아키텍처 모델링을 위한 요구사항 정량화 기법
(A Quantitative Approach to Requirements Analysis for Architectures Modeling)

김 진 태 [†] 양 원 석 [††] 정 창 해 [††] 박 수 용 [†††]
(Jintae Kim) (Wonseok Yang) (Changhae Jang) (Sooyong Park)

요 약 요구사항은 아키텍처를 생성할 때 매우 중요한 요소로써 기능 요구사항과 품질 요구사항으로 구분된다. 기능 요구사항은 하위시스템 또는 컴포넌트 생성에 영향을 미치고 품질 요구사항은 아키텍처의 구조를 결정하는데 영향을 준다. 이와 같이 요구 사항은 아키텍처 설계에 중요한 영향을 끼치기 때문에 아키텍처 설계를 위해서는 요구사항에 대한 명확한 이해가 필요하다. 본 논문에서는 요구사항에 대한 명확한 이해를 돕기 위해 요구사항을 정량화하는 방법을 제안한다. 기능 요구사항은 기능적인 우선순위 계산을 통해 컴포넌트를 정량화하고 품질 요구사항은 정량화된 컴포넌트와 품질 속성의 연관성 계산을 통해 정량화한다. 제안된 방법은 DRAMA (Domain Requirements Analysis for Modeling Architectures) 도구를 통해 구현되었으며, 요구사항의 정량화 방법을 실 예제에 적용해본 결과를 소개한다.

키워드 : 요구사항, 품질 속성, 아키텍처, AHP, 정량적 분석

Abstract Requirements are very important to model software architecture. Requirements are divided into functional and quality requirements. Functional requirements are pinpointed subsystems and components. Quality requirements affect the structure of architecture. Thus requirements are essential to understand clearly in order to design software architecture. This paper focuses on a quantitative approach to requirements analysis for modeling architectures. In our proposal, functional requirements are quantified through calculating each priority of components. Quality requirements are quantified through calculating the correlation degree between components and quality attributes. The proposed method is implemented by DRAMA (Domain Requirements Analysis for Modeling Architectures), which fully supports our approach and are developed in Java environments. Our proposal is validated to apply some industrial examples.

Key words : Requirements, Quality attribute, Architecture, AHP, Quantitative analysis

1. 서 론

요구 공학(Requirements Engineering)과 소프트웨어 아키텍처(Software Architecture)는 성공적인 소프트웨어 개발을 위한 중요한 영역이다[1]. 요구 공학은 개발될 소프트웨어 시스템을 정의한다[2]. 소프트웨어 아키텍처는 소프트웨어 시스템의 구조, 구성요소, 품질을 나타낸다[3]. 소프트웨어 아키텍처는 요구사항, 개발 전략, 제약 사항을 고려하여 설계된다[4]. 그 중 요구사항은 아키텍처 설계의 핵심 요소다[5].

요구사항은 기능적 요구사항(Functional Requirements)과 품질 요구사항(Quality Requirements)으로 구분된다. 기능적 요구사항은 아키텍처를 구성하는 서브시스템 또는 컴포넌트의 생성에 영향을 미친다. 반면에 품질 요구사항은 아키텍처의 구조에 영향을 미친다[6]. 이와 같이 요구사항은 아키텍처 생성의 근거가 되므로 요구사항을 명확하게 이해할 필요가 있다.

기존에 아키텍처 생성의 근거로써 요구사항을 이해하는 대표적인 연구는 Chung의 ABAS(attribute based architecture style)가 있다[7]. ABAS는 품질속성에 따른 적절한 아키텍처 스타일을 제공한다. 그러나 품질속

· 본 연구는 정보통신부 지원 ITRC 프로그램의 지원을 받아 수행되었음
· 본 연구는 한국과학재단 목적기초연구 R01-2003-000-10197-0 지원으로 진행되었음
† 정 회 원 : 삼성전자 정보통신연구소
canon.kim@gmail.com
†† 학생회원 : 서강대학교 컴퓨터학과
hiyws@sogang.ac.kr
zenlover@sogang.ac.kr
††† 정 회 원 : 서강대학교 컴퓨터학과 교수
sypark@sogang.ac.kr
논문접수 : 2004년 11월 29일
심사완료 : 2005년 12월 1일

성을 파악하는 것이 정성적인(qualitative) 방법에 의해 이뤄지므로 매우 주관적으로 평가하게 된다. 하지만 아키텍처를 결정하기 위해서는 요구사항을 객관적으로 분석할 필요가 있다.

본 논문은 아키텍처 설계를 위해 요구사항을 정량화하는 방법(DRAMA: Domain Requirements Analysis for Modeling Architectures)제안하며 이를 지원하는 도구를 설명한다. 아키텍처를 객관적 근거에 의해 생성하기 위해서는 요구사항을 정량화하는 것이 필요하다. 요구사항을 정량화 하기 위해서는 기능 요구사항을 분석하여 통해 도출된 컴포넌트에 대한 기능 측면에서 우선순위를 정하고 우선순위를 갖는 컴포넌트 정보를 바탕으로 품질 속성과의 연관성을 계산한다. 이를 통해 우선순위를 갖는 컴포넌트와 정량화된 품질 속성 값이 도출된다. 우선순위를 갖는 컴포넌트와 정량화된 품질 속성 값은 객관적으로 아키텍처를 결정하기 위해 사용된다.

본 논문에서 제안하는 방법의 제약 사항은 다음과 같다.

1) 논문이 다루는 범위는 그림 1에서 점선으로 표현된 영역이다.
2) 품질 속성은 [4]에서 제시한 분류를 사용한다.
3) 아키텍처 스타일은 Mary Shaw[3], SAP(Software Architecture in Practice[4]), 그리고 POSA (Pattern-Oriented Software Architecture[8])가 제안한 방법을 사용한다.
4) 소프트웨어 개발을 위한 기술 및 관련된 하드웨어의 특성은 배제한다.

본 논문의 구성은 다음과 같다. 1장에서는 본 연구의 필요성과 방향을 제시하였고, 2장에서는 DRAMA에서 사용되는 기본 개념을 소개하며 3장에서는 ATM 예제를 통해 DRAMA의 전체공정을 설명한다. 4장에서는 DRAMA 도구를 이용하여 실 예제에 적용한 사례를 통해 제안된 방법을 검증한다. 5장에서는 결론과 향후 연구에 대해 설명한다.

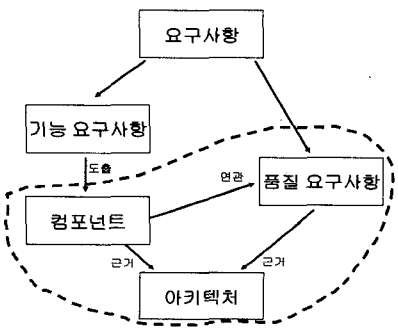

그림 1 연구범위

2. DRAMA에서 사용되는 기본 개념

본 장에서는 DRAMA에서 나타나는 개념을 설명한다. DRAMA에서 나타나는 개념은 목표 기반의 요구사항 분석, AHP(Analytic Hierarchy Process), 품질속성 정량화를 위한 매트릭스, 그리고 아키텍처 스타일이다.

2.1 목표 기반의 요구사항 분석

목표 기반의 요구사항 분석[9-11]은 요구사항으로부터 컴포넌트를 도출하기 위해 사용된다. 목표란 이해 관계자가 개발될 시스템에서 성취하고자 하는 것이며 추상화된 최상위 수준의 요구사항이다.

목표 기반으로 요구사항을 추출, 분석하면 1) 시스템 전체에 대한 요구사항을 파악하기 쉬워지며, 2) 기능적, 비 기능적 요구사항을 균형적으로 추출할 수 있고, 3) 추출된 요구사항에 대한 생성 근거를 제시할 수 있다. 따라서 요구사항으로부터 아키텍처를 생성하기 위해 목표 기반의 요구사항 분석 기법이 유용하다.

그러나 목표들의 분화로 요구사항을 분석하는 기법은 어느 단계에서 목표의 분화를 멈춰야 할지 모호하다. 따라서 그림 2와 같이 4개의 요구사항 추상화 단계를 통해 목표 기반의 요구사항 분석을 진행한다.

요구사항은 요구사항 추상화 계층에 따라 목표를 분화(decomposition)하고 정제(refinement)하여 구체화(concretion)한다. 요구사항 추상화 계층은 4개의 계층(비즈니스 단계, 서비스 단계, 상호작용 단계, 기능 단계) 으로 나눈다[11]. 비즈니스 단계는 제품에 대한 궁극적인 목표를 정의하고 기술한다. 비즈니스 단계에서 요구사항은 비즈니스 목표(Business goal)로 표현한다. 서비스 단계는 시스템이 사용자에게 제공해야 하는 서비스를 정의한다. 서비스 단계에서 나타나는 서비스 목표는 비즈니스 목표를 만족시켜야 한다. 상호작용 단계는 서비스가 실현되기 위해 필요한 상호작용을 나타내며 상호작용 목표로 표현된다. 상호작용은 시스템과 에이전트(외부 시스템, 사용자, 개발자)들 간에 발생한다. 기능 레벨은 상호작용을 만족시켜주기 위해 시스템이 내부적으로 가져야 할 기능에 중점을 둔다.

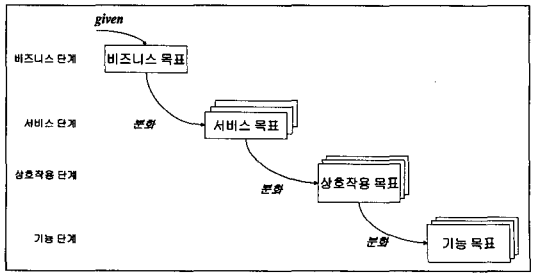

그림 2 요구사항 계층에 의한 목표 기반 분석

표 1 쌍방 비교 값

상대적 강도	정의	설명
1	거의 같음	두 요구사항이 동일한 중요하다.
3	다소 높음	한쪽 요구사항이 다소 선호도가 높다.
5	높음	한쪽 요구사항이 선호도가 높다.
7	많이 높음	한쪽 요구사항이 매우 선호도가 높고 실례에서도 우세하다.
9	가장 높음	모든 경우 중에서 가장 높은 선호도와 중요도를 갖는다.
2,4,6,8	두 값들 사이의 중간값	타협이 필요할 때에 사용된다.
역수관계	만일 C_i가 C_j와 비교하여 위의 수치를 갖게 되면 C_j는 C_i와 비교할 때 상대적 강도의 값으로 역수를 취하게 된다.	

본 절에서 제시한 목표 기반의 요구사항을 분석하면 트리 형태로 요구사항이 표현된다[11].

2.2 AHP(Analytic Hierarchy Process)

AHP(Analytic Hierarchy Process)[12]는 비교 대상들을 서로 쌍으로 비교(pairwise comparison)하여 상대적인 값이나 수치로 나열하는 방법이다. 이를 바탕으로 모든 대상들의 중요도가 정량적 값으로 나타난다. AHP 방법을 사용할 때 얻을 수 있는 이점을 다음과 같다. 첫째, 정성적인 문제를 정량적인 방법으로 해결함으로써 체계적인 의사결정을 가능하게 한다. 둘째, 복잡하고 불명확한 문제를 여러 계층으로 정리하고 일대일 비교를 통해 각각의 중요성이나 성취도를 평가함으로써 좀더 정확한 의사 결정을 할 수 있다. 셋째, 다수 관계자들의 의견과 비중을 반영함으로써 보다 객관적인 평가를 할 수 있다. 아래는 AHP를 사용하는 방법을 각 절차 별로 설명한다.

STEP1: N개의 요구사항들을 **n X n 표에 나열한다.** 예를 들어 어느 시스템의 요구사항이 Req1, Req2, Req3, Req4로 분석이 되었다고 가정하자. 4개의 요구사항을 4×4크기의 표에 입력한다.

STEP2: N개의 요구사항들을 쌍으로 서로 비교한다. 비교할 때 입력 값은 표 1에서 제시된 범위 값을 입력한다.

예를 들어 Req1이 Req2보다 1/3의 중요도를 갖는다면 2행 3열에 1/3을 입력한다(표 2 참조). 대신 3행 2열에는 역수값 3을 입력한다.

표 2 AHP 예제 1

	Req1	Req2	Req3	Req4
Req1	1	1/3	2	4
Req2	3	1	5	3
Req3	1/2	1/5	1	1/3
Req4	1/4	1/3	3	1

STEP3: eigen value를 구하기 위해 정형화 행에 대해 평균을 구한다. 표 3에서 Req1열의 합으로 Req1 열의 각 행의 값을 나눈다. 새로운 열을 오른쪽에 추가하여 각 행의 합을 구한다(표 3 참조).

표 3 AHP 예제 2

	Req1	Req2	Req3	Req4	*Sum*
Req1	0.21	0.18	0.18	0.48	*1.05*
Req2	0.63	0.54	0.45	0.36	*1.98*
Req3	0.11	0.11	0.09	0.04	*0.34*
Req4	0.05	0.18	0.27	0.12	*0.62*

그 후에 행의 합으로 정형화(normalize)한다.

$$\frac{1}{4}\cdot\begin{bmatrix}1.05\\1.98\\0.34\\0.62\end{bmatrix}=\begin{bmatrix}0.26\\0.50\\0.09\\0.16\end{bmatrix}$$

STEP4: 요구사항 각각에 대해 상대적인 중요도를 부여한다.

Req1 = 26%

Req2 = 50%

Req3 = 9%

Req4 = 16%

STEP5: 일관성(Consistency)을 검사한다.

각 요구사항의 상대값의 신뢰 정도를 확인하기 위해 일관성을 검증한다. 예를 들어 Req1이 Req2 보다 매우 크고, Req2가 Req3보다 크고, Req3이 Req1보다 다소 크다면 불일치(inconsistency)가 발생한다. 그러므로 각 요구사항의 중요도는 신뢰성이 떨어진다. 이러한 불일치를 해결하기 위해 AHP 기법은 일대일 비교 과정에서의 일관성 여부 즉, 일관성을 점검할 수 있는 방법을 제공한다.

AHP 기법에서는 일관성을 점검하기 위해 *CI*(consistency index)를 다음과 같이 정의한다.

$$CI=\frac{\lambda_{\max}-n}{n-1}$$

λmax 값을 다음과 같이 구한다. 쌍방비교를 통해 나타난 행렬을 우선순위 값으로 행렬 곱한다.

$$\begin{pmatrix}1&1/3&2&4\\3&1&5&3\\1/2&1/5&1&1/3\\1/4&1/3&3&1\end{pmatrix}\bullet\begin{pmatrix}0.26\\0.50\\0.09\\0.16\end{pmatrix}=\begin{pmatrix}1.22\\2.18\\0.37\\0.64\end{pmatrix}$$

그 결과 값을 다시 우선순위 값으로 나눈다.

$$\begin{pmatrix} 1.22/0.26 \\ 2.18/0.50 \\ 0.37/0.09 \\ 0.64/0.16 \end{pmatrix} = \begin{pmatrix} 4.66 \\ 4.40 \\ 4.29 \\ 4.13 \end{pmatrix}$$

λmax 값을 계산하기 위해 앞에서 계산된 벡터 값을 n으로 나눈다.

$$\lambda \max = \frac{4.66 + 4.40 + 4.29 + 4.13}{4} = 4.37$$

따라서 CI는 다음과 같이 구해진다.

$$CI = \frac{\lambda \max - n}{n-1} = \frac{4.37 - 4}{4-1} = 0.12$$

일관성 지수 CI를 구한 후에 이를 이용하여 일관성 비율(consistency ratio: CR)을 구한다. CR은 다음과 같이 정의한다.

$$CR = \frac{CI}{RI}$$

일관성 비율은 일관성 지수를 RI(random indices)로 나눈 값이 된다. RI는 통계학적으로 n값이 따라 아래와 같이 이미 정해져 있다[12].

1	2	3	4	5	6	7	8	9	10	11	12
0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.48

일반적으로 CR 값이 0.10 이하일 경우 AHP 기법을 통한 쌍방비교 값은 적합하다고 할 수 있다.

따라서 n=4 이므로

$$CR = \frac{CI}{RI} = \frac{0.12}{0.90} = 0.14$$

CR 값이 0.10 보다 크므로 정합성이 거짓이 되므로 STEP2~STEP4를 반복하여 쌍방비교 값을 재입력한다.

2.3 Matrix

본 논문에서는 품질 속성의 중요도를 정량적으로 계산하기 위해 Matrix 기법을 사용한다. Matrix 기법은 Jintae Kim[9,10]이 제안한 품질 속성 계산 방법을 기반으로 확장한 기법이다.

컴포넌트 중요도를 바탕으로 각 컴포넌트들과 관련된 품질속성의 상호 관련도를 계산한다. 이를 통해 전체적인 시스템의 품질 속성별 중요도를 계산한다.

그림 3은 Matrix를 이용한 품질속성 측정방법을 설명하고 있다. 그림 3을 이용하여 품질속성을 정량화하기 위해 먼저 표를 만들어 상단 행에는 품질속성들을 나열한다(표 7 참조). 좌측 열에는 식별된 컴포넌트들을 나열한다. 각 컴포넌트들을 기준으로 해당 품질속성에 관련성(correlation)을 체크한다. 이때에 0~3의 값을 기입한다(0: 없음, 1: 약함, 2: 보통, 3: 강함)-본 기법의 목표는 정성적인 품질속성을 정량화 하여 각 품질속성 별로 비교하기 수월하게 하는 것이다. 따라서 4단계(0~3의 값)의 비교 순위 만으로도 충분히 그 정도를 객관화하여 판단할 수 있으며 이러한 4단계의 등급을 통한 방법은 소프트웨어 공학에서 정통적으로 사용되고 있는 방법이다. 최종적으로 품질속성 별로 컴포넌트 중요도와 품질속성 관련도의 곱으로 품질속성 중요도를 계산한다.

x is a quality attribute
the priority of x: P(x)
n is the total number of components
$1 \le i \le n$
correlationValue(x,i) = 0, 1, 2, *or* 3

$$P(x) = \sum_{1}^{n} the\ weight\ of\ Component(i) \times correlationValue(x,i)$$

그림 3 Matrix를 이용한 품질속성 측정 방법

2.4 아키텍처 스타일

본 논문에서는 Mary Shaw[3], Len Bass[4], POSA [8]의 아키텍처 스타일을 기반으로 아키텍처 스타일을 분류한다. 다음은 아키텍처 스타일 별로 특징을 기술하고 있다.

Data-Centered architectural style: 데이타의 무결성(integrity)을 보장해주는 아키텍처 스타일이다. 중앙에 데이타 저장소가 있고 다수의 클라이언트가 접속해 커뮤니케이션을 이룬다. 클라이언트는 각각 독립적이며 데이타 저장소 또한 클라이언트에 대해 독립적이다. 따라서 클라이언트가 추가될 수 있다는 측면에서 확장성(Scalability)이 높다. 종류로는 Blackboard와 Repository 스타일이 있다.

Data-Flow architectural style: 데이타의 흐름을 처리하는 시스템을 구조화하는데 유용하다. 대표적인 예로 pipe and filter 스타일이 있다. 데이타는 pipe를 통해서 전송되고 pipe는 filter를 연결한다. 각각의 처리 과정은 filter 컴포넌트에 캡슐화되어 있기 때문에 reusability와 modifiability가 높은 반면에 Performance는 매우 낮다.

Call-and-Return architectural style: 대형 소프트웨어 시스템에 주로 사용된다. 대표적인 예로 Layered style이 있다. Layered style은 하위 작업의 그룹들을 계층구조로 나눌 수 있는 application을 구조화하는데 유용하다. 각 계층은 기본적으로 위 아래의 계층과 통신이 가능하다. 여러 추상화 계층이 필요한 시스템에 적합하며, 한 layer에 대해 여러 구현 방법이 제공되기 때문에 재사용성이 높아진다. Layer간의 통신은 상, 하위 layer만 통신이 가능하므로 안전성을 높일 수 있고 layer간의 확장(scalability)과 수정(modifiability)도 가

능하다.

Broker architectural style: 분산 소프트웨어 시스템을 구조화하는데 유용하다. Broker style은 컴포넌트를 이용하여 시스템을 구조화한다. 이 컴포넌트는 상호간의 결합력이 약하고 원격 서비스 호출을 이용하여 다른 컴포넌트와 통신을 한다. Broker를 사용하면 컴포넌트 간의 통신을 조정하고 복잡한 컴포넌트간의 통신을 간략화 할 수 있기 때문에 코드의 수성용이성(modifiability)을 높일 수 있다.

Model-View-Controller style: 어플리케이션을 세 가지 컴포넌트로 나눈다. Model은 핵심 기능과 데이타 처리하는 기능을 한다. View는 정보를 사용자에게 보여주는(display) 기능을 한다. Controller는 사용자의 입력을 처리해 준다. View와 Controller은 사용자와의 인터페이스의 역할을 하기 때문에 코드의 usability를 높여준다.

분류된 아키텍처 스타일은 해당 아키텍처를 사용함으로써 추구하고자 하는 적절한 품질속성이 있다. 따라서 아키텍처 스타일과 그에 의해 추구되는 품질속성의 관계를 정리할 수 있다. 표 4는 품질속성과 아키텍처 스타일의 관계를 설명한다. 표 4에서 보여주고 있는 품질속성과 아키텍처 스타일은 Mary Shaw[3], Len Bass[4], POSA[8]의 책을 기반으로 분류하였고 6가지 품질속성을 대분류로 하여 세부 품질속성을 셀 안에 정리하였다.

Layered 아키텍처 스타일은 Availability와 Modifiability를 표현하는데 적절하다. Pipe&Filter 아키텍처 스타일은 Modifiability를 표현하는데 적절하다. Blackboard 아키텍처 스타일은 Availability와 Modifiability를 표현하는데 적절한 반면에 Testability에는 적절하지 않다. Broker 아키텍처 스타일은 Modifiability를 표현하는데 적절하다. Model View Controller 아키텍처 스타일은 Usability를 표현하는데 적절하다.

3. 전체 공정

본 장에서는 아키텍처 생성을 위해 요구사항을 정량화하는 절차를 ATM 예제를 통해 설명한다. 전체 공정은 그림 4에서 볼 수 있듯이 첫째, 목표 기반의 요구사항 분석을 통해 컴포넌트를 식별한다(**컴포넌트 식별**). 둘째, 식별된 컴포넌트를 분석하여 컴포넌트의 중요도를 부여한다(**컴포넌트 중요도 부여**). 이를 바탕으로 품질속성을 추출하고 정량화 하여 전체 시스템의 품질 속성을 예측한다(**품질 속성 중요도 계산**). 셋째, 정량화된 품질 속성은 시스템에서 요구되는 품질 속성이므로, 이를 바탕으로 아키텍처 스타일을 선택하고, 마지막으로 아키텍처 스타일을 바탕으로 하여 식별된 컴포넌트로 기본적인 아키텍처를 생성한다(**아키텍처 생성**).

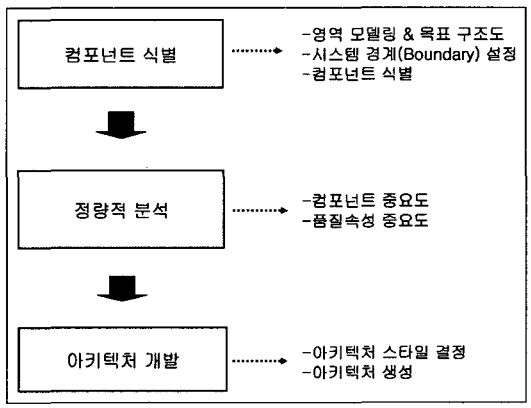

그림 4 전체 공정

3.1 컴포넌트 식별

목표 기반의 요구사항을 바탕으로 분석된 목표트리를 바탕으로 컴포넌트를 도출한다. ATM 예제 경우 그림 5와 같이 목표 트리가 생성된다.

표 4 품질속성과 아키텍처 스타일 관계

	Layers	Pipes and Filters	Blackboard	Broker	Model View Controller
A	availability ↑	Availability ↑ Error handling ↓	Fault tolerance ↑ robustness ↑	interoperability ↑ fault tolerance ↓	
M	reusability ↑ exchangeability ↑ portability ↑	Replacement ↑ reusability ↑	changeability ↑ maintainability ↑ reusability ↑	portability ↑ reusability ↑ changeability ↑ extensibility ↑	pluggable ↑ exchangeability ↑ reusability ↓
P	performance ↓	performance ↓	performance ↓	performance ↓	performance ↓
S	Security ↑				
T	testability ↑	testability ↑	testability ↓	client ↑ broker ↓	
U	usability ↑	usability ↑	usability ↑		usability ↑

Availability(A), Modifiability(M), Performance(P), Security(S), Testability(T), Usability(U)

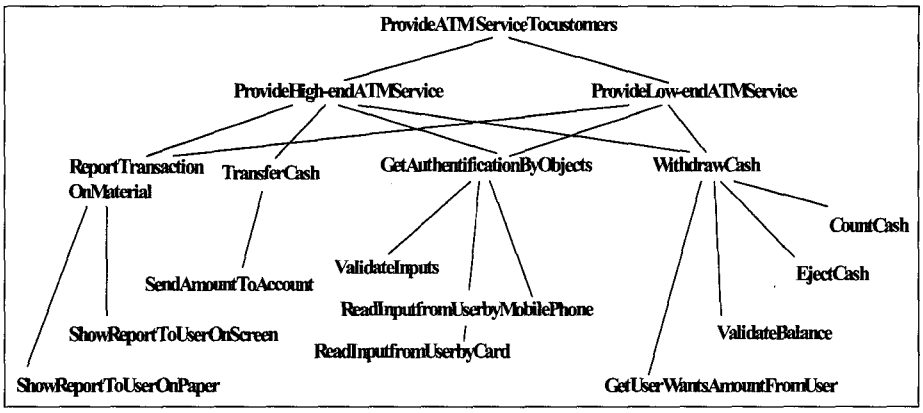

그림 5 ATM 예제에 적용한 목표 트리

목표 트리에 있는 노드(목표) 중 기능단계에서 생성된 기능 목표가 컴포넌트로 전환된다. 이는 기능 단계에 나타나는 목표들이 시스템이 내부적으로 가져야 할 기능에 중점을 두고 있기 때문이다. 그림 6은 기능 목표와 컴포넌트의 관계를 설명한다.

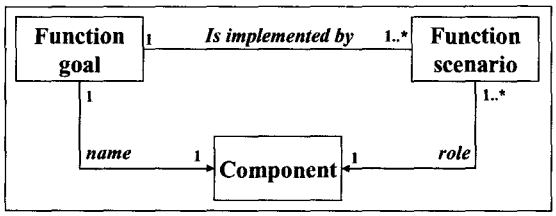

그림 6 기능 목표와 컴포넌트의 관계

기능 목표는 목표를 구체적으로 실행하기 위한 시나리오를 갖는다. 시나리오는 컴포넌트의 역할(Role)을 설명한다.

ATM 예제의 경우 다음과 같이 10개의 컴포넌트가 도출된다.

- c1: ShowReportToUserOnPaper
- c2: ShowReportToUserOnScreen
- c3: SendAmountToAccount
- c4: ValidateInputs
- c5: ReadInputFromUserByMobilePhone
- c6: ReadInputFromUserByCard
- c7: GetUserWantsAmountFromUser
- c8: ValidateBalance
- c9: EjectCash
- c10: CountCash

본 절에서 생성되는 컴포넌트는 개념적(logically)으로 시스템을 구성하는 컴포넌트를 의미한다. 즉, 플랫폼이나 하드웨어의 특성을 배제한 비즈니스 규칙에 의해 생성된 컴포넌트이다.

3.2 정량적 분석

본 절에서는 컴포넌트 중요도를 부과하는 방법과 이를 기반으로 품질 속성의 중요도를 계산하는 방법을 설명한다. 컴포넌트 중요도 계산은 기능적 측면에서 컴포넌트에 중요도를 부과하여 계산한다. 이때 AHP 기법을 이용한다. 품질 속성의 중요도 계산은 컴포넌트 중요도를 바탕으로 Matrix 기법을 사용하여 계산한다.

3.2.1 컴포넌트 중요도 계산

컴포넌트 중요도 계산은 AHP 기법을 이용한다. ATM 예제의 경우 다음과 같이 진행된다.

표 5를 바탕으로 각 열의 합으로 각 셀의 값을 나눈다. 예를 들어 c1 행의 c3 열의 값은 1/5를 6.57로 나눈 후 그 값을 c1 행의 c3 열에 입력한다. 모든 셀에 대해 작업을 마친 후에는 각 행에 대해 셀의 합을 구한다. 표 6은 컴포넌트 별 중요도를 계산한 결과를 보여준다 (step3~step5).

표 6에서 나타난 오른쪽 열에 대해 컴포넌트의 개수인 10으로 나누면 아래와 같은 결과가 도출된다.

$$\frac{1}{10}\begin{pmatrix}0.24\\0.24\\1.49\\1.33\\0.41\\0.46\\1.11\\1.96\\1.53\\1.24\end{pmatrix}=\begin{pmatrix}0.02\\0.02\\0.15\\0.13\\0.04\\0.05\\0.11\\0.20\\0.15\\0.12\end{pmatrix}$$

따라서 컴포넌트들은 다음과 같은 상대적인 중요도를 갖는다.

표 5 ATM 예제에 적용한 AHP 기법: STEP1, STEP2

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
c1	1	1	1/5	1/6	1/2	1/2	1/3	1/5	1/8	1/7
c2	1	1	1/5	1/6	1/2	1/2	1/3	1/5	1/8	1/7
c3	5	5	1	2	3	3	2	1/2	1	2
c4	6	6	1/2	1	2	2	1	1/4	3	2
c5	2	2	1/3	1/2	1	1/2	1/3	1/4	1/5	1/3
c6	2	2	1/3	1/2	2	1	1/3	1/5	1/4	1/4
c7	3	3	1/2	1	3	3	1	1/2	2	1
c8	5	5	2	4	4	5	2	1	1	1
c9	8	8	1	1/3	5	4	1/2	1	1	2
c10	7	7	1/2	1/2	3	4	1	1	1/2	1
sum	40	40	6.57	10.17	24.00	23.50	8.83	5.10	9.20	9.87

표 6 ATM 예제에 적용한 AHP 기법: STEP3, STEP4

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	SUM
c1	0.03	0.03	0.03	0.02	0.02	0.02	0.04	0.04	0.01	0.01	0.24
c2	0.03	0.03	0.03	0.02	0.02	0.02	0.04	0.04	0.01	0.01	0.24
c3	0.13	0.13	0.15	0.20	0.13	0.13	0.23	0.10	0.11	0.20	1.49
c4	0.15	0.15	0.08	0.10	0.08	0.09	0.11	0.05	0.33	0.20	1.33
c5	0.05	0.05	0.05	0.05	0.04	0.02	0.04	0.05	0.02	0.03	0.41
c6	0.05	0.05	0.05	0.05	0.08	0.04	0.04	0.04	0.03	0.03	0.46
c7	0.08	0.08	0.08	0.10	0.13	0.13	0.11	0.10	0.22	0.10	1.11
c8	0.13	0.13	0.30	0.39	0.17	0.21	0.23	0.20	0.11	0.10	1.96
c9	0.20	0.20	0.15	0.03	0.21	0.17	0.06	0.20	0.11	0.20	1.53
C10	0.18	0.18	0.08	0.05	0.13	0.17	0.11	0.20	0.05	0.10	1.24

c1 = 0.02, c2 = 0.02, c3 = 0.15, c4 = 0.13, c5 = 0.04, c6 = 0.05, c7 = 0.11, c8 = 0.20, c9 = 0.15, c10 = 0.12

위의 값에 대한 $\lambda \max = 10.9$, $CI = \frac{10.9-10}{10-1} = 0.1$ 이다. $CR = \frac{0.1}{1.49} = 0.067$ 이므로 0.10보다 작다. 따라서 쌍방비교를 통한 컴포넌트의 정량화는 신뢰할 만하다.

3.2.2 품질 속성 중요도 계산

컴포넌트의 중요도에 대한 정량화 수치를 조합하여 전체 시스템에서 필요로 하는 품질 속성의 비중을 계산한다. 본 연구에서는 [4]에서 제시한 품질 속성을 사용한다. 사용될 품질 속성은 availability, modifiability, performance, security, testability, usability이다. 이를 통해 각각의 컴포넌트들에 대해 품질 속성을 파악한다.

ATM 예제에 적용할 경우 테이블을 만들고 첫 번째 열에는 컴포넌트이름 또는 번호를 기입한다. 두 번째 열에는 컴포넌트 중요도를 입력한다. 세 번째 열부터 여덟 번째 열의 맨 위 행에 품질속성의 이름을 기술한다. 컴포넌트 별로 6개의 품질속성에 대한 상호관계성(correlation) 정도를 기입한다(표 7 참조). 마지막으로 그림

표 7 Matrix 기법을 이용한 품질속성 중요도 계산의 예

	weight	A	M	P	S	T	U
c1	0.02	0	0	0	0	0	1
c2	0.02	0	0	0	0	0	1
c3	0.15	2	1	1	2	1	0
c4	0.13	2	2	0	1	2	0
c5	0.04	1	1	0	0	0	1
c6	0.05	1	1	0	0	0	1
c7	0.11	2	0	0	2	0	1
c8	0.20	3	0	0	2	3	0
c9	0.15	3	2	2	0	0	0
c10	0.12	2	2	1	0	0	0
SUM		2.16	1.04	0.57	1.05	1.01	0.24

2에 나타난 계산 방법을 통해 각 품질 속성의 중요도를 계산한다.

예를 들어 품질 속성 Availability의 경우,

$$2.16 = 0.02\times0 + 0.02\times0 + 0.15\times2 + 0.13\times2 + 0.04\times1 + 0.05\times1 + 0.11\times2 + 0.20\times3 + 0.15\times3 + 0.12\times2$$

이다.

ATM 도메인의 경우, Availability가 2.16으로 가장 높게 계산되었으며, Security가 1.05로 계산되었다. 따라서 ATM 도메인의 아키텍처 생성시 주의해야 할 품질 속성은 Availability와 Security가 된다.

3.3 아키텍처 생성

3.2에서 식별된 시스템의 품질 속성과 중요도와 2.4에서 언급한 아키텍처 스타일을 참조하여 아키텍처 구조를 결정한다. ATM 예제의 경우 품질 속성 중요도 분석 결과 availability와 security가 높게 나타났으므로 이를 지원하는 아키텍처 스타일로써 Layered 아키텍처 스타일을 사용한다(표 4 참조). 아키텍처의 틀이 결정되고 구성요소인 컴포넌트가 생성되었으므로 이를 바탕으로 아키텍처를 생성한다.

ATM 예제에서는 Interface Layer, Control Layer, Data Layer를 갖춘 Layered 스타일과 Repository를 중심으로 상호작용을 맺는 스타일의 조합으로 이뤄진다(그림 7 참조). 아키텍처의 구조가 결정된 후에 컴포넌트의 역할(Role)을 참조하여 배치하고 컴포넌트 간의 관계를 커넥터로 연결한다. 구체적인 아키텍처를 설계하기 위해서는 본 연구의 결과물(품질속성 값, 아키텍처 스타일, 컴포넌트 목록, 컴포넌트에 대한 시나리오)과 아키텍트의 경험이 필요하다.

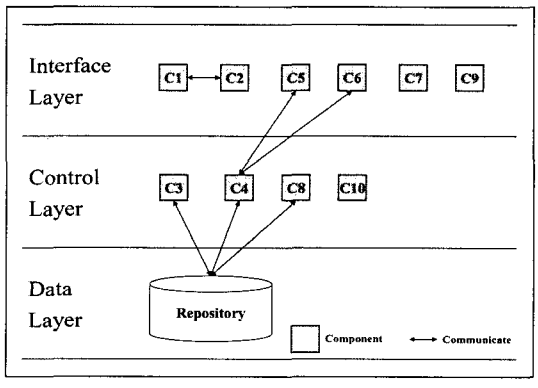

그림 7 ATM 시스템의 아키텍처

4. DRAMA 도구와 산업체 예제를 통한 검증

본 장에서는 DRAMA를 지원하는 도구와 이를 사용하여 실 예제에 적용한 결과로 본 논문에서 제안하는 정량화 기법에 대한 유용성을 검증한다.

4.1 DRAMA 도구의 개요

DRAMA 도구는 앞서 언급한 공정과 기본 개념들을 지원하는 반자동화도구이다. DRAMA 도구는 도메인 요구사항 분석기, 컴포넌트 중요도 계산기, 품질속성 중요도 정량적 분석기, 아키텍처 모델링 도구로 구성된다. 그림 8은 공정과 DRAMA 도구의 관계를 보여준다. 그림 8에서 알 수 있듯이 DRAMA 도구는 아키텍처 생성을 위해 요구사항을 정량화하기 위한 공정을 단계별로 지원한다. "요구사항 분석기"는 '컴포넌트 식별'을 지원하며, "컴포넌트 중요도 계산기"와 "품질속성 중요도 정량적 분석기"는 아키텍처 생성을 위한 입력 값으로써 요구사항을 정량적으로 분석한다. 마지막으로 "아키텍처 모델링 도구"는 요구사항에 적합한 아키텍처를 모델링할 수 있는 환경을 지원한다.

4.2 산업체 예제를 통한 검증

본 절에서는 DRAMA를 산업체에 적용한 결과와 교훈을 설명한다. 이를 통해 DRAMA가 실제 산업체에서 얼마나 실용적이고 유용한지를 검증한다. 이를 위해 IT 개발자 20명에게 DRAMA를 이해 시킨 후 3개의 실 예제에 적용하여 시스템을 개발하도록 하였다. 그 후에 실험 참가자들과 면담, 설문조사를 통해 DRAMA에 대한 평가를 정리하였다.

4.2.1 실험조건

실험 대상자들은 SI(System Integration) 업체에 종사자가 가장 많았고 그 다음이 컨설턴트였다. 그림 8은 실험 참가자들의 경력분포를 보여준다.

그림 9에서 볼 수 있듯이 실험 대상자 군은 SI 업체 38%, 컨설턴트 25%, 솔루션사업 19%, 프리랜서 6%, 은행 6%, 하드웨어 벤더 6%로 나타났으며 이는 전체 대상자의 82%가 소프트웨어 개발에 관련된 자들로 산업체 예제를 적용하기 적절한 군을 형성하고 있다. 또한 실험 대상자들의 88%가 소프트웨어 개발 경력이 3년 이상이었다. 이는 실험 대상자들이 소프트웨어 개발에 있어서 전문가 집단임을 알 수 있었다.

실험 대상 예제는 과거에 개발한 프로젝트 또는 현재 개발하고 있는 프로젝트를 실험 예제로 잡아 DRAMA를 이용하여 요구사항을 분석하고 아키텍처를 설계하도록 하였다. DRAMA를 적용한 예제는 '인터넷 e-learning 시스템', '신용카드 시스템', 'CRM 시스템'이다. 실험 후에는 적용 결과를 놓고 DRAMA에 관한 전반적인 설문조사와 예제의 결과를 통해 실험결과를 정리하였다.

4.2.2 실험결과

본 절에서는 DRAMA를 산업체 실 예제에 적용한 결과를 요약하여 정리한다. 실 개발자들이 DRAMA를 이

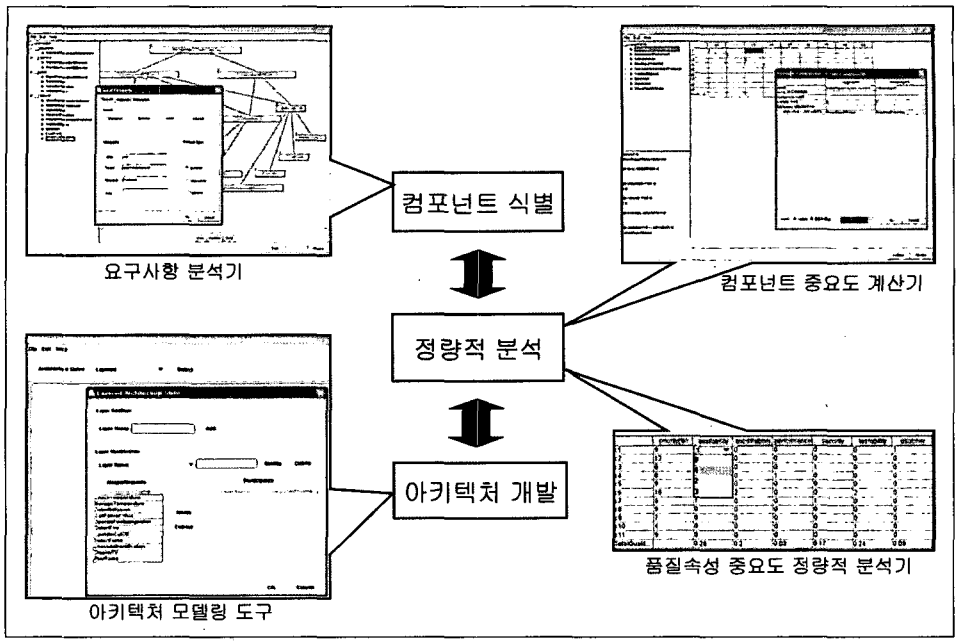

그림 8 DRAMA 도구

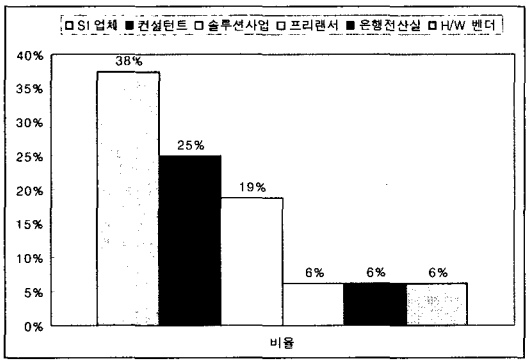

그림 9 실험자 경력분포

용해 개발한 후 그 결과를 분석하고 또한 개발자들에게 설문조사를 실시하여 정리한 결과이다. 그러므로 실험결과는 2가지 측면의 장, 단점이 있다. 첫째, 실 예제에 적용한 결과와 산업체 개발인력에게 조사한 설문이므로 DRAMA의 유효성과 효과가 실질적이라 할 수 있다. 둘째, IT 개발 인력 20명을 대상으로 진행된 실험이기 때문에 완전히 객관화 하기에는 어느 정도의 수치상의 오차가 발생할 수 있다.

실험에 참여한 20명의 인원 중 85%가 DRAMA를 통해 생성된 아키텍처가 구체적인 아키텍처 설계에 유용한 정보를 제공한다고 답했다. 이는 DRAMA를 통해 구체적인 도메인 아키텍처 설계의 프로토타입을 제공하고 있음을 알 수 있다. 그러나 DRAMA를 통해 생성된 아키텍처는 실제 설계될 아키텍처의 72% 정도만을 표현하고 있다고 답했다. 이는 아키텍처 설계에 영향을 주는 요소가 요구사항, 기술, 개발제약 등이지만 DRAMA는 기술, 개발제약과 같은 요구사항의 외적 요소를 잘 반영하지 않고 있음을 알 수 있다. DRAMA는 요구사항을 기반으로 아키텍처를 생성하는 데 초점을 두고 있는 프레임워크이기 때문에 이러한 결과가 나타났다. 이를 뒷받침 하듯이 응답자의 77%가 요구사항이 변경되었을 경우 아키텍처 변경을 예측 관리 할 수 있다고 답하고 있다. 따라서 DRAMA는 요구사항을 기반으로 하여 아키텍처를 생성하는데 있어서 유용한 프레임워크이며 이는 구체적인 도메인 아키텍처 생성에 매우 중요한 시작점을 제공한다. 다음은 세부항목에 대한 DRAMA 적용 결과이다.

컴포넌트의 우선순위 부여는 응답자의 86%가 유용하다고 답했다. 그 이유로는 중요도를 막연하게 답하는 것이 아니라 그것을 정확하게 판단하고 그 근거를 제시하여 주기 때문에 객관적으로 중요도를 적용할 수 있기 때문이라 답했다.

품질속성 정량적 계산 프로세스는 응답자의 79%만이 유용하다고 답했다. 이는 품질속성을 정량화하기 위해 필요한 품질속성을 상세하게 설명하지 못 했기 때문이다. 이는 실험 참가자들이 DRAMA를 통해 생성한 품질속성의 값에 대해 76.67% 정도만 신뢰하는 것으로도 알 수 있다. 따라서 품질속성을 보다 상세히 정의하고

실제적인 적용 매뉴얼이 필요하다. 그러나 품질속성을 휴리스틱하게 예측하기 보다는 DRAMA를 통해 정량적으로 계산할 수 있는 것은 유용하다 할 수 있다.

DRAMA를 통해 상세한 아키텍처 설계 전에 아키텍처의 프로타입을 제공하는 것이 아키텍처를 바로 설계하는 것에 비해 146% 정도 유용한 것으로 나타났다. 이는 DRAMA가 실 예제에서도 요구사항을 근간으로 하여 아키텍처를 생성하는 것이 만족시켜주고 있음을 알 수 있다.

4.2.3 교훈

DRAMA를 실 산업체의 예제에 적용해 본 결과 DRAMA 사용을 통해 요구사항 식별, 요구사항의 변화에 따른 아키텍처의 변화예측, 품질속성을 바탕으로 아키텍처를 설계하는 문제를 해결할 수 있었다. 이는 DRAMA가 이론적 측면뿐만 아니라 실제적 측면에서도 아키텍처를 생성하는데 필요한 요구사항 분석을 잘 지원함을 알 수 있다.

그러나 DRAMA를 산업체에 적용할 경우 보완해야 할 점이 몇 가지 나타났다. 예를 들어 DRAMA의 프로세스와 기법 중에서 컴포넌트의 개수가 많을 경우 AHP 기법을 적용하여 쌍방비교를 하는데 시간이 오래 걸리기 때문에 사용자가 다소 불편함을 느꼈다. 또한 DRAMA를 지원하는 도구는 undo, 버전관리, 도움말 기능이 추가적으로 필요한 것으로 나타났다.

DRAMA를 적용한 예제들은 모두가 사용자와 상호작용이 나타나는 시스템, 데이타를 처리하는 시스템, 일반적인 응용 소프트웨어 시스템인 반면에 임베디드 시스템이나 실시간 시스템과 같이 하드웨어와 긴밀한 관련을 갖는 소프트웨어 시스템에는 DRAMA를 적용하기 어려웠다. 이런 원인은 DRAMA가 아키텍처에 영향을 주는 요소 중에서 요구사항에 집중하여 아키텍처를 생성하는 기법이므로 하드웨어와 상호작용이 많은 임베디드 시스템이나 실시간 시스템에는 부적합하기 때문이다.

5. 결 론

본 논문은 요구사항으로부터 아키텍처를 객관적으로 생성하기 위해 요구사항을 정량화하는 방법(DRAMA)을 제안하였고 이를 지원하는 도구를 개발하여 산업체 예제에 적용해 보았다.

요구사항을 정량화는 목표기반의 요구사항 분석을 통해 식별된 컴포넌트에 중요도를 부과하고 이를 바탕으로 품질속성을 추출하고 정량화하는 과정을 거친다. 품질 요구사항의 정량화는 정성적인 문제를 정량적인 방법으로 해결함으로써 체계적인 의사결정과 객관적인 평가를 가능하게 한다. 이를 통해 시스템의 아키텍처 구조를 결정하기 위한 근거를 제시할 수 있고 객관적인 평가를 통해 아키텍처를 결정할 수 있다.

산업체 예제에 적용해본 결과, DRAMA는 요구사항 식별과 요구사항의 변화에 따른 아키텍처의 변화를 예측할 수 있도록 하였고 품질속성을 바탕으로 아키텍처를 설계할 수 있도록 지원하였다. 그러나 컴포넌트의 개수가 많을 경우 AHP 기법을 적용하여 쌍방비교를 하는데 시간이 오래 걸리는 단점이 있었다.

본 논문의 초점은 아키텍처 생성을 위한 기초자료를 요구사항으로부터 분석하는 것이므로 구체적인 아키텍처 생성 방법은 다루지 않았다. 따라서 제안한 아키텍처는 매우 초기 단계의 아키텍처이며 플랫폼과 하드웨어의 특성을 반영하지 않았다.

향후 연구 과제는 보다 체계적이고 정확한 방법으로 아키텍처 스타일의 품질 속성을 분석하는 작업이 필요하다. 이를 통해 정량화된 품질 요구사항으로부터 아키텍처 스타일을 선택하기 위해 학습이론(Learning theory)를 사용할 예정이다. 또한 도구의 Undo 기능, copy & paste 기능 등을 추가 보완할 예정이다.

참 고 문 헌

[1] Daniel M. Berry, et al. Foreword by the Workshop Co-Chairs, STRAW 03 in conjunction with ICSE 03, Oregon, 2003.

[2] Alan M. Davis, Software Requirements Analysis & Specification, Prentice-Hall, 1990.

[3] Mary Shaw and David Garlan, Software Architecture: Perspectives on emerging discipline, Prentice-Hall, 1996.

[4] Len Bass, Paul Clements, and Rick Kazman, Software Architecture in Practice, Addison-Wesley, 1998.

[5] Maarit Harsu, From architectural requirements to architectural design, Report 34, Institute of Software Systems, Tampere University of Technology, May 2003.

[6] Dongyun Liu, Hong Mei, Mapping Requirements to Software Architecture by Feature-Orientation, STRAW'03 Second International SofTware Requirements to Architectures Workshop, Portland, Oregen May 3-11, 2003, pp. 69-76.

[7] Lawrence Chung, Brian A. Nixon, Eric Yu, "An Approach to Building Quality into Software Architecture," IBM Centre for Advanced Studies Conference, Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research 1995.

[8] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, Pattern-oriented Software Architecture: A system of pattern, Vol. 1, Wiley, 1996.

[9] Jeongwook Kim, Jintae Kim, Sooyong Park,

Vijayan Sugumaran, A Multi-View Approach for Requirements Analysis using Goal and Scenario, Industrial Management and Data Systems Journal, Emerald publisher, Vol. 104, No. 9, 2004, pp. 702-711.

[10] 김진태, 김동선, 박수용, 목표와 시나리오 기반의 통합적 요구사항 분석 방안, 한국정보과학회 논문지, Vol. 31, No. 5, pp. 543-554.

[11] Jintae Kim, Sooyong Park, Vijayan Sugumaran, "Improving Use Case Driven Analysis using Goal and Scenario Authoring: A Linguistics-Based Approach," Data & Knowledge Engineering Journal, Available online 24 June 2005. (to be published)

[12] T.L Saty, The Analytic Hierarchy Process, McGraw-Hill, New York, 1980.

김 진 태

1998년 서강대학교 컴퓨터학(공학사). 2000년 서강대학교 컴퓨터학(공학석사). 2002년 데이콤시스템테크놀로지 (Software Engineer). 2005년 서강대학교 컴퓨터학(공학박사). 현재 삼성전자 정보통신연구소(Senior Software Engineer). 관심분야는 소프트웨어 프로덕트 라인, 요구공학, 아키텍처

양 원 석

1999년 고려대학교 전산학과(이학사). 현재 서강대학교 컴퓨터공학과(석사). 관심분야는 아키텍처, 소프트웨어 프로덕트 라인, 소프트웨어 개발비용 산정

정 창 해

2004년 서강대학교 컴퓨터학과(공학사) 현재 서강대학교 컴퓨터공학과(석사). 관심분야는 요구공학, 소프트웨어 프로덕트 라인, 소프트웨어 개발비용 산정

박 수 용

1986년 서강대학교 전자계산학(공학사) 1988년 Florida State University(전산학 석사). 1995년 George Mason University (정보기술학박사). 1998년 TRW(Senior Software Engineer). 현재 서강대학교 컴퓨터학과 부교수. 관심분야는 적응형 소프트웨어, 소프트웨어 프로덕트 라인, 요구공학, 유비쿼터스 컴퓨팅