고정되지 않은 수의 장치로부터 사용자 입력과 컨텍스트 정보를 받고, 그에 맞게 서비스 시스템에 있는 서비스 컴포넌트들이 유동적으로 상호작용을 하여 사용자에게 정확한 서비스를 제공한다. 즉, 서비스를 사용자에게 제공하기 위해서 필요한 장치와 서비스 컴포넌트의 종류와 수는 컨텍스트에 따라 유동적으로 서비스 시스템에 속하게 되고, 이런 시스템의 변화는 시스템 전반적으로 부정적인 영향을 미칠 수 있는 큰 요인이 된다. 그러므로, 유동적으로 서비스 시스템에 포함되는 장치와 서비스 컴포넌트로 인해 시스템 전반적으로 미치는 영향을 최소화시켜야 한다.

아키텍처가 여러 계층으로 구성된다면 한 계층에 컴포넌트가 추가, 삭제, 변경되더라도 다른 계층에 미치는 영향을 최소화시키기 때문에 시스템은 높은 수정가능성과 이식성을 가지게 된다[13]. 그러므로, 본 논문에서 제안하는 아키텍처는 장치와 서비스 컴포넌트를 각각 다른 계층에서 처리하는 계층 스타일을 채택하여 4장에서 도출한 모든 아키텍처 드라이버를 해결한다. 계층 스타일을 적용함으로써 새로운 장치나 서비스 컴포넌트가 추가 또는 삭제되거나 변경이 되어도 시스템은 계속해서 안정적인 상태를 유지할 수 있다.

제안한 아키텍처는 임베디드 시스템 아키텍처의 대표적인 계층구조인 [15]을 기반으로 하드웨어 제어 계층, 시스템 소프트웨어 계층, 어플리케이션 소프트웨어 계층으로 구성된다. 하드웨어 제어 계층에서는 시스템 소프트웨어 계층에 데이타를 전달하고 사용자의 서비스 요청에 대한 처리 결과를 수행하기 위한 다양한 장치를 관리한다. 시스템 소프트웨어 계층은 하드웨어 계층과 어플리케이션 소프트웨어 계층 사이의 중개자 역할을 하며, 사용자의 요청 또는 컨텍스트 정보에 따라 적절한 서비스를 제공할 수 있도록 한다. 어플리케이션 소프트웨어 계층에는 사용자가 필요로 하는 서비스를 구현한 어플리케이션 서비스 컴포넌트가 존재한다. 그러므로, 이 계층에는 서비스 시스템이 제공하는 서비스 종류에 따라 엔터테인먼트 서비스, 응급 처리 서비스와 같이 직접 사용자가에게 제공될 서비스가 구현되고 관리된다.

• 발행과 구독 스타일(Publish-and-Subscribe Style)

서비스 시스템은 사용자의 요청과 컨텍스트 정보에 따라서 서비스 시스템은 다수의 서비스 컴포넌트를 합쳐서 동적으로 서비스를 제공해야 한다. 이러한 요구사항을 만족시키기 위해서 서비스 시스템에 있는 모든 컴포넌트를 관리하는 역할을 가진 컴포넌트 관리자가 시스템 내에 있는 모든 서비스 컴포넌트들의 상태를 지속적으로 유지/관리하여, 컨텍스트에 맞게 적절한 서비스 컴포넌트들을 선택하고 서비스 컴포넌트들을 형상을 만들며, 이들 간의 상호작용 관계로 관리해야 한다. 그러므로, 본 논문에서는 발행과 구독 스타일을 채택하여[14], '적응 가능한 동적 재구성'과 '다수의 사용자에게 개인화된 서비스 제공'의 아키텍처 드라이버를 해결한다.

서비스 컴포넌트가 어플리케이션 소프트웨어 계층에 새로 추가, 수정, 삭제되면 이러한 컴포넌트들의 상태는 서비스 시스템에 등록(발행)되어야 서비스 시스템의 다른 컴포넌트 또는 다른 서비스 시스템에서 사용될 수 있다. 특정 기능을 구현한 서비스 컴포넌트는 구독자에 의해 사용될 수 있는데, 구독자는 미리 정해놓은 특정 조건이 만족되면 해당 기능의 서비스 컴포넌트가 실행된다. 이런 작업은 서비스 컴포넌트의 상태를 관리하는 특정 컴포넌트에 의해 수행될 수 있다. 발행과 구독 스타일을 실체화 함으로써 서비스 시스템은 배포 시점과 실행 시간에 컨텍스트 정보에 따라 서비스 컴포넌트들을 동적으로 구성하고 적절한 서비스를 사용자에게 제공할 수 있다.

여기서 선택한 계층 스타일과 발행과 구독 스타일 이외에도 어플리케이션의 특성에 따라 추가적인 스타일이 채택될 수 있다. 예를 들어, 파이프와 필터 스타일(Pipe-and-Filter Style)은 컴포넌트들의 일련의 작업이 순차적으로 수행되어야 하는 경우에 적용될 수 있다.

6. 서비스 시스템 아키텍처

본 장에서는 5장에서 제안된 아키텍처 스타일에 구체

적인 컴포넌트를 할당하는 과정을 거쳐 추상적인 아키텍처를 인스턴스화하여 서비스 시스템의 구체적인 아키텍처를 나타낸다. 그러므로, 모든 서비스 시스템에 적용될 수 있는 범용적인 아키텍처를 먼저 보여주고, 범용 아키텍처를 의사 결정 모델을 이용하여 특정 어플리케이션을 인스턴스화 하는 방법을 제안한다.

6.1 서비스 시스템을 위한 범용 아키텍처

본 장에서는 5장에서 도출된 추상적인 컴포넌트들을 좀 더 구체적인 컴포넌트로 분해함으로써 선택된 아키텍처 스타일을 인스턴스화 한다. 그림 3은 계층 스타일과 발행과 구독 스타일이 적용된 제안된 아키텍처와 각각의 계층에 속한 컴포넌트를 보여준다. 하드웨어 제어 계층에는 서비스 시스템에서 사용될 수 있는 다양한 장치들을 관리하는 장치관리자가 있으며, 시스템 소프트웨어 계층에는 컨텍스트 정보를 관리하는 컨텍스트 관리자, 분석된 컨텍스트 정보에 따라 서비스 컴포넌트를 결정하는 서비스 결정 관리자, 선택된 서비스 컴포넌트간의 상호작용을 고려하여 컴포넌트 구성도를 만드는 서비스 구성기가 있고, 어플리케이션 서비스 계층에는 기능을 구현한 서비스 컴포넌트들이 존재한다. 서비스 시스템 아키텍처는 사용자의 요구에 맞는 특정 서비스를 수행하기 위한 계층간에 데이타 흐름이 존재한다.

• 장치 관리자

장치 관리자는 네트워크 상에 존재하는 여러 종류의 장치와 장치로부터 전달된 데이타를 관리하는 소프트웨어 컴포넌트이다. 한 서비스 시스템의 네트워크에는 다수의 장치들이 유선 또는 무선으로 연결되어 있기 때문에, 서비스 시스템은 장치의 상태를 지속적으로 알아야 한다. 그러므로, 장치 관리자는 장치들을 저장소에 등록하고 관리하며, 등록된 장치로부터 데이타를 수집하는 작업을 수행한다. 그리고, 장치에서 수집되거나 사용될 데이타 형식과 시스템에서 인식할 수 있는 형태가 다르기 때문에, 장치와 시스템 간의 데이타 형식을 알맞게 변환시켜주는 역할도 한다.

장치관리자에는 입력장치와 작동장치가 등록된다. 입력장치는 사용자로부터 필요한 정보(사용자가 직접 입력한 정보, 특정 센서로부터 감지된 컨텍스트 정보 등)를 획득하는 역할을 담당한다. 그리고, 작동장치는 서비스의 결과를 사용자에게 알리거나 서비스가 수행된 결과에 따라 특정 행위를 수행하는 역할을 담당한다.

• 컨텍스트 관리자

컨텍스트 관리자는 컨텍스트 정보를 관리하고 검증하는 역할을 담당하는 소프트웨어 컴포넌트이다. 사용자가 특정 서비스를 요청하거나 컨텍스트 변화에 따라 자동적으로 특정 서비스가 트리거될 때, 서비스 시스템은 적절한 서비스를 사용자에게 제공하기 위해 컨텍스트 정보를 사용하고, 심지어 동일한 서비스라 할지라도 컨텍스트 정보에 따라 다르게 제공될 수 있다. 따라서 사용자에게 최적의 서비스를 제공하기 위해서 컨텍스트 정보는 체계적으로 관리되어야 한다.

컨텍스트 관리자의 역할은 크게 세 가지로 분류할 수 있다. 첫째, 서비스가 기능을 수행할 때 저장소에 있는 모든 컨텍스트 정보를 활용하는 것이 아니라 서비스와 사용자에 관련된 컨텍스트 정보만을 선택한다. 그러므로, 컨텍스트 관리자는 서비스에 관련된 컨텍스트 정보에 대해서 알아야 하며, 이에 관한 데이타를 장치관리자에게 전달하여 서비스 제공에 필요한 정보만을 수집한다. 둘째, 컨텍스트 관리자는 필요한 모든 정보가 수집되었는지와 수집된 정보가 정확한지를 검증해야 한다. 마지막으로, 현재 사용자가 처한 상황을 결정하기 위해서 컨텍스트 정보를 분석하는 것이다. 수집된 정보를 이용해서 특정 컨텍스트를 분석하기 위해서 온톨로지의 추론 방법을 이용하거나 사용자가 직접 정한 규칙을 이용할 수 있다. 예를 들어, 광도, 온도 등의 수집된 정보

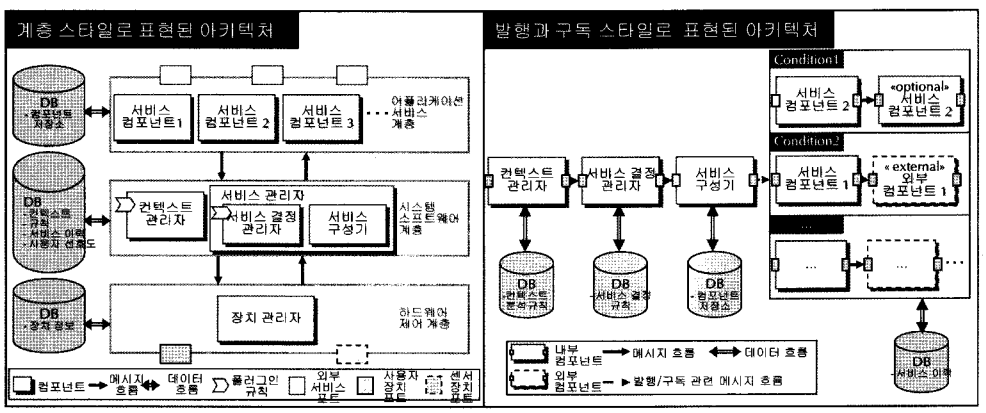

그림 3 서비스 시스템의 프로덕트 라인 아키텍처

와 상황을 결정하기 위한 규칙을 이용하여 날씨가 흐린지 맑은지에 대해 컨텍스트를 분석한다.

• 서비스 관리자

서비스 관리자는 모든 시스템에 있는 어플리케이션 서비스 컴포넌트를 관리하는 책임을 가지고 있으며, 서비스 결정 관리자와 서비스 구성기(Configurator)로 이루어진다.

서비스 결정 관리자는 특정 서비스 기능을 수행하기 위한 정보와 적절한 규칙을 이용해 가장 적절한 서비스를 결정한다. 사용자 개개인의 선호나 시스템의 특성에 따라 사용되는 규칙이 다를 수 있기 때문에 사용자에게 최적의 서비스를 제공하기 위해서 연관된 규칙이 서비스 관리자에 플러그인될 수 있다. 그리고, 서비스 결정 관리자는 다수의 사용자가 서비스를 요청할 때 어떤 사용자에게 서비스를 제공해야 하는지를 결정한다. 서비스 구성기는 서비스 결정 관리자에 의해 선택된 서비스를 제공하기 위해 서비스 시스템에 있는 서비스 컴포넌트의 상태를 파악하고, 사용 가능한 컴포넌트들을 배열하며 서비스 컴포넌트들 간의 워크플로우를 정의하는 역할을 한다. 서비스 구성기가 이러한 작업을 수행하기 위해서 서비스 시스템에 발행된 모든 서비스 컴포넌트와 서비스 컴포넌트의 상태에 대해 알아야 한다. 예를 들어, 컨텍스트 관리자가 컨텍스트를 분석하여 응급 상황이라는 결론을 내렸다면, 서비스 결정 관리자는 응급 상황을 처리하기 위해 가족들에게 연락해주는 서비스 컴포넌트와 구급차를 불러주는 기능을 하는 컴포넌트를 결정하고, 서비스 구성기는 이 결정된 두 컴포넌트의 상태를 파악하고 두 컴포넌트 간의 워크플로우를 정의한다.

• 서비스 컴포넌트

서비스 컴포넌트는 서비스를 구성하는 사용자가 인지할 수 있는 작업의 단위이며 하나의 컴포넌트로 구현되는 요소이다. 사용자에게 완성된 서비스를 제공하기 위해서는 일련의 서비스 컴포넌트는 미리 정의된 순서에 따라 작업을 수행한다. 서비스 컴포넌트는 특정 서비스 시스템에 존재하거나 외부의 다른 시스템에 위치할 수 있으므로 서비스 구성기에 의해 활용되려면 발행되어 등록되어야 한다.

6.2 서비스 시스템의 가변성과 가변성 해결 방법

본 절에서는 그림 3에서 명시된 아키텍처에 발생할 수 있는 가변성을 알아보고, 그 가변성을 처리하여 특정 어플리케이션의 아키텍처를 특화할 수 있는 방법을 위한 서비스 시스템의 의사 결정 모델(Decision Model)을 제안한다. 가변적인 다루기 위한 주요 수단인 의사결정 모델을 다루기에 앞서 아키텍처에서 발생 가능한 가변성을 다음과 같이 식별한다.

• 다양한 장치 (V1)

각 서비스 시스템은 서로 다른 개수의 입력과 출력 장치와 연결되므로, 장치관리자는 다양한 인터페이스를 가지는 장치를 모두 관리해야 한다.

• 다양한 요구 데이타 (V2)

서비스에 따라서 필요로 하는 데이타와 데이타의 타입이 다양하며 동일한 서비스라 할지라도 연결된 장치에 따라 다양한 데이타 타입이 존재할 수 있다.

• 컨텍스트 분석 규칙 (V3)

사용자에게 정확한 서비스를 제공하기 위해서 입력된 사용자 정보 외에 수집된 컨텍스트 정보를 분석하게 된다. 수집된 컨텍스트 정보의 종류와 수가 모두 다양하므로, 이에 따라 다양한 컨텍스트 분석 규칙이 존재하게 된다.

• 적절한 서비스 결정하는 규칙 (V4)

분석된 컨텍스트 결과에 따라 적절한 서비스를 결정하는 규칙은 서비스 시스템이 제공하는 서비스 종류에 따라 다양할 수 있다.

• 서비스 컴포넌트의 구성 (V5)

각각의 서비스 시스템과 사용자의 상황에 따라 사용자에게 제공되는 서비스는 다양하고, 그에 따른 서비스 컴포넌트들의 구성과 포함된 서비스 컴포넌트들이 다를 수 있다. 심지어, 서비스결정에 따라 동일한 컴포넌트가 사용될지라도 컴포넌트 자체가 제공하는 기능이 다를 수도 있다.

• 서비스 컴포넌트간의 워크플로우 (V6)

컴포넌트들의 구성에 포함된 서비스 컴포넌트는 동일할 수 있지만 서비스 컴포넌트들 간의 워크플로우는 사용자의 선호도나 시스템에 따라서 다를 수 있다.

제안된 범용 아키텍처가 특정 어플리케이션에 맞게 특화되려면 어플리케이션에 맞게 가변성이 해소되어야 하며 이 때 의사결정 모델을 이용한다. 표 2의 의사 결정 모델은 서비스 시스템에 존재하는 6가지의 가변점에 대한 정보와 각 가변점에 할당될 수 있는 가변치, 가변성을 해결할 수 있는 방법에 대한 정보를 가지고 있다. 아키텍처 설계의 산출물인 아키텍처 명세서에는 아키텍처 드라이버를 해결하는데 사용한 스타일과 해당 스타일을 선택한 이유, 아키텍처를 구성하는 컴포넌트와 컴포넌트간의 관계를 보여주는 다이어그램 등이 기술된다[2]. 아키텍처를 설계하는 동안에 의사 결정 모델의 기술된 내용은 개념적인 내용에서 구체적인 내용으로 구체적으로 기술된다. 즉, 분석/ 설계 과정을 거치면서 분석 수준에서 기술되는 내용은 설계 단계에서 좀 더 상세히 기술된다. 그러므로, 표 2의 내용에는 아키텍처 명세서에 가변치에 따라 수정되어야 하는 내용 외에 아키텍처를 구성하는 컴포넌트에 대한 컴포넌트 명세서와 인터페이스 명세서, 더 나아가 각 컴포넌트의 구현 부분

산출물을 어떻게 수정해야 하는지도 포함된다.

예를 들어 V1 가변점을 특정 어플리케이션에 맞게 해결하기 위해서, 어플리케이션을 위한 아키텍처 명세서에는 해당 어플리케이션에 연결될 수 있는 다양한 장치와 인터페이스 정보를 기술하고, 아키텍처를 보여주는 다이어그램에도 연결되는 장치들과 관련 인터페이스 정보를 표현한다. 만약 '건강 보조 서비스 시스템'을 위한 아키텍처를 설계하는 경우, V1에 해당하는 가변치에는 '혈압 측정기', '맥박 측정기'와 같은 입출력 장치 이름이 기술되고 아키텍처 명세서에는 이 장치들의 인터페이스 정보가 추가되어야 하며, 아키텍처의 구조를 보여주는 다이어그램에도 이 정보들이 표현되어야 할 것이다. 그리고, 새로 추가된 입출력 장치가 해당 서비스 시스템과 상호작용할 수 있도록 인터페이스 명세서에는 장치에 대한 인터페이스 정보를 기술하고, 장치관리자의 인터페이스에 해당 장치의 인터페이스 정보를 추가한다. 예를 들어, 장치관리자의 인터페이스에 혈압 측정기에서 사용자의 혈압 정보를 받을 수 있도록 getUserBloodPressure(String userID, int highBP, int lowBP) 정보를 추가한다. 그리고나서, 컴포넌트 명세서와 관련 소스 부분에는 혈압측정기를 장치관리자에 등록하기 위해 장치관리자에 정의된 registerDevide(String id, String devideName)에서 혈압측정기 정보를 DB에 저장하는 로직을 추가하며, 혈압측정기에서 받은 혈압 정보를 다

표 2 서비스 시스템의 아키텍처 의사결정 모델

가변점		가변치	인스턴시에이션 작업
ID	이름		
V1	다양한 장치	어플리케이션에 따른 특정 입출력 장치	1. 다양한 입출력 장치 정보를 아키텍처 명세서에 기술한다. 2. 인터페이스 명세서에 입출력 장치에 따른 인터페이스 정보와 변경된 장치관리자의 인터페이스 정보를 기술하고, 이에 맞게 오퍼레이션의 매개변수를 수정한다. 3. 컴포넌트 명세서와 관련 소스 코드에 장치관리자에 연결된 입출력 장치를 처리할 수 있도록 장치관리자와 입출력장치의 오퍼레이션을 맞추고, 관련 로직을 수정한다.
V2	다양한 요구 데이타	서로 다른 요구데이타 집합	1. 아키텍처 명세서의 관련 다이어그램에 일련의 요구데이타를 아키텍처 명세서에 표현한다. 2. 인터페이스 명세서 요구데이타에 따라 알맞게 인터페이스 정보를 수정한다. 즉, 인터페이스 명세서 내의 오퍼레이션 매개변수를 수정한다. 3. 컴포넌트 명세서와 관련 소스 코드에 선택된 가변치에 맞게 입력 데이타를 처리할 수 있도록 오퍼레이션 구현부의 입력데이타 처리 로직을 수정한다.
V3	컨텍스트 분석 규칙	다양한 분석 규칙	1. 아키텍처 명세서의 관련 다이어그램에 특정 컨텍스트 분석 규칙을 컴포넌트에 플러그인시킬 수 있도록 표현하고, 간략하게 로직을 기술한다. (플러그인된 규칙은 한 인터페이스를 구현하기 때문에 인터페이스 명세서에는 수정할 내용이 없다.) 2. 컴포넌트 명세서와 관련 소스 코드 중, 컨텍스트 관리자의 구현부에 규칙 DB로부터 관련 규칙을 불러와서 해당 규칙에 맞게 컨텍스트를 분석할 수 있는 로직을 수정하거나 구현한다.
V4	적절한 서비스 결정하는 규칙	다양한 서비스 결정 규칙	1. 아키텍처 명세서에 서비스 시스템에 따른 서비스 결정 규칙을 컴포넌트에 플러그인될 수 있도록 표현하고, 규칙을 간략하게 기술한다. (플러그인된 규칙은 한 인터페이스를 구현하기 때문에 인터페이스 명세서에는 수정할 내용이 없다.) 2. 컴포넌트 명세서와 관련 소스 코드 중, 서비스 결정 관리자의 구현부에 규칙 DB로부터 관련 규칙을 불러와서 규칙에 따라 가장 적절한 서비스를 선택할 있는 로직을 새로 만들거나 수정한다.
V5	서비스 컴포넌트 구성	서로 다른 컴포넌트군	1. 아키텍처 명세서에 결정된 서비스의 종류 컴포넌트와 컴포넌트의 구성을 표현한다. 2. 동일한 컴포넌트가 다른 기능성을 수준을 제공할 경우, 컴포넌트 명세서에 컴포넌트의 제공된 기능성의 수준을 구체적으로 표현한다. 3. 선택된 서비스 컴포넌트에 해당하는 소스 코드의 경우, 다른 수준으로 기능을 제공하는 컴포넌트 적응시킬 수 있는 로직을 추가해야 한다.
V6	서비스 컴포넌트 간의 워크플로우	컴포넌트 간의 서로 다른 워크플로우	1. 서비스 컴포넌트 간의 메시지 교환 관계를 아키텍처 명세서의 다이어그램에 표현한다. 2. 인터페이스 명세서에 워크플로우를 구성하는 컴포넌트 간의 인터페이스가 서로 호환될 수 있는지 확인하고, 만약 서로 불일치하는 인터페이스가 존재한다면 어느 부분이 일치하지 않는지를 기술한다. 3. 선택된 서비스 컴포넌트에 해당하는 소스 코드의 경우, 인터페이스 불일치가 발생하는 컴포넌트에 대해서는 이를 해결할 수 있는 로직을 해당 컴포넌트에 추가해야 한다.

음 처리를 위해 컨텍스트 관리자에 보내는 로직 등을 추가한다.

이 장에서 식별한 가변성은 아키텍처와 관련된 가변성이지 컴포넌트의 가변성이 아님에 주목한다. 아키텍처와 관련된 가변성은 아키텍처 드라이버, 아키텍처 스타일, 컴포넌트와 컴포넌트 간의 관계에서 발행할 수 있다. 반면에 컴포넌트 가변성은 속성, 로직, 컴포넌트 내부의 워크플로우, 인터페이스 등에 발생할 수 있다[9].

소스코드를 제외한 산출물에 기술되는 내용은 자동화되기 힘들지만, 선택된 가변치에 따라 소스 코드를 수정해야 하는 것은 변경 로직을 지원하는 도구가 존재한다면 자동화될 수 있다. 그러므로, 의사 결정 모델 내용 중 소스 코드를 수정하는 부분을 뒷받침할 수 있는 도구가 필요하며, 이는 향후 연구로 진행될 것이다.

7. 사례연구

제안된 아키텍처의 적용 가능성을 보이기 위해 "실버 보조 서비스 시스템"에 대한 사례연구를 수행한다. 이 시스템의 요구사항, 드라이버, 유도된 아키텍처 스타일, 구체화된 컴포넌트를 가진 인스턴스화된 아키텍처를 보인다.

7.1 아키텍처 요구사항과 드라이버

실버 보조 서비스 시스템은 노약자의 삶의 질을 향상시키고 독립적인 생활을 지원하기 위한 지능화된 서비스를 제공한다[16]. 자동화된 서비스를 통해 시스템은 사람에 의한 수동적인 사회보장과 의료보장의 요구를 최소화한다.

실버 보조 서비스 시스템은 응급 상황에 대처할 수 있는 서비스를 제공하기 때문에 사용자의 건강 상태의 변화에 따라 적절한 서비스를 제공해야 한다. 시스템은 사용자에게 알맞은 서비스를 제공하기 위해서 사용자에 대한 특성 정보인 사용자 프로파일과 같은 정적인 정보와 컨텍스트 정보와 같이 주기적으로 획득한 동적인 정보를 분석한 후 미리 정의된 규칙에 기반한 서비스를 수행한다. 시스템은 신체적인 능력에 어려움을 가진 사용자가 쉽게 기능을 사용할 수 있도록 음성인식 장치, 아이콘화된 스크린과 같은 다중 모달(Multi-modal) 사용자 인터페이스를 지원하는 다양한 장치를 통해 이용할 수 있어야 한다. 시스템은 사용자가 이동하여도 수집된 건강 상태에 대한 컨텍스트 정보를 상황 분석에서 활용할 수 있도록 제공하여야 한다. 즉, 사용자가 움직일지라도 시스템은 끊김 없는 서비스를 제공하여야 한다. 또한, 실버 보조 서비스 시스템은 사람의 생명을 다루기 때문에 멈추지 않고 하루 24시간 운영되어야 한다. 지금까지 기술된 요구사항으로부터 식별된 아키텍처 드라이버를 표 3에서 설명한다.

7.2 아키텍처 스타일과 인스턴스화된 아키텍처

5장에서 명시한 아키텍처 스타일 중에서 시스템의 가용성을 높이기 위해 추가적인 아키텍처 드라이버로 파이프와 필터 스타일을 사용하기로 결정한다.

아키텍처 명세서에 기술되는 그림 4는 표 2의 의사결정 모델을 이용해 인스턴스화된 실버 보조 서비스 시스템의 아키텍처를 보여준다. 어플리케이션 서비스 계층은 식별된 응급상황을 통보하고 적절한 상황에 따른 적절한 대응과 이에 대한 피드백을 제공하는 응급상황 보고 서비스를 포함하며, 하나의 서비스를 구성하기 위해 서비스 컴포넌트 간의 워크플로우가 형성되므로, 가변성 V5를 해결하기 위해 어플리케이션 서배스 계층에 각 기능을 제공할 수 있는 서비스 컴포넌트(응급상황 통보, 응급상황 대처, 응급상황 기록/보고)를 표현하였다. 먼저 식별된 응급상황을 가족, 의사, 간호사와 같이 사전에 등록된 사람이나 기관에 통보하고, 온도, 습도 조절과 같이 시스템이 응급상황에 대처 가능한 경우에는 이를 수행한 후 응급상황을 기록/보고하고 그렇지 못한 경우에는 바로 응급상황 기록/보고한다. 이와 같이, 동일한 서비스일지라도 상황에 따른 다수의 워크플로우가 형성되기 때문에 V6을 해결하기 위해 각 컴포넌트간의 워크플로우를 간략하게 표시하였다. 미리 정의된 규칙에 기반해서 응급 상황의 식별과 응급 상황의 분석이 시스템 소프트웨어 계층에서 수행되므로, V3과 V4를 해결하기 위해 컨텍스트를 분석하는 규칙과 서비스 컴포넌트를 결정하는 규칙이 추가되었음을 컨텍스트 관리자와 서비스 결정 관리자에 표현하였다. 각 상세 규칙에 대한 설명은 아키텍처 명세서 내의 텍스트 설명 또는 해당 컴

표 3 실버 보조 서비스 시스템 아키텍처 드라이버

드라이버	설명
적응 가능한 동적 재구성	서비스를 제공하기 위해서 유연하게 구성하고 런타임에 새로운 기능을 추가할 수 있는 시스템의 능력.
컨텍스트 인지	사용자에게 적절한 서비스를 제공하기 위해 사용자의 컨텍스트를 획득하고 분석할 수 있는 시스템의 능력.
다수의 사용자에게 개인화된 서비스 제공	하나의 서비스라 할지라도 다수의 사용자에게 동시에 개인화된 서비스를 제공할 수 있는 시스템의 능력.
매우 높은 분산 환경	움직이는 사용자 중심적으로 다수의 시스템과 장치가 상호작용하여 정보의 접근, 의사소통, 비즈니스 트랜잭션을 처리할 수 있는 시스템의 능력.
높은 가용성	사용중인 상태에서 주어진 시간에 요청된 기능을 수행할 수 있는 시스템의 능력.

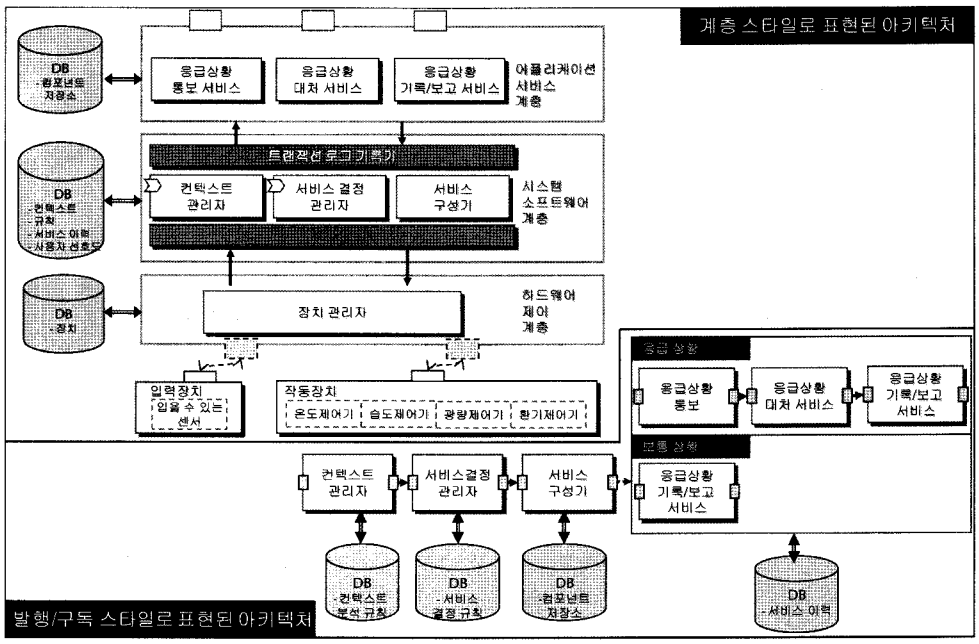

그림 4 실버 보조 서비스 시스템 아키텍처

```
3.1 컴포넌트 명세
3.1.1 장치관리자
장치 관리자 컴포넌트는 '실버 보조 시스템'에 연결되는 모든 입출력 장치를 관리하는
컴포넌트로, 입출 수 있는 센서, 온도/습도/광량/환기 제어기 등을 관리한다.
  가. 입력 데이터
    a. 기본 데이터
       성별(gender), 나이(age), 키(height), 몸무게(weight) …
    a. 입출 수 있는 센서
       최저혈압(lowBP), 최고혈압(highBP), 체온(temp), 심장박동수(HeartBeat)…
…
3.2 컨텍스트 관리자
컨텍스트 관리자 컴포넌트는 환자의 상태를 파악하는데 필요한 컨텍스트 정보를 장치
관리자에게 요청하며, 컨텍스트 규칙을 사용하여 환자 상태를 분석한다.
  가. 컨텍스트 분석 규칙
    if ( lowBP < 70  or highBP > 150 ) then state = 'alert'
    if ( temp > 45 ) then state = 'danger'
    …
```

그림 5 아키텍처 명세서에 기술되는 텍스트 기술의 일부

포넌트 명세서에 기술된다. 트랜잭션 로그 기록기는 시스템의 가용성을 높이기 위해 파이프와 필터 스타일이 적용되어 추가된 컴포넌트로 모든 트랜잭션을 필터링하고 로그 기록을 남긴다. 실버보조 시스템에는 환자의 건강 상태를 체크하는 다양한 장치들이 연결되기 때문에, V1을 해결하기 위해서 하드웨어 제어 계층에 연결될 수 있는 구체적인 입출력 장치인 사용자의 건강상태 정보를 전달하기 위한 입력장치와 응급처리 서비스의 결과로 제어하는 온도, 습도, 광량과 같은 환경을 조정하기 위한 작동장치를 표현하였다.

앞서 설명했듯이, 아키텍처 명세서에는 그림 4와 같은 다이어그램 외에 아키텍처를 구성하는 각 컴포넌트에 대한 간략한 기술도 포함되기 때문에, 이에 대한 상세 설명은 아키텍처 명세서나 다른 산출물에 명시되어야 한다. 그림 5는 다이어그램에 나타나지 않은 V2의 인스턴시에이션 작업과 다이어그램에 표현되었지만 V6과 관련된 인스턴스화 작업이 좀더 구체적으로 어떻게 아키텍처 명세서에 기술되는지에 대한 일부 내용을 간략히 보여준다.

8. 평가 및 결론

서비스 시스템은 사용자의 요구와 컨텍스트 정보를 이용해 사람의 일상 활동을 지원하는 자동화된 서비스를 제공한다. 서비스 시스템은 사용자의 요구사항을 만족시킬 수 있는 개인화된 서비스를 제공할 수 있어야 한다. 따라서, 사용자의 요구에 충분히 만족하는 서비스를 제공하기 위해서 서비스 시스템은 컨텍스트 인지와 컨텍스트에 맞게 서비스를 적응시키는 것이 매우 중요하다. 기존의 연구에서는 시스템과 시스템의 차이점에 해당하는 가변성에 대한 연구가 부족하였으며 가변성을 체계적으로 다루지 못하여 서비스 시스템의 아키텍처를 효율적으로 재사용하지 못하였다. 본 논문에서는 컨텍스트에 따라 다양하게 서비스가 제공되는 것을 가변성으로 간주하고, 가변성을 체계적인 방법으로 다루기 위해서 대표적인 재사용 방법론 중의 하나의 프로덕트 라인 공학 개념을 적용함으로써 컨텍스트 기반의 동적으로 적응 가능한 아키텍처를 제안하였다. 아키텍처 요구사항, 아키텍처 드라이버, 아키텍처 스타일, 컴포넌트의 선택을 통해 실체화된 서비스 시스템의 아키텍처를 명세하였으며 응급 상황을 다루는 서비스를 지원하는 실버

보조 서비스 시스템에 대한 사례연구를 통해 제안한 아키텍처의 적용성을 보였다.

PLA의 적용을 통해 서비스 시스템 도메인 내의 다수의 어플리케이션에 적용 가능한 범용적인 아키텍처를 기술하였다. 이를 통해 재사용성과 관련된 다수의 이점을 얻을 수 있다. 서비스 시스템 개발 비용 절감, 표준성 향상, 상호운용성 향상, 확장성과 유연성의 향상, 서비스 시스템의 구현에 앞선 증명이 가능하다. 본 논문에서 제안한 서비스 시스템의 범용 아키텍처가 특정 서비스 시스템의 요구사항에 맞게 인스턴스화 되기 위해서는 의사 결정 모델의 핵심 로직을 구현하는 도구가 필요하다. 이런 도구를 이용함으로써, 인스턴스화 작업이 보다 빨리 자동화되기 때문에 사용자의 요구에 신속하게 대처할 수 있도록 서비스 시스템의 구성이 변경될 수 있고 적응될 수 있게 된다. 그러므로, 의사 결정 모델을 뒷받침할 수 있는 도구에 대한 연구로 향후에 진행될 것이다.

참 고 문 헌

[1] Chesbrough, H., and Spohrer, J., "A Research Manifesto for Services Science," *Communications on the ACM*, Vol.49, No.7, pp. 35-34, ACM Press, 2006.

[2] Bosch, J. *Design and Use of Software Architectures*, Addison-Wesley, 2000.

[3] Gu, T., Pung, H.K., and Zhang, D.Q., "A Service-oriented middleware for building context-aware services," *Journal of Network and Computer Applications*, Vol.28, pp. 1-18, 2005.

[4] Hayes-Roth, B., Pfleger, K., Lalanda, P., Morignot, P., and Balabanovic, M., "A Domain-specific Software Architecture for Adaptive Intelligent Systems," *IEEE Transactions on Software Engineering*, Vol. 21, No.4, pp. 288-301, April 1995.

[5] Garlan, D., Cheng, S., Huang, A., Schmerl, B., and Steenkiste, P., "Rainbow : Architecture-based Self-Adaptive With Reusable Infrastructure," *IEEE Computer*, Vol.37, No.10, 2004.

[6] Garlan, D. and Schmerl, B., "An Architecture for Personal Cognitive Assistance," *Proceedings of the 2006 Conference on Software Engineering and Knowledge Engineering (SEKE 2006)*, pp. 91-97, 2006.

[7] Papazoglou, M.P., and Georgakopoulos, D., "Service-Oriented Computing," *Communications of the ACM*, Vol.46, No.10, pp. 25-28, 2003.

[8] Spohrer, J. and Riecken, D., "Services Science: Introduction," *Communications on the ACM*, Vol. 49, No.7, pp. 30-32, ACM Press, 2006.

[9] Kim S., et al., "A Theoretical Foundation of Variability in Component-Based Development," *Information and Software Technology(IST)*, Vol. 47, pp. 663-673, 2005.

[10] Soo Ho Chang, Hyun Jung La and Soo Dong Kim, "Key Issues and Metrics for Evaluating Product Line Architectures," *Proceedings of 18th International Conference on Software Engineering and Knowledge Engineering (SEKE 2006)*, pp. 212-219, 2006.

[11] Matinlassi, M., Niemela, E., and Dobrica, L., "Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture," VTT Technical Research Center of Finland, *Proceedings of ESPOO2002*, 2002.

[12] Kyo C. Kang et. al., "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," *Annals of Software Engineering*, 5, pp. 143-168, 1998.

[13] IEEE Architecture Working Group (AWG), 2000, Systems and Software Engineering - Recommended Practice for Architectural Description of Software-Intensive Systems (ISO/IEC 41020), IEEE, 2007.

[14] Clements, P., et al., *Documenting Software Architectures Views and Beyond*, Addison-Wesley, 2003.

[15] Noergaard, T., *Embedded Systems Architecture : A Comprehensive Guide for Engineers and Programmers*, Newnes, 2005.

[16] Nehmer, J., and Karshmer, A., "Living Assistance Systems - An Ambient Intelligence Approach," *Proceedings of the 28th international conference on Software engineering (ICSE 2006)*, pp. 43-50, 2006.

라 현 정

정보과학회논문지 : 소프트웨어 및 응용
제 35 권 제 2 호 참조

김 성 안

2005년 서울산업대학교 컴퓨터공학과 공학사. 2007년 숭실대학교 컴퓨터학과 공학석사. 2007년~현재 슈어소프트테크 연구원. 관심분야는 컴포넌트 기반 개발(CBD), Eclipse 플랫폼, 애자일 소프트웨어 개발

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), 객체지향 S/W공학, 컴포넌트 기반 개발(CBD), 소프트웨어 아키텍처