한국산학기술학회논문지
Vol. 10, No. 8, pp. 1970-1976, 2009

UML을 응용한 GLORY 소프트웨어 아키텍처의 표현

궁상환[1*]
[1]백석대학교 정보통신학부

UML based Documentation for GLORY Software Architecture

Sang-Hwan Kung[1*]
[1]**Division of Information Telecommunication, Baekseok University**

요 약 최근 소프트웨어의 규모가 증대되고, 소프트웨어에 대한 관리가 능동적이어야 한다는 요구에 따라 소프트웨어 아키텍처의 중요성은 더욱 증대되고 있다. 소프트웨어 아키텍처는 건축물의 청사진과 마찬가지로 소프트웨어 골격의 구조에 대한 표현이 된다. 소프트웨어 구성요소와 이들간의 관계를 보다 정확하고, 종합적으로 표현하기 위해 소프트웨어 아키텍처는 다양한 뷰를 통해 명세되어진다. UML(Unified Modeling Language)은 소프트웨어 및 소프트웨어의 아키텍처를 문서화하기 위해 사용되는 모델링 도구이다. 그러나 UML은 실제 사용하는 것이 용이하지 않으며, 또한 그 표준도 지속적으로 변경된다. 도한 UML을 정확히 배워서 사용하는 것이 용이하지 않으며, 특히 도구없이는 표현하기가 매우 어렵다는 것도 사실이다. 본 연구에서는 소프트웨어 아키텍처 설계를 위한 아키텍처 뷰를 소개하고, 각각의 뷰를 설계하기 위한 UML을 소개한다. 특히 UML의 단순화된 표현을 소개하여 파워포인트와 같은 일반적인 문서화 도구를 이용하여서도 소프트웨어 아키텍처를 쉽게 문서화할 수 있는 방법을 소개한다. 또한 이 표현방법을 GLORY 시스템의 아키텍처 설계에 적용해 보고 적용효과를 분석해 본다.

Abstract It is more emphasized on the software architecture recently, as the scale of a software becomes huge and the need of the software management becomes more dynamic. Software architecture is a representation of structures of software framework just like the blueprint of building architecture. In order to describe software components and their relationships accurately and entirely, software architecture is documented in some different views, by using of modeling tools. UML(Unified Modeling Language) is a software modeling tool recently used for documentation of software and as well as software architecture. Nevertheless, what we have to agree with is that UML is not easy to use and its standard changed continuously. And also the documentation with UML is found some burden because of its difficulties in learning and using. This inconvenience enforces us to purchase and use commercial tool for UML. The study introduces the architecture views refined from 4+1 Views for architecture design and shows how to represent architecture views for software architecture. Especially, we simplifies UML diagrams for the purpose of focusing on architecture views and facile manipulation. At the end, we add the evaluation on the refined architecture views as well as refined UML diagram.

Key Words : Software Architecture, Architecture Documentation, UML, GLORY

1. 서론

소프트웨어의 개발이 착수된 후, 가장 먼저 수행되어야 하는 작업 중의 하나는 소프트웨어 아키텍처를 설계하는 작업이다. 설계된 아키텍처는 차후의 세부 설계에 대한 방향을 제시하고, 개발 이후의 확장 및 유지보수 과정에서의 소요비용을 결정하게 한다.

아키텍처의 설계결과를 문서로 표현한 것을 아키텍처 기술서라고 부른다. 아키텍처 기술서는 설계자 뿐 아니라 개발자, 고객 등 다양한 이해관계자들 사이의 의사소통 도구로 활용되기 때문에, 아키텍처 기술서는 아키텍처의 실체를 명확하게 표현하기 위하여 몇 가지 관점에서 문

*교신저자 : 궁상환(kung@bu.ac.kr)
접수일 09년 05월 18일 수정일 09년 07월 10일 게재확정일 09년 08월 19일

서화한다.

한편, 다양한 관점을 표현하는 아키텍처의 문서화에는 다음과 같은 몇 가지 문제점을 내포한다.

첫째, 아키텍처를 설계하는 표준화된 방법이 정립되어 있지 않다. 따라서 개발자들은 정형화된 아키텍처 개발절차를 준수하지 못하며, 각 단계에서의 적절한 산출물을 작성하지 못한다.

둘째, 아키텍처 문서화를 위한 통일된 표현방법을 활용하지 못한다. 따라서 회화적인 표현을 많이 사용하거나, 특히 아키텍처 설계자에 종속된 비정형적이고, 독특한 표현방법이 많이 사용한다.

셋째, 최근에는 UML의 활용되고 있는 추세에 있기는 하지만, 다이어그램의 구성이나 표현방법이 복잡하고 특히 CASE의 도움이 없이는 활용이 용이하지가 않다.

이러한 배경으로, 본 연구에서는 소프트웨어 아키텍처 문서화에 활용될 수 있는 간편한 도구를 제안하고자 한다. 이 도구는 UML을 기반으로 하며, 아키텍처 문서화에 필요한 핵심적인 뷰의 표현에 활용이 가능한 한편, 또한 범용의 문서편집기로도 작성이 가능하다는 특징을 갖는다.

2. 관련 연구

최근 소프트웨어의 규모가 방대해 짐에 따라 소프트웨어 아키텍처를 먼저 정립하고 아키텍처 베이스라인을 토대로 하여 소프트웨어를 구현하는 방법들이 많이 활용되고 있다[3].

소프트웨어 아키텍처란 소프트웨어 시스템의 구조나 구조들의 집합을 말하며, 이 구조는 소프트웨어의 구성요소와 각 요소의 가시적 특성, 그리고 요소들 간의 관계를 설명해 준다[11]. 소프트웨어 아키텍처 설계를 위한 방법론은 다양하게 제시되고 있으나, 중요한 절차나 흐름에 있어서는 방법론들 사이에 커다란 차이가 없다. 즉, 시스템이 제공해야 하는 기능적인 요구사항을 충족하기 위해 설계요소를 단계적으로 분해해 가면서 품질속성과 연관된 아키텍처 스타일을 참조하여 설계를 수행한다[6,7,9,10,12,14.

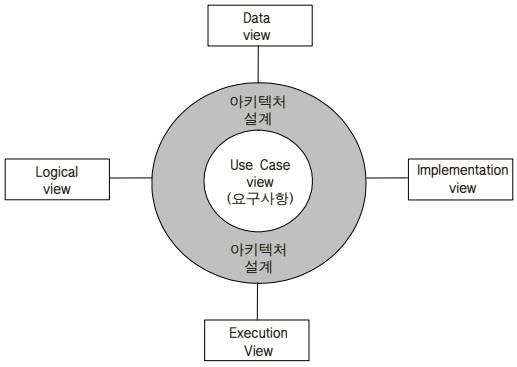
이러한 과정을 통해 나오는 아키텍처 문서를 과거에는 단순히 원이나 화살표를 사용하였지만, 최근에는 점차 표준화된 정형적인 방법을 선호하며, 이러한 정형적인 표현도구 중의 하나가 객체지향 모델링 언어로 출현한 UML이다[8,13].

3. UML을 응용한 아키텍처 표현방법의 제안

3.1 아키텍처 뷰의 모델

소프트웨어 아키텍처는 핵심적인 관점을 표현하기 위해 Philippe Kruchten의 4+1 뷰를 많이 활용한다. 4+1 뷰란 사용자 관점의 Use Case 뷰와, 설계 관점의 논리 뷰, 실행 뷰, 배치 뷰, 구현 뷰를 말한다. 그러나 이 4+1 뷰 중 배치 뷰는 구현 뷰나 실행 뷰와 함께 묘사되는 것이 타당하며, 독립적인 뷰로 보는 것은 부적절하다고 본다. 또한 4+1 뷰에서 생략하고 있는 데이터 관점의 뷰는 컴포넌트를 결정하는 중요한 요인이 되므로 중요하다.

이러한 관점에서 본 연구에서는 아키텍처 설계를 위한 4가지 뷰를 그림 1에서와 같이 데이터 뷰와 논리 뷰, 구현 뷰, 실행 뷰로 제안을 한다.



[그림 1] 개선된 아키텍처 뷰

3.2 UML 표현도구의 단순화방안 제안

UML은 기본적으로 소프트웨어 아키텍처를 구성하는 소프트웨어 요소를 표현하기 위해 다양한 다이어그램을 활용한다. 예를 들어, 논리적인 설계요소를 표현하기 위해서는 패키지 다이어그램을 이용하며, 실행모듈이나 테이블과 같은 컴포넌트를 표현하기 위해서는 컴포넌트 다이어그램을 이용하고, 객체나 쓰레드를 표현하기 위해서는 클래스 다이어그램을, 그리고 컴포넌트가 컴퓨팅 노드에 할당되는 것을 표현하기 위해서는 배치 다이어그램을 이용한다. 이러한 패키지 다이어그램이나 컴포넌트 다이어그램, 클래스 다이어그램, 그리고 배치 다이어그램은 서로 다른 모양을 하고 있고, 또 복잡해서, UML 다이어그램을 지원하는 자동화(CASE) 도구의 도움 없이는 문서화가 용이하지가 않다.

이러한 불편에 대해, 본 제안에서는 UML의 표현방법을 보다 더 간편하게 개선하여 아키텍처의 문서화에 활용하고자 한다. 특히, 이 제안은 다음의 두 가지 개념을 기초로 출발한다.

첫째, UML에서 정의하는 다양한 설계요소를 하나의 박스로 표현한다. UML에서 박스는 클래스를 의미하지만 경

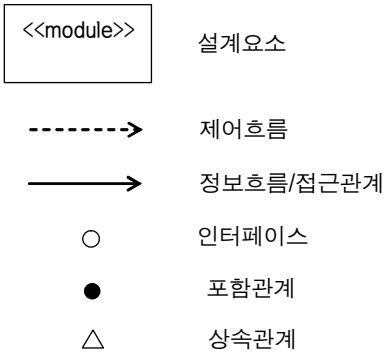
우에 따라서는 개념적인 설계요소를 표현하기도 한다.

둘째, 설계요소의 유형을 표현하기 위해서는 스테레오타입을 이용한다. UML에서 스테레오 타입은 설계요소를 상세하게 표현하거나 부족한 설명을 추가할 때 사용한다.

소프트웨어 구성요소를 표현하기 위해 박스를 사용하는 것은 전통적으로 일반화 되어 왔고, 간단한 박스를 그리는 것은 어떠한 드로잉 도구(파워포인트, 흔글 등)를 이용하여서도 매우 손쉬운 일임에 틀림이 없다. 한편, UML의 스테레오 타입은 <<include>>, <<extend>> 등과 같이 미리 정의된 예약어가 될 수도 있고, 경우에 따라서는 설계자가 필요로 하는 타입을 정의해서 손쉽게 사용하는 것도 가능하다. 예를 들어, UML에는 정의되지 않은 <<module>>, <<block>> 등을 응용분야에 필요에 맞도록 다양한 표현의 변형이 가능하도록 한다.

본 제안에서 사용하고자 하는 기본적인 표현도구를 소개하면 다음의 그림 2와 같다.

4. GLORY 아키텍처를 위한 적용



[그림 2] 기본 표현 도구

4.1 GLORY의 개요

차세대 인터넷 서비스 솔루션인 GLORY는 대규모의 고성능 서비스를 인터넷에서 제공하기 위해 개발 중인 분산 컴퓨팅 시스템이다. 특히 대용량의 단일 서버가 아니라, 방대한 컴퓨팅 클러스터(computing cluster)들의 집합체이다. 즉, 소형의 LINUX 시스템들을 십 만대 이상 연결하여 하나의 컴퓨팅 서비스를 제공하는 시스템이다.

시스템의 대표적인 서비스로는 대규모 정보검색 및 동영상 서비스 등이 있다. 특히 컴퓨팅 클러스터들이 모여 하나의 시스템으로서 인터넷 서비스를 가능하게 하기 위해, GLORY는 분산파일처리, 분산병렬처리, 분산데이터베이스와 같은 기능을 미들웨어로 제공한다[1,2].

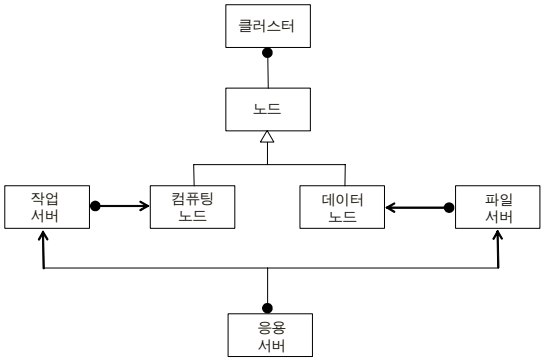
4.2 GLORY 아키텍처 뷰의 설계

GLORY는 시스템개발을 위한 기본적인 프로세스와 산출물은 각각 관련문서에 정의하고 있지만[4][5], 특별히 아키텍처를 위한 설계절차나 문서화방법에 대해서는 정의하고 있지 않다.

앞서 설명한 아키텍처 문서화 방법을 이용하여 GLORY의 아키텍처 뷰에 대한 설계를 적용해 보기로 한다.

4.2.1 데이터 뷰의 표현

GLORY는 많은 수의 노드 컴퓨터들을 묶어 하나의 가상 기계를 구성한 시스템이기 때문에, 가장 골격이 되는 데이터는 클러스터의 구성에 대한 정보가 될 수 있다. 다음의 그림 3은 GLORY의 클러스터 개념을 설명하는 가장 기본적인 모델이 되는 그림이다. 즉, 하나의 클러스터 그룹은 복수의 노드로 구성되고 있음을 설명한다. 여기서 노드는 다시 컴퓨팅 노드와 데이터 노드와 같이 더욱 구체화된 노드의 유형이 될 수 있음을 상속의 개념을 가지고 설명한다. 또한, 작업서버는 컴퓨팅 노드들의 집합을 통해서 응용으로부터 요청된 연산처리를 담당하고, 파일서버는 자료의 접근에 대한 요청을 처리하기 위해 데이터 노드들의 집합을 이용한다. 자료와 연산에 대한 처리를 동시에 요청하는 응용서버는 이 작업서버와 파일서버를 활용하여 처리를 하게 된다.



[그림 3] 데이터 뷰의 표현

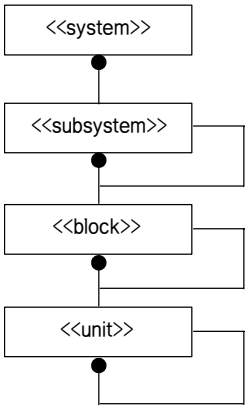
4.2.2 논리 뷰의 표현

논리 뷰는 논리적 설계요소의 상호관계를 표현하는 뷰이다. 따라서 논리적 설계요소 간의 포함관계나 사용관계가 이 뷰의 중요한 표현내용이다. 특히, 논리 뷰는 소프트웨어의 개념적인 관점을 보여 주기 때문에, 소프트웨어의 설계 초기에 아키텍처를 보여주는 중요한 표현이 된다.

통상 논리적 설계에는 UML의 패키지 다이어그램을 사용하지만, 여기서는 단순한 박스와 스테레오타입을 이용하여 정의한다. 한편, 설계요소는 그 설계요소의 타입을 설명

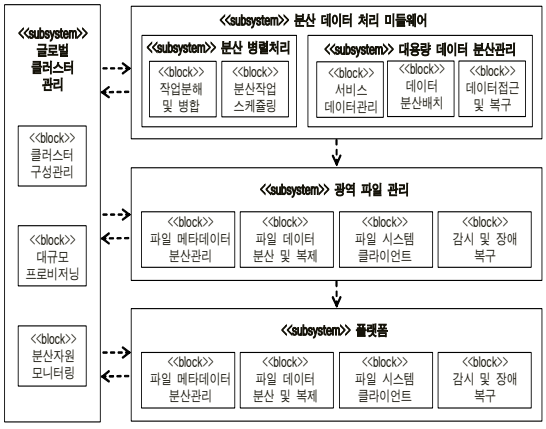
하는 스테레오 타입을 명시(예: <<subsystem>>)하고, 그 설계요소의 이름(예: 분산처리미들웨어)을 스테레오 타입 아래에 기록한다.

소프트웨어 도메인에 따라 아키텍처를 구성하는 설계요소는 다양한 규모 또는 크기에 의해 정의된다. 또한, 이러한 규모에 따른 설계요소의 명칭도 소프트웨어 개발조직에 따라 다르게 된다. GLORY는 시스템의 논리적인 구성을 위해 시스템을 서브시스템으로 분해하고, 서브시스템은 블록으로, 블록은 다시 유니트를 포함하는 계층구조를 갖는다. 또한 서브시스템이나 블록, 유니트는 동일한 레벨에서의 재귀적인 구성도 가능하다. 그림 4는 이러한 계층적 구성을 위한 모델을 보여 준다.



[그림 4] GLORY의 소프트웨어 구성단위

GLORY의 기본적인 소프트웨어 구성단위를 기초로 하여, GLORY의 시스템 구성을 서비스시스템과 블록레벨에서의 구성을 제안된 방법을 사용하여 표현하면 다음의 그림 5와 같다.



[그림 5] GLORY의 서브시스템 구성

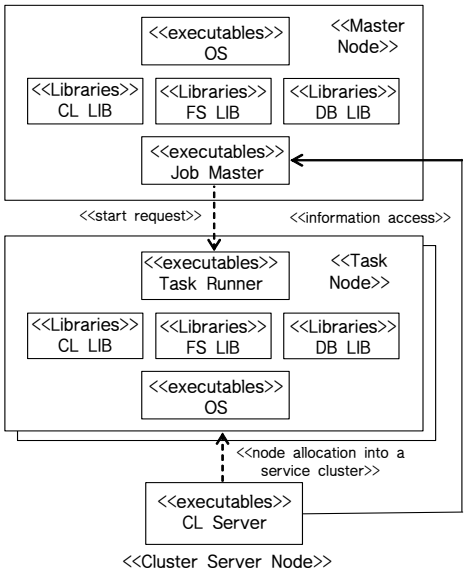
그림에서와 같이, GLORY는 크게 5가지의 서브시스템으로 구성된다. 즉, 분산데이터처리 미들웨어, 광역파일관리, 플랫폼, 글로벌 클러스터 관리, 분산보안관리 서브시스템이 있으며, 분산데이터처리 미들웨어 서브시스템은 다시 분산병렬처리 서브시스템과 대용량데이터 분산관리 서브시스템으로 구성된다. 각각의 서브시스템은 몇 개의 부속 블록으로 구성되는 데, 예를 들어 분산병렬처리 서브시스템은 작업분해 및 병합 블록과 분산작업 스케쥴링 블록으로 구성된다.

4.2.3 구현 뷰의 표현

구현 뷰는 논리적 설계요소가 프로그램으로 구현된 이후의 상태를 표현하는 뷰이다. 따라서 이 뷰는 통상 UML의 컴포넌트 다이어그램에 의해 표현되며, 설계요소에는 실행파일이나 라이브러리, 테이블이 표현된다. 이 뷰의 개선된 표현은 설계요소를 표현하는 박스 안에 설계요소의 유형을 스테레오 타입으로 기록한다. 예를 들어, 실행파일의 경우는 <<executables>>, 테이블의 경우 <<table>>을 설계요소의 이름과 함께 사용한다. 이와 같이 실행파일이나 테이블이라고 명시하는 것에 의해서도 우리는 이 설계요소가 구현 컴포넌트라고 인식하는 데 아무런 문제가 없다. 즉, 스테레오 타입의 명시만으로도 UML의 컴포넌트 다이어그램의 의미를 표현할 수 있다는 뜻이 되는 것이다.

또한, 구현 뷰는 이 뷰에 나타나는 설계요소들을 이용하여 다양하게 변형된 표현을 가질 수가 있다. 예를 들어, 설계요소들의 포함관계를 표현할 수 있고, 또한 이들 간의 접근관계, 그리고 이 설계요소들이 어느 컴퓨팅 노드에 위치하는 가를 나타내는 배치관계를 표현하는 것도 가능하다.

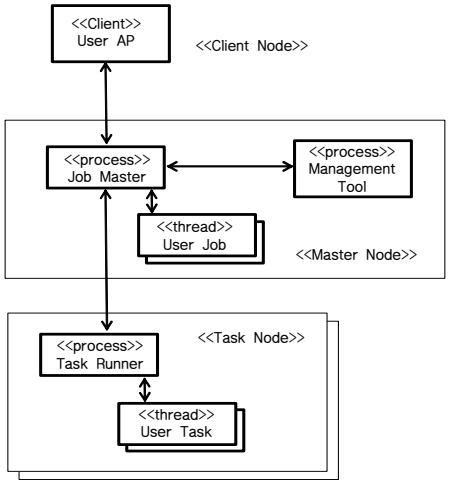
다음의 그림 6은 병렬처리와 관련된 구현 뷰의 예를 보여준다. 박스로 표시된 노드는 병렬처리의 가장 작은 단위인 태스크를 실행할 <<Task Node>>들이 있고, 이 노드들은 <<Master Node>>에 통제됨을 보여 준다. 또한 이 노드들에는 어떠한 소프트웨어 컴포넌트가 포함되어 있는 가를 보여준다. 예를 들어, 마스터 노드에는 실행파일의 형태로 된 OS와 Job Master가 있고, 클러스터와 관련된 라이브러리(CL LIB), 파일시스템을 사용하기 위한 라이브러리(FS LIB), 그리고 데이터베이스를 사용하기 위한 라이브러리(DB LIB)가 같이 포함되어 있음을 보여 준다.



[그림 6] 구현 뷰의 표현

4.2.4 실행 뷰의 표현

실행 뷰는 실행관점의 뷰이므로, 이 뷰는 실행파일이 동작 중인 모습 즉, 프로세스나 쓰레드와 같은 소프트웨어 구성요소로 표현한다. 특히, 이 뷰에서는 능동적인 요소들을 표현하기 위하여 UML의 경우와 같이 설계요소를 굵은 선의 박스로 표현한다.



[그림 7] 실행 뷰의 표현

한편, 실행 뷰의 경우도 구현 뷰와 마찬가지로, 프로세스나 쓰레드의 컴퓨팅 노드를 <<Client>>나 <<Server>>와 같이 명시하는 것이 가능하다. 다음의 그림 20은 병렬처리의 실행 뷰를 보여 준다. <<Client Node>>의 응용인 User AP는 <<Master Node>>에 있는 Job Master 프로세스의 통제 가운데, <<Task Node>>의 Task Node>>의 도움을 받아 쓰레드의 형태를 갖춰 User Task로서 동작을 수행한다. 한편, <<Master Node>>에는 태스크의 모임인 작업단위의 처리를 위해 작업별로 처리 및 통제를 위한 User Job 모듈이 쓰레드의 형태로 동작하고 있음을 보여 준다.

5. 제안방법의 평가

본 연구의 목표는 먼저 아키텍처 문서화를 위한 뷰 모델을 정의하고, GLORY 시스템의 설계를 위하여 정의된 뷰를 작성한 것이다. 특히, 뷰 작성을 위한 표현방법을 표준의 UML 표현방법으로부터 변형하여 간편한 작성이 가능하도록 한다. 따라서 제안에 대한 평가도 이 두 가지 관점에서 살펴보고자 한다.

5.1 뷰 모델의 개선에 대한 평가

본 연구에서 아키텍처 뷰의 개선은 두 가지 관점에서 제안한다. 첫째는 기존의 4+1뷰 중 설계 뷰로 데이터 뷰의 도입을 제안한다. 이것은 기존의 소프트웨어 아키텍처의 연구에서는 데이터의 관점에 대한 고려가 부족한 점을 지적하고, 중요한 데이터가 소프트웨어의 골격을 이룬다는 점을 아키텍처 모델에 반영하고자 한 것이다. 특히, 컴포넌트 기반 개발방법론도 데이터 뷰를 컴포넌트의 도출에 유용하게 활용하고, 또한 상세설계를 위한 객체의 추출에도 중요한 역할을 할 수 있다는 점을 고려한 것이다.

두 번째는 배치 뷰를 별도로 표현하지 않고, 구현 뷰 그리고 실행 뷰와 연관시켜 표현하여 배치 뷰의 표현을 생략하도록 한 점이다. 이것은 뷰의 종류를 줄인다는 장점 뿐 아니라 배치 뷰가 컴퓨팅 노드만의 표현으로는 소프트웨어 아키텍처에서 별다른 의미를 갖지 못한다는 점을 고려한 것이다.

5.2 표현방법의 정량적 평가

다음의 표 1은 비정형적인 방법과 UML에 의한 정형적인 방법, 그리고 UML을 간소화하여 제안한 표현방법을 비교하고 있다.

각 방법의 비교는 문서작성의 효율성, 작성문서에 대한 이해의 용이성, 문서의 정형화 수준, 작성법 습득의 용이성, 개발방법론의 종속성이라는 5가지 관점에서 수행하였다. 개발방법론의 종속성이란 소프트웨어의 설계에 많이 사용되는 구조적방법론과 객체지향방법론 등 특정한 방법론으로부터의 종속성을 의미한다.

비교를 위하여서는 몇 가지 측정값을 사용한다. 먼저 점수는 각각의 표현방법이 비교항목에 대해 평가된 정도를 의

미하며, 최저와 최고 사이에 1 ~ 5의 값을 갖는다. 중요도는 특정한 비교항목이 다른 비교항목에 비해 상대적으로 중요한 정도를 의미하며, 1 ~ 5의 값을 부여한다. 환산점수는 점수와 가중치의 곱으로 계산된다.

[표 1] 표현방법의 평가

비교항목	중요도	비정형적 방법		UML 표현 방법		제안된 표현방법	
		점수	환산점수	점수	환산점수	점수	환산점수
문서의 정형화 수준	5	2	10	5	25	4	20
문서이해의 용이성	5	3	15	4	20	5	25
문서작성의 효율성	4	3	12	4	16	5	20
표현법 습득의 용이성	3	5	15	3	9	4	12
개발방법론의 종속성	2	5	10	4	8	5	10
총 계			66		78		87

표 1에 제시된 내용을 비교항목을 중심으로 설명하면 다음과 같다.

- o 문서의 정형화 수준 : UML을 활용하는 경우가 다양한 표현방법이 표준으로 제시되어 정형화 수준이 가장 높다고 할 수 있다. 이에 비해 비정형적 방법은 일정한 표현방법이 없어 회화적으로 치우치기 쉬우므로 작성된 문서에 대한 통일된 이해를 갖기가 어렵다고 할 수 있다. 한편, 제안된 표현방법은 아키텍처 문서화에 필요한 필수적인 표현법을 통일하고 있지만, 세부적인 상황에 대한 표현법의 정의가 부족할 수 있다.
- o 문서이해의 용이성 : 정형화된 문서는 이해가 용이하다고 할 수도 있지만 먼저 필요한 표현법을 이해하지 못하는 경우에는 오히려 문서를 이해하는 것이 어려워진다. 한편, 비정형적 방법은 회화적 표현을 사용하여 직감적인 이해를 가질 수 있지만 오히려 이해의 정확성 측면에서는 부족할 수 있다. 이에 비해 제안된 방법은 제한된 표기법을 사용하여 이해하기가 용이하고 내용에 대한 이해도 역시 가장 높다고 할 수 있다.
- o 문서작성의 효율성 : UML 표현방법은 UML의 표현방법이 복잡하고, 특정한 편집도구를 이용하지 않고는 문서의 작성이 오히려 용이하지 않다. 비정형적 방법은 별도로 표현방법을 정의해야 하는 어려움이 있는 반면, 일반적 문서작성기로 작성이 가능하다는 장점이 있다. 이에 비해, 제안된 표현방법은 일반적인 편집도구의 사용이 가능하고, 또한 일관된 표현방법이 제공되어 문서작성이 용이하다고 할 수 있다.
- o 표현법 습득의 용이성 : UML은 표준화된 표현방법을 제공하고 있어 별도의 시간을 투자해서 표현법을 익혀야만 관련 도구를 활용하여 문서를 작성할 수 있다. 그러나 비정형적 방법의 경우는 별도의 노력 없이도 쉽게 표현법을 익힐 수 있다는 장점이 있다. 한편, 제안된 표현방법은 부분적인 표현방법의 습득으로도 아키텍처를 문서화할 수 있다는 장점을 얻을 수 있다.
- o 개발방법론의 통일성 : UML은 객체지향 방법론으로부터 출발하여, 구조적 방법론에는 적용하지 않는 경향이 있다. 구조적 설계방법에서 사용하는 다이어그램(예: DFD)과 UML 다이어그램의 가장 큰 차이점은 객체나 함수와 같은 가장 작은 설계단위에서 찾아 볼 수 있다. 그러나 아키텍처 설계에서는 최소단위의 설계요소가 모인 컴포넌트 레벨에서의 표현(문서화)을 다룬다. 특히, 본 연구에서 제안하는 방법을 활용할 경우는 이 두 가지 중 어떠한 설계개념을 적용한다 하더라도 아키텍처를 표현하는 데 문제가 없다고 하겠다.

이러한 비교의 결과를 살펴보면, 제안된 방법이 UML이나 비정형적방법에 비해 가장 적절한 방법임을 알 수 있다. 이것은 아키텍처 표현의 관점에서 비교한 결과이고, 더 구체적인 상세설계나 구현의 관점이 고려된다면 또 다른 결과를 만들 수 있을 것으로 예상한다.

6. 결론

논문의 제안은 크게 두 가지로 요약된다. 하나는 소프트웨어 아키텍처를 위한 4+1 뷰의 모델을 개선한 시도이고, 다른 하나는 UML을 개선한 정형적인 방법으로 GLORY의 아키텍처의 문서화를 실행한 것이다.

본 연구는 GLORY의 아키텍처 설계를 위하여 정형적인 표현방법을 사용하기 위하여 착수되었다. 우선 아키텍처의 문서화에 필요한 뷰를 정의하기 위하여 4+1뷰를 검토하고 보완하였으며, 이 뷰에 적합한 표현방법을 UML의 표현도구로부터 분석해 보았다. 따라서 연구에서 제안하는 표현방법은 UML의 문법과 전부 일치하지는 않지만 일관성은 유지하고 있다. UML의 요소 중 아키텍처 설계에 필수적인 요소들만을 추출하고 스테레오 타입을 이용하는 간편한 표현방법을 접목하도록 하였다. 그러나 중요한 뷰를 표현하는 데는 부족함이 없도록 하였고, 필요시 작성된 아키텍처 문서를 다시 UML 표현으로 바꾸는 것도 가능하도록 고려하였다.

참고문헌

[1] 김명준 외, 글로벌 인터넷 서비스 솔루션, 한국콘텐츠학회지, 제5권, 제1호, pp.17-22, 2007.

[2] 김명준 외, GLORY : 대규모 저가 노드 기반 글로벌 인터넷 서비스 솔루션, 한국정보처리학회지, 제14권, 제3호, pp.53-61, 2007.

[3] Dean Leffingwell, Don Widrig, Managing Software Requirements - Unified Approach, Addison-Wesley, 2001.

[4] ETRI, 인터넷서버그룹, 연구개발 표준 프로세스 3.0 산출물 양식서, Version 3.0, 1997.

[5] ETRI, 인터넷서버그룹, 연구개발 표준 프로세스 3.0 프로세스 설명서, Version 3.0, 1997.

[6] Felix Bachmann, Len Bass, Gay Chastek, Patric Donohoe, Fabio Perzzi, Architecture Based Design Method, Technical Report CMU/SEI-2000-TR-001, CMU Software Engineering Institute, 2000.

[7] Frank Buschmann, Regine Meunier, Hans Rohnert, Perter Sommerlad, Michael Stal, Pattern-Oriented Software Architecture Volume 1 : A System of Patterns, John Wiley & Sons, July, 2001.

[8] Grady Booch, James Rumbaugh, Ivar Jacobson, The Unified Modeling Language, Addison Wesley, 2005. 5.

[9] Less Bass and Rick Kazman, Architecture-Based Development, CMU Software Engineering Institute, Technical Report CMU/SEI-99-TR-007, ESC-TR-99-007, 1999.

[10] Len Bass, Mark Klein, Felix Bachmann, "Quality Attribute Design Primitives and the Attribute Driven Design Method", 4th International Wotkshop on Product Family Engineering Bilbao, Spain, 2001.

[11] Len Bass, Paul Clements, and Rick Kazman, Software Architecture in Practice, 1998.

[12] Mark Klein and Rick Kazman, Attribute-Based Architecture Style, Technical Report CMU/SEI-99-TR-022, CMU Software Engineering Institute, 1999.

[13] Paul Clements, etc, Documenting Software Architectures: Views and Beyond, Addison Wesley, 2002.

[14] Rob Wojcik and et al, Attribute-Driven Design(ADD), Version 2.0, Technical Report CMU/SEI-2006-TR-023, CMU SoftwareEngineering Institute, 2006.

궁 상 환(Kung, Sang Hwan) **[정회원]**

- 1977년 2월 : 숭실대학교 전자계산학과 졸업(이학사)
- 1983년 8월 : 고려대학교 대학원 전자정보차리학과 졸업(경영학석사)
- 1998년 2월 : 충북대학교 대학원 전자계산학과 졸업(이학박사)
- 1981년 7월 ~ 1998년 2월 : 한국전자통신연구원 책임연구원
- 20001년 3월 ~현재 : 백석대학교 정보통신학과 부교수

<관심분야>
소프트웨어 아키텍처, 분산시스템