프로덕트 라인 기반의 모바일 응용 시스템 아키텍처 개발 프로세스

Architecture Development Process of Mobile Application System Based on Product Line

손이경(대구가톨릭대학교 컴퓨터정보통신공학부)
김행곤(대구가톨릭대학교 컴퓨터정보통신학부)
황하진(대구가톨릭대학교 경영학부)

요 약: 유비쿼터스 사회의 도래로 모바일 비즈니스 서비스 수요의 증대와 다양한 사용자 요구사항 및 변경이 빈번해짐에 따라 소프트웨어의 특성을 적시에 반영할 수 있는 새로운 소프트웨어 개발 기술이 필수적이다. 소프트웨어 프로덕트 라인은 공통의 유사한 기능을 가지고 있는 소프트웨어 제품 혹은 소프트웨어 시스템 집합으로 특정 영역의 시장과 용도의 요구사항에 따라 재사용 가능한 아키텍처 및 컴포넌트를 구성함으로써 생산성과 품질을 향상시킬 수 있다. 특히, 시스템을 분할하고 구조화하여 시스템의 성능과 효율성을 향상시킬 수 있는 소프트웨어 아키텍처 개념이 중요시 되면서 아키텍처의 개발과 평가에 대한 체계적인 연구가 필요하다. 본 논문에서는 CBD (Component Based Development)를 기반한 소프트웨어 프로덕트 라인(PLD: Product Line based Development)을 도입하여 모바일 비즈니스 도메인에 적합한 모바일 응용 시스템 아키텍처(MASA: Mobile Application System Architecture)를 제시한다.

Ⅰ. 서론

최근 유비쿼터스 사회의 도래로 모바일 비즈니스에 대한 서비스 수요의 증대와 다양한 사용자 요구로 인해 소프트웨어 산업은 빠르게 변화하고 있다. 제품 경쟁력의 핵심이 하드웨어 생산에서 소프트웨어 최적화 기술로 이동하면서 상품의 가치가 소프트웨어에 의해 좌우되는 기술집약적인 고부가가치 산업으로 발전하고 있다. 또한 다양한 비즈니스 영역에서의 새로운 요구사항에 대한 변경과 적용은 시스템의 성공뿐만 아니라 프로젝트의 효율성 및 생산성에도 영향을 끼친다. 따라서 요구사항들과 소프트웨어 특성을 적시에 반영할 수 있는 새로운 소프트웨어 개발 기술이 필수적이다[1,2,3]. 따라서 컴포넌트 기반 개발(CBD: Component Based Development) 기술을 확장하여 소프트웨어 아키텍처, 소프트웨어 컴포넌트 등 핵심자산들을 체계적으로 자원화하고 조직적으로 재사용하는 소프트웨어 생산기술인 프로덕트 라인 개발(PLD: Product Line Development) 기술이 필요하다. 프로덕트 라인은 연관된 시스템 그룹의 아키텍처를 기반으로 공통성과 변화성을 중심으로 특정 영역의 요구 사항을 만족하는 핵심자산의 체계적인 개발 및 관리, 조직적 재사용 기술을 제공하는 최상의 응용 개발을 위한 패러다임으로 인식되고 있다[4,5]. 특히, 시스템을 분할하고 구조화하여 시스템의 성능과 효율성을 향상시킬 수 있는 소프트웨어 아키텍처 개념이 중요시 되면서 완성된 시스템이 다양한 스테이크홀더들의 요구를 만족시키는 시스템인지의 여부를 결정하는 아키텍처의 개발과 평가에 대한 체계적인 연구가 필요하다[6].

본 논문에서는 모바일 응용 시스템 개발에 프로덕트 라인 기반 개발을 적용함으로써 모바일 응용 시스템의 개발 기간과 비용을 줄이고 소프트웨어의 품질을 보장하며 사용자들의 다양한 요구사항을 쉽고 빠르게 반영하고자 하였다. 따라서, CBD를 기반으로 프로덕트 라인 개발을 도입하여 모바일 응용 시스템 아키텍처 개발 프로세스를 제시하고자 한다.

Ⅱ. 관련 연구

2.1 프로덕트 라인

프로덕트 라인은 프로덕트 패밀리 사이에서 아키텍처와 재사용 가능한 컴포넌트의 집합을 공유함으로서 소프트웨어 재사용성 향상과 개발비용 감소를 가능하게 하는 방법론이다. 즉, 공통의 유사한 기능을 가지고 있는 소프트웨어 시스템 집합으로 핵심자산 개발과 프로덕트 개발 측면을 포함하고 있다. 프로덕트 라인의 핵심은 스코핑된 영역내의 프로덕트 간의 공통성과 변화성을 분석하여 프로덕트 라인 아키텍처를 수립하는 것이다. 프로덕트 라인 아키텍처는 연관된 프로덕트 집합과 조직에 의해 개발된 시스템을 위한 공통 아키텍처로서 생산성과 품질 향상에 이바지한다. 프로덕트 라인과 관련된 연구로는 아키텍처 정의와 전개, 컴포넌트 개발, COTS(Commercial Off-The -Shelf) 활용, 기존 자산의 마이닝(mining), 소프트웨어 시스템 통합 등이 있으며 이들은 실제 핵심자산을 생성하고 전개하는데 필요한 기술들이다.

(그림 1)은 프로덕트 라인 개발의 핵심 개념을 나타낸 것으로 프로덕트들은 시장 전략과 응용 도메인에 한정적이며 공통성을 가지는 아키텍처를 공유하고 기존 컴포넌트로부터 개발되고 구축된다. 이들 프로덕트들 역시 재사용을 위해 핵심자산으로 등록되어 관리된다[7,8,9].

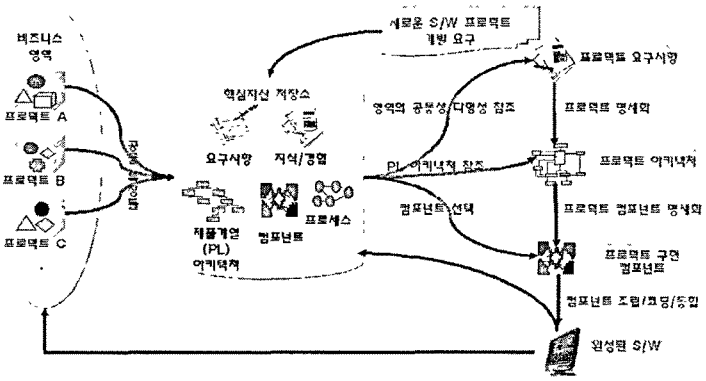


그림 1 프로덕트 라인 핵심 개념

2.2 소프트웨어 아키텍처

소프트웨어 아키텍처는 소프트웨어 컴포넌트, 이들 컴포넌트의 가시적인 속성, 컴포넌트들 사이의 관계로 구성된 시스템의 전체적인 구조이다. 아키텍처는 다양한 스테이크 홀더들 간의 원활한 의사소통의 수단이며 프로젝트 초기의 설계 결정사항을 기재해 놓은 산출물의 역할을 한다. 이러한 아키텍처에 대해 좋은 명세를 가지는 것은 아키텍처의 평가 성공에 결정적인 역할을 하게 되므로 완벽하고 명확한 문서가 필수적이다. Kruchten에 의해 제시된 4+1 뷰 모델은 아키텍처를 구축하고 분석하는 동안 스테이크 홀더 간의 개념 분리가 가능하여 아키텍처에 대한 이해를 향상시킬 수 있다. 이 모델은 4개의 뷰로 각 스테이크 홀더에게 적절한 설계 결정을 획득하게 하고 다섯 번째 뷰를 통해 각각을 설명하고 확인한다[10].

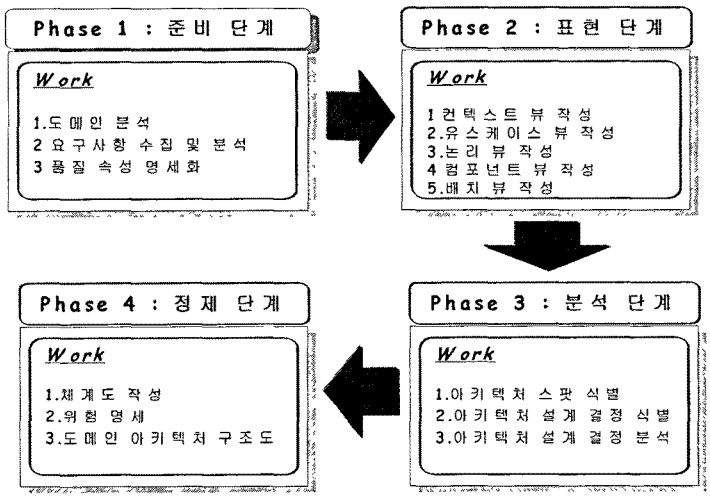
소프트웨어 품질 속성이란 시스템이 만족해야 하는 품질에 대한 요구사항을 의미하는 것으로 시스템 생명주기 초반에 완벽한 품질에 대한 요구가 결정되기 어렵지만 품질 속성을 만족하지 못하는 자원이 시스템 구축에 사용될 경우 시스템 전체가 품질에 대한 요구를 만족시키지 못하는 경우가 발생할 수 있다. 모바일 시스템 아키텍처에서 요구되는 품질 요소로서 즉시성, 편재성, 신뢰성, 확장성, 사용성, 이식성 등이 포함되어있다[11,12].

2.3 모바일 시스템

현대 사회에서 인터넷을 제외하고는 개인이나 사회생활을 이야기하기 어려울 정도로 인터넷이 차지하는 비중은 상당하다고 할 수 있다. E-메일은 생활의 필수가 되었고 인터넷을 통하여 쇼핑, 예약, 은행 관련 업무뿐 아니라 각종 동영상을 실시간으로 볼 수도 있게 되었다. 여기에 언제 어디서나 통화할 수 있는 휴대폰의 등장으로 이동통신과 인터넷이 결합하여 시간과 물리적 공간의 한계를 벗어나 이동 중에 무선으로 인터넷 정보를 송·수신할 수 있게 되었다. 이러한 무선 인터넷 서비스는 현재 지속적인 무선 인터넷 인프라 확장과 무선 인터넷 망 개방 추진으로 그 수요가 폭발적으로 확산될 것으로 전망된다. 모바일 시스템은 모바일 기기를 이용하여 인터넷에 접속하여 이동 중에 무선으로 인터넷 정보를 송·수신할 수 있는 서비스 시스템으로서 필요할 때 편리하게 이용 가능하고 장소에 구애받지 않으며 생활필수품으로 제2의 정보 획득 수단이 된다. 모바일 서비스가 주는 편리함과 자유로움은 이제 단순히 음성 통화에 국한되지 않고 데이터 전송 서비스(SMS), E-메일 서비스, 모바일 컨텐츠, 모바일 커머스, 모바일 비즈니스까지 그 범위를 넓히고 있다. 모바일 비즈니스는 휴대폰과 PDA등 모바일 장치에 기반을 둔 비즈니스를 의미하며 M-정부, M-캠퍼스, M-유통, M-뱅킹, M-의료 등 종류와 특성에 따라 다양화되고 있고 개인에 특화된 모바일 서비스가 가능해진다[13].

Ⅲ. 모바일 응용 시스템 아키텍처 개발 프로세스

본 논문에서 제시하는 방법론은 크게 준비, 표현, 분석, 정제하는 네 가지 단계로 구성된다. 먼저 준비 단계에서는 기본 S/W 요구사항으로부터 아키텍처 개발 범위와 목표를 정하고 표현 단계에서는 UML을 이용한 4+1 뷰 모델을 이용하여 아키텍처를 표현하고, 분석 단계에서는 아키텍처 스팟을 식별하고 이들로부터 아키텍처의 설계 결정을 찾아 분석한다. 마지막 정제 단계에서는 이전 단계의 결과물들을 자신의 아키텍처의 품질 속성과 연결하기 위해 구조적인 형태로 통합하고 잠재적으로 문제점이 있는 아키텍처 설계 결정에 대해 위험을 기술하고 지금까지의 결과를 바탕으로 시스템의 공통 기능과 요구 기능을 중심으로 개발 환경을 고려한 도메인 아키텍처를 작성한다. 이러한 과정을 거쳐 소프트웨어 아키텍처에 대한 품질 속성을 예측하거나 품질 요구를 충족시킬 수 있도록 도와주며 품질 속성의 획득에 많은 영향이 있는 아키텍처 설계결정이 체계적으로 발견되어 명확히 표현되며 이들 설계 결정에 근거하여 아키텍처 위험이 기술된다. (그림 2)는 본 논문에서 제안된 아키텍처 개발 위한 핵심적인 활동을 나타낸 것이다. 입력단계는 초기소프트웨어 아키텍처와 응용소프트웨어 요구사항을 정제하여 입력하는 단계이다.



(그림 2) 모바일 응용 시스템 아키텍처 개발 프로세스

3.1 준비 단계

1) 개발 범위와 목표 결정

이 단계는 명확한 아키텍처 개발 범위와 목표가 결정되는 단계로써 먼저 시스템의 기능적 요구와 품질요구를 분리하여 문서화 할 수 있는 간단한 템플릿을 정의한다<표 1>. 또한, 주요 기능들을 상대적 중요도에 따라 정렬함으로써 기능적 요구들에 대한 평가의 범위를 결정하고(FRs), 품질 요구와 기능 요구간의 관계를 기술함으로써 품질 요구에 대한 평가 범위를 결정한다(QRs). 마지막으로 각각의 품질 요구에 관련되어 있는 품질 속성을 식별한다.

표 1 요구 명세 템플릿

<table>
<tr><td>품질속성</td><td colspan="3">품질 속성의 이름(QA1, QA2, . .)</td></tr>
<tr><td rowspan="3">컨텍스트</td><td>품질 속성(QA1)</td><td colspan="2">관련 품질 요구 리스트(QR1, QR2,…)</td></tr>
<tr><td>품질 속성(QA2)</td><td colspan="2">관련 품질 요구 리스트(QR3, QR4,…)</td></tr>
<tr><td>. . .</td><td colspan="2">. . .</td></tr>
<tr><td rowspan="3">기능적
요구
(FRs)</td><td colspan="2">P1: 주요 기능의 명세1
P2: 주요 기능의 명세2
. . .</td><td>1(우선순위)
2
. . .</td></tr>
<tr><td colspan="3">P1_M1* 필수 기능 명세1(관련된 주요기능 표시)
P1_M2: 필수 기능 명세2
. . .</td></tr>
<tr><td colspan="3">O1* 선택 기능 명세1(관련된 주요기능 표시)
O2 선택 기능 명세2
. . .</td></tr>
<tr><td>품질
요구
(QRs)</td><td colspan="2">QR1* 품질 요구 명세1
QR2 품질 요구 명세2
.</td><td>관련 기능의 식별자
System
P1_M1</td></tr>
</table>

2) 품질 속성 명세화

본 논문의 방법은 품질 속성에 초점을 맞춘 개발이므로 각 품질 속성에 대한 명확하고 유용한 특성을 관리하는 것이 중요하다. 품질 속성에 대한 특성을 잘 표현할 수 있는 템플릿을 표 2와 같이 정의하였다.

표 2 품질 속성 명세표

속 성	요 인	원 인	대 응
QA#* 품질 속성 이름	품질 속성에 대한 구체적 요인	품질 속성에 영향을 미칠 수 있는 상황	원인에 대한 적절한 대응
	…	…	…

3.2 표현 단계

도메인 분석과 요구사항 템플릿을 기반으로 컨텍스트 뷰와 UML의 4+1 뷰 모델에 해당되는 유스케이스 뷰, 논리 뷰, 컴포넌트 뷰, 배치 뷰를 작성한다. 4+1 뷰 모델은 좋은 아키텍처 문서 획득에 도움이 되는 모델로서 품질 속성이 다양하게 다루어질 수 있고 광범위한 애플리케이션에 대해 적절한 수준의 추상화를 가진 아키텍처가 기술될 수 있다.

3.3 분석 단계

1) 아키텍처 스팟 식별

표현 단계에서 얻어진 다양한 아키텍처 뷰로부터 아키텍처 스팟을 식별한다. 아키텍처 스팟이란 설계에 있어 특정 품질 요소에 영향을 줄 수 있는 아키텍처 뷰나 아키텍처 스타일을 의미한다. 이러한 스팟을 식별하기 위해서는 먼저 시스템의 주요 요구 사항을 기술한 요구 사항 템플릿 분석하고, 템플릿의 기능적 요구에 관련되어 있는 품질 속성을 식별한 후, 이들 품질 속성에 영향을 줄 수 있는 아키텍처 스팟을 파악한다.

2) 아키텍처 설계 결정 식별 및 분석

표 4 아키텍처 설계 결정표

관련 품질 속성	QA1, QA2, . . .		
ADD #	결정 변수	결정 값	대 안
개략적 명세	설계 문제점	설계 해결책	대안 해결책
원리	품질 속성에 미치는 영향		
아키텍처 스팟	해당되는 아키텍처 스팟		

아키텍처 설계 결정은 품질 속성 특성표의 원인에 의해 제시될 수 있는 하나 또는 그 이상의 설계 이슈들을 의미하며 품질 속성 획득 여부에 많은 영향을 미치므로 개발 기간동안 아키텍처 설계 결정의 식별과 분석 작업은 매우 중요하다. 아키텍처 설계 결정을 찾기 위해서는 아키텍처 스팟으로부터 설계상에서 야기될 수 있는 문제점을 식별하여 <표 4>와 같이 결정 변수로 정의한다. 설계상의 지식이나 아키텍처들 간의 비교를 통해 제시된 문제점에 대한 해결 방법 중 한 가지를 결정 값으로 지정하며 다양한 해결 방법 중 결정 값으로 제시되지 않은 나머지를 대안으로 제시할 수 있다.

3) 작업 3: 아키텍처 설계 결정 분석

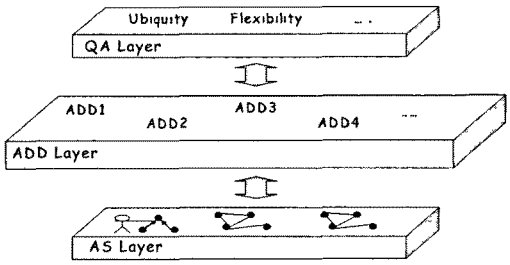
식별된 아키텍처 설계 결정들은 하나하나 독립적으로 분석되어야 하며 이들 설계 결정이 특정 품질 속성에 미치는 효과를 원리 항목에 기술하고 관련된 아키텍처 스팟도 함께 표현한다. 이 단계에서의 분석은 상세한 시뮬레이션이나 정확한 수학적 모델링을 필요로 하지는 않는다. 다만 설계에서의 위험 영역을 노출시키기 위한 품질 분석 정도를 의미한다.

3.4 완료단계

1) 결과를 체계화

이제까지 획득한 산출물 중 품질 속성과 아키텍처 설계 결정, 아키텍처 스팟간의 관계를 도식화하는 단계로서 (그림 3)과 같이 품질 속성 계층, 아키텍처 설계 결정 계층, 아키텍처 스팟 계층으로 구성된다. 각 계층 내에는 기존에 산출된 요소들을 포함하고 있으며, 이들 중 가운데 계층에 있는 아키텍처 설계 결정을 중심으로 품질 속성 계층의 요소들과 설계 결정 요소들 간의 연관관계를 표현하고, 다시 아키텍처 설계 결정과 아키텍처 스팟사이의 연관관계를 표현함으로써 각 계층에 포함되어 있는 요소들 간의 연관 관계를 종합적으로 조직화하게 된다.

아키텍처와 품질 속성들 간의 관계를 체계적으로 보여주는 것은 아키텍처에서 품질 속성을 이해하고 제어하는 능력을 향상시키고 설계에서 위험 영역을 식별하는데 도움을 줄 뿐만 아니라, 자신의 품질 속성에 대한 아키텍처의 타당성을 이해하는데도 유용하다.



(그림 3) 체계도

2) 아키텍처 위험 요소 기술

잠재적 위험 요소를 가진 설계 결정에 대한 명세를 기술하는 단계로서 <표 5>와 같은 위험 명세표가 산출물로 나타난다. 먼저, 아키텍처 설계 결정으로부터 위험 요인을 식별하여 위험의 종류로 표현하고 관련된 아키텍처 설계 결정을 기술하며 위험에 관련되어 있는 속성을 기술한다. 또한 위험을 야기시키는 구체적인 원인을 기술하고 결과로써 이러한 위험 요소가 품질 속성에 미치는 영향을 파악한다. 마지막으로 이러한 위험에 관련되어 있는 아키텍처 스팟의 식별자를 표기한다.

아키텍처에 관한 위험을 문서로 표현하는 이유는 잠재적인 위험을 노출시킴으로써 아키텍처 설계자가 사전에 위험 요소를 인식할 수 있게 되고 궁극적으로는 위험을 완화시킬 수 있는 계획을 세울 수 있게 하는데 의의가 있다.

표 5 위험 명세표

RISK #	위험의 종류
품질 속성	관련 있는 품질 속성
아키텍처 설계 결정	관련 있는 아키텍처 설계 결정
원 인	위험의 원인이 되는 상황
결 과	관련된 품질 속성에 미치는 영향

3) 도메인 아키텍처

지금까지의 결과를 토대로 모바일 시스템의 도메인에 적용될 수 있는 도메인 아키텍처를 작성하는 단계로서 공통의 기능과 요구 기능을 중심으로 개발 환경을 고려하여 3-tier 구조로 도메인 아키텍처를 작성한다.

(그림 4)와 같이 모바일 시스템의 도메인에 적용될 수 있는 도메인 아키텍처는 모바일 시스템에 대한 사용자, 설계자, 개발자의 요구 사항과 기능적 독립성을 고려하여 사용자 계층, 비즈니스 로직 계층, 데이터 계층으로 구성된다. 각 계층은 상호 독립적으로 운용되며 사용자 계층과 비즈니스 계층 간에는 모바일 체제의 네트워크가 존재한다.

- 사용자 계층 : 사용자 측면을 고려하여 일반적으로 화면상에 존재하는 UI를 담당하는 부분으로 다양한 사용자 요구사항을 서비스 형태로 외부에 표현하기 위한 부분이다.
- 비즈니스 로직 계층 : 사용자에게 지원되는 로컬 비즈니스 행위를 구현하는 부분으로 외부적인 서비스를 지원하기 위한 내부 구현과 밀접한 관련이 있는 부분이다. 서비스로는 음성통화, 데이터 전송, E-메일 서비스, 모바일 커머스, 교육이나 게임, 뉴스, 아바타 등과 같은 모바일 컨텐츠, 모바일 뱅킹이나 텔레매틱스 등과 같은 모바일 비즈니스가 포함된다.
- 데이터 계층 : 내·외부적인 서비스에서 수반될 수 있는 자료를 관리하기 위한 부분으로 공유되는 자원들에 대한 물리적인 처리를 담당하는 부분이다.

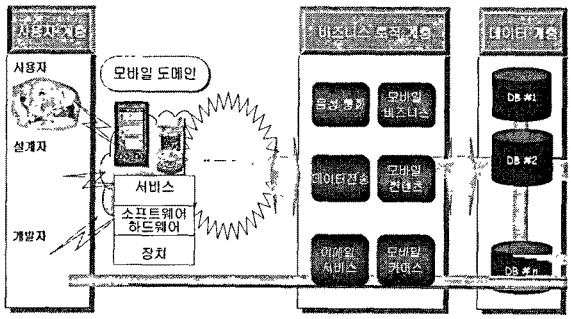


그림 4 모바일 시스템 도메인 아키텍처

Ⅳ. 결론 및 향후 연구

본 논문에서는 CBD 기반의 소프트웨어 프로덕트 라인을 도입하여 모바일 응용 시스템에 적용한 것으로 모바일 응용 시스템의 요구사항을 분석하여 모바일 응용 시스템 아키텍처 개발 프로세스를 제안하였다.

제안된 프로세스는 아키텍처 개발을 위해 준비, 표현, 분석, 정제 단계로 구성된다. 준비 단계에는 목표 시스템의 개발 범위와 목표 등을 파악하는 도메인 분석과 요구사항 수집 및 분석, 품질 속성 명세화 작업을 제시하였다. 표현단계는 시스템의 경계를 표현하는 컨텍스트 뷰와 요구사항으로부터 유스케이스 뷰, 논리 뷰, 컴포넌트 뷰, 배치 뷰를 작성한다. 분석 단계는 아키텍처 뷰로부터 설계에 영향을 줄 수 있는 뷰들을 파악하여 아키텍처 스팟으로 식별하고 이들 아키텍처 스팟으로부터 아키텍처 설계 결정을 식별하고 분석하여 아키텍처 설계 결정표를 제시하였다. 마지막 정제 단계에서는 품질 속성과 아키텍처 설계 결정, 아키텍처 스팟간의 관계를 파악할 수 있는 체계도를 작성하고 잠재적인 위험을 명세하며 지금까지의 결과를 바탕으로 시스템의 공통 기능과 요구 기능을 중심으로 개발 환경을 고려한 도메인 아키텍처를 작성하였다. 이와 같이 각 단계마다 세부적인 작업 절차와 산출물을 제시하였으며 이들 세부 작업을 수행함에 따라 품질 속성 중심의 체계적인 아키텍처 개발이 가능하였다.

기대 효과로는 소프트웨어 핵심자산의 체계적인 개발과 관리 및 조직적인 재사용을 통하여 모바일 응용 시스템 개발 기간과 비용을 줄일 수 있고, 검증된 핵심자산을 사용함으로써 소프트웨어 품질 보장과 다양한 요구사항을 쉽고 빠르게 반영할 수 있다. 또한 품질 속성 중심의 아키텍처 설계 결정을 제시하고 잠재적인 위험을 명세함으로써 아키텍처의 풍부한 재사용 가능성을 제공할 수 있게 된다. 특히 모바일 시스템과 같은 실시간 응답이 요구되는 응용 시스템에서 소비자 요구를 충족시킬 수 있으므로 더욱 효율적이다.

향후 연구로는 동일 영역 프로덕트 생산 시 핵심자산들을 재사용하여 소프트웨어 프로덕트를 조립 및 생산하는 기술에 관한 연구와 다양한 영역별 프로덕트 라인 아키텍처 개발을 통한 특정 프로덕트로의 적용이 필요하다.

참 고 문 헌

[1] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere,, "The Architecture Tradeoff Analysis Method," The 4th IEEE International Conference on Engineering of Complex Computer Systems, pp. 68-78, August 1998.

[2] Dobrica, L. and Niemela, E., "A Survey on Software Architecture Analysis Engineering" IEEE Transactions on Software Engineering, Vol. 28, No. 7, pp. 638-653, July 2002.

[3] Klaus Schmid, "The Economic Impact of Product Line Adoption and Evolution," IEEE SOFTWARE, Vol. 19 No. 4, pp. 50-57, July/August 2002.

[4] eila Niemela, Tuomas Ihme, "Product line software engineering of embedded systems," Symposium on Software

Reusability Proceedings of the 2001 symposium on Software reusability, putting software reuse in context, Toronto, Ontario, Canada, pp. 118-125, 2001.
[5] Northrop, "A Framework for Software Product Line Practice, 2001. http://www.sei.cmu.edu/plp/framework.html
[6] Klaus Schmid, "People Issues in developing Software Product Lines," IESE-Report No. 051.01/E, Version 1.0, 2001.
[7] Colin Atkinson, Component-based Product Line Engineering with UML, Addision-Wesley, 2002.
[8] Mari Matinlassi, "Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA," International Conference on Software Engineering, Proceedings of the 26th International Conference on Software Engineering, Vol. 00, pp. 127-136. 2004.
[9] Charles W. Krueger, "Variation Management for Software Product Lines," SPLC 2, San Diego, CA, USA, Vol. 2379, pp. 37-48, 2002.
[10] Jacobson, Booch, Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999.
[11] Christine Hofmeister, Robert Nord and Dilip Soni, Applied Software Architecture, Addsion-Wesley, 2000.
[12] Carnegie Mellon University, "How Do You Define Software Architecture," February 2003. http://www.sei.cmu.edu/architecture/definitions.html
[13] 김기천, "모바일 서비스 기술 동향," 한국정보처리학회지, 제9권, 제2호, pp. 17-23, March 2002.
[14] Haeng-Kon Kim, Lee-Kyeong Son "A Study on Software Architecture Evaluation," Journal of Electronics & Computer Science, Vol.5, No. 2, pp. 37-48, 2003.
[15] Action Semantics Models, Unified Modeling Language Specification, Version 1.5 OMG Document, Formal /03-03-01, 2003.