한국산업정보학회논문지 제5권 제3호 (2000. 9)

HappyWork :소프트웨어 구조 설계 환경의 개발

(HappyWork :A Software Architecture Design Environment)

강 병 도*
(Byeong-Do Kang)

요 약 근래의 소프트웨어 관련업계에서는 소프트웨어 개발 및 관리에 있어 소프트웨어 아키텍처를 중심으로 생산성 및 품질의 극대화를 추구하고 있다. 또한 소프트웨어 컴포넌트 관련 기술 및 컴포넌트 기반 소프트웨어 개발 방법론에 대한 연구도 활발히 진행중이며, 많은 응용기술들이 나오고 있다. 소프트웨어 아키텍처는 소프트웨어를 컴포넌트로 구성하고 그 사이의 상호작용을 커넥터를 이용해 기술함으로서 전체적인 구조를 분석하고 유지하는데 필수적인 요소로 고려된다. 본 연구에서 제안하는 소프트웨어 모델링 기법은 HappyWork라는 모델링 도구를 사용하여 소프트웨어 구조를 기술하는 HappyWork Language를 생성한다. 그 과정에서 System Context Diagram, Component Diagram, Component Sequence Diagram과 같이 세 가지 다이어그램이 사용되며, User, System, Component, Connector와 같은 네 가지 Elements로 구성된다.

Abstract Recently Software Industry has tended to enhance the productivity and quality with using the software architecture in software development and administration. The research of software component technique and software development methodology are just doing and making many applications. The software architecture is considered as the essential element for analyzing and maintaining the entire structure with organizing the software into components and describing the relations with connectors. The software modeling methodology that we propose is generating HappyWork Language describing the software structure with the modeling tool as HappyWork. We can use System Context Diagram, Component Diagram, Component Sequence Diagram, and they are composed of four Elements as Users, Systems, Components, Connectors.

1. 서 론

소프트웨어 규모가 커지고 그 구성이 복잡해짐에 따라 시스템을 컴포넌트의 조합으로 해석하여 분석하고 설계하려는 노력이 계속되어 왔다. 이는 개발기간의 단축이라는 생산성 향상의 과제와 함께 컴포넌트 관련 기술이 발전함에 따라 소프트웨어 아키텍처에 대한 연구가 그 중요성을 더해가고 있다는 것을 보여준다.

소프트웨어 아키텍처를 중심으로 시스템을 설계하고 분석하는 연구는 컴포넌트 기반 소프트웨어 개발(Component Based Software Development :CBSD) 기술 중 하나로서, 시스템을 컴포넌트와 컴포넌트들간의 상호작용으로 표현한다[1]. 소프트웨어 아키텍처를 중심으로 한 개발방법은 소프트웨어를 구조적인 차원에서 심도 있게 분석하여 기능별로 컴포넌트로 구분하고 그 관계를 커넥터를 통해 기술함으로서 CBSD의 여러 기술 중 가장 주목받고 있다[2][3].

소프트웨어 아키텍처에 대한 중요한 세 가지 개념은 첫째, 아키텍처 기술 언어(Architecture Description Languages)와 도구(Toolkits)는 소프트웨어 아키텍처에 대해 표현하거나, 문서화하거나, 또는 추론해 내는데 있어 유용하고, 둘째, 소프트웨어는 서로 다른 특징들을 가지는 각각의 아키텍처 수준(Architecture levels)이 있고, 아키텍처 기술 언어는 각 단계마다 흥미 있고, 유용한 분석 도구들을 제공한다는 것과, 셋째, 소프트웨어 아키텍처에 대한 연구는 끊임없이 발전하고 있고, 계속적으로 더욱 강력하고

* 대구대학교 컴퓨터정보공학부

효과적인 분석 및 생성도구가 생겨나고 있다는 것이다 [4][5]. 이것은 소프트웨어 아키텍처 모델링이 설계 목적을 분명히 나타낼 수 있고, 설계 시 기초적인 분석을 제공할 수 있으며, 유지관리 및 가독성을 향상시킨다는 것을 의미한다.

따라서 본 논문에서는 소프트웨어 아키텍처를 이용하여 소프트웨어를 모델링하기 위해 System Context Diagram, Component Diagram, Component Sequence Diagram 과 같은 세 가지 다이어그램을 제시한다. 이 세 가지 다이어그램은 User, System, Component, Connector와 같은 소프트웨어 표현양식을 그래프 형태로 나타내며, 기능 및 분석을 위한 Properties를 제공한다.

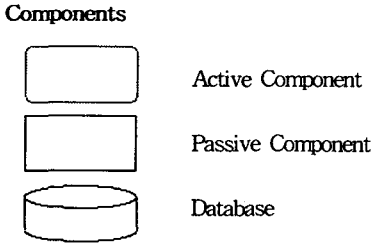
2. 소프트웨어 아키텍처

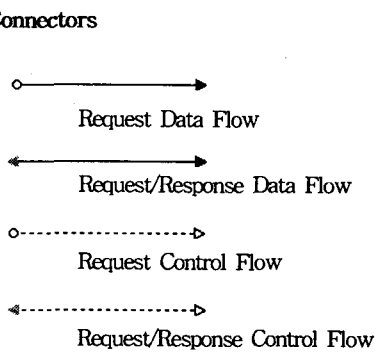
소프트웨어 아키텍처를 일관성 있는 구성으로 표현한다는 것은 매우 어려운 일이다. 이것은 완전히 상이한 아키텍처들을 기술할 만한 공통의 언어가 없고, 아키텍처를 완벽하게 기술한다는 것도 불가능한 일이기 때문이다. 그러나 가능한 소프트웨어 아키텍처들의 공통특성(기능성, 이식성, 재사용성, 유지관리 등)을 찾고, 아키텍처 유형을 분석하여 이해하기 쉽게 기술하고자 한다.

소프트웨어 시스템의 아키텍처 디자인은 구조. 기능. 관계. 이렇게 세 가지 관점으로 구분할 수 있으며, 소프트웨어공학론에서 바라보는 입장이기도 하다[6][7][8].

2.1. 구조

소프트웨어 구조는 소프트웨어를 구성하는 작은 개체들과 그것들 간의 연결로서 나타내어진다. 즉 연산을 수행하는 실체인 컴포넌트와 컴포넌트간의 상호작용을 중재하는 커넥터로 표현할 수 있으며, 우리는 아래 [그림 1]과 같이 기본적인 두 가지 요소들로 아키텍처를 구성하려고 한다.





[그림 1] 아키텍처 구성 기본요소

Active Component는 컴포넌트가 Request할 수 있는 능동적인 객체임을 뜻하고, Passive Component는 파일과 같이 스스로 연산이나 제어와 같은 작업을 수행할 수 없는 수동적인 객체임을 나타낸다. Database의 경우 Passive Data의 저장소로 표현된다.

커넥터의 Request Data Flow 속성은 단방향 Data 전송을 나타내고, Request/Response Data Flow는 양방향 Data 전송을 표현한다. Request Control Flow는 단방향 Control 전송을 나타내고, Request/Response Control Flow는 양방향 Control 전송을 표현한다.

위의 아키텍처 구성 기본요소들은 이 논문이 제안하는 소프트웨어 모델링 연구의 중심으로 작용되는 것이지만 소프트웨어 구조를 완벽하게 표현하는 것은 아니다. 구성요소의 기능 및 구성요소들간의 관계를 명시함으로서 소프트웨어를 실체화할 수 있다.

2.2. 기능

소프트웨어의 기능을 분석한다는 것은 그 소프트웨어가 무엇을 할 수 있는가를 나타내는 것이다. 기능은 시스템 전체의 기능과 요소들 각각의 기능으로 구분할 수 있다.

소프트웨어는 하향식 또는 상향식 설계방법을 통해 프로그램을 세분화할 수 있는데, 하향식(Top-down) 설계방법은 문제를 세분화하여 모듈화된 프로그램 설계구조를 유도해 나가는 비정형적 설계방법이다. 상향식(Bottom-up) 설계방법은 소프트웨어 계층구조의 최하위부터 점진적으로 모듈들을 통합시켜 나가는 방법이다.

소프트웨어를 하향식 또는 상향식의 어떤 방법으로든 기능별 구분을 하여야 하고 기능별로 우리는 컴포넌트를 구성할 수 있다. 하지만 이것은 적용영역에 따라 적절한 분석방법이 요구되는 매우 신중한 작업이며, 소프트웨어의 전체적인 구조를 형성하는 가장 기초적인 작업이라 할 수

있다.

HappyWork는 이를 위하여 일반적인 컴포넌트를 세분화하여 System과 Component로 나누어 아키텍처를 설계하도록 유도하고 있다. 이것은 소프트웨어를 계층화된 구조로 표현할 수 있도록 한 것이다. 즉, System은 Subsystem 또는 Component 및 Connector로 구성된 하위 계층을 포함할 수 있다. System은 특수한 기능을 수행하기 위해 구성된 일련의 요소들의 집합이라 할 수 있으며, System의 경계를 정의하는 것은 System 내부 요소와 외부 요소를 식별하고 System의 기능을 정의하는 것이므로 매우 중요하다. 따라서 System은 그룹화 및 계층화를 통한 정보의 은닉을 도모한다.

2.3. 관계

기능별로 소프트웨어 구조를 형성하고 각각의 요소들간의 관계를 정의한다는 것은 소프트웨어 아키텍처를 실체화하는 것이다. 이것은 시간에 따른, 또는 이벤트에 따른 소프트웨어의 동작을 정의하는 일로, 인간에 비유하자면 기능별로 머리와 팔과 다리 몸을 형성하고 구조적으로 이것을 하나로 연결했다면 관계는 신경체계를 유지하는 것이라 할 수 있다. 따라서 설계자는 요소들간의 상호관계라든지 제약사항 등을 고려하여 소프트웨어를 모델링 해야할 것이다.

HappyWork에서는 이를 위하여 각각의 구성요소들에 Properties를 제공하고 구성요소들간의 관계를 명시하도록 하고 있다. 시스템 전체에 대한 제어구조는 커넥터를 통하여 이루어지며, 컴포넌트간의 상호작용 문제, 통신 프로토콜, 동기화 문제, 자료접근에 대한 제약 및 방법의 문제, 오류인식 및 복구의 기능, 성능개선의 문제 등을 표현한다.

3. 소프트웨어 아키텍처 모델링

소프트웨어 아키텍처를 중심으로 한 모델링 연구의 기원은 Edger Dijkstra의 소프트웨어를 작은 단위로 나누어 구성하는 연구로 볼 수 있다. 이러한 연구는 Panas의 정보은닉모듈(Information Hiding Module)등의 연구로 이어져 지금의 소프트웨어 아키텍처 모델링의 형태로 발전해 왔다.

하지만, 소프트웨어 아키텍처에 대한 연구는 컴포넌트 기반 소프트웨어 개발 방법론의 등장과 함께 새로운 시작을 맞게 되었는데, 기존의 객체지향 개발 방법에서의 설계 과정에서 각 클래스 사이의 연관관계로 표현되는 것들이 컴포넌트 기반 소프트웨어 개발에서는 더욱 복잡한 관계 기술을 필요로 하였다. 이러한 요구에 따라 소프트웨어 아키텍처는 소프트웨어 전반에 대한 설계에 해당하는 역할을 수행한다[9].

3.1. HappyWork의 특성

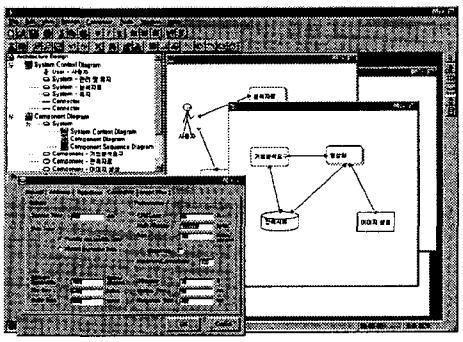
HappyWork는 소프트웨어 아키텍처를 이용한 소프트웨어 모델링 도구로서, 소프트웨어 구조를 설계하고, 모형화하기 위한 기능 및 환경을 제공하며, 완성된 모델은 HappyWork Language를 자동으로 생성할 수 있다.

HappyWork는 두 가지 측면에서 강점을 가지고 있는데, 첫째로 소프트웨어 구조의 모델링 측면에서는 새로운 기능인 디자인과 다이어그램의 지원을 들 수 있다. 이것은 소프트웨어 구조를 기술하고, 이를 기능별로 구체화하기 위한 세 가지 다이어그램을 제시함으로서 소프트웨어의 정적인 면과 동적인 면을 표현한다. 또한 소프트웨어 구조를 구성하는 네 가지 요소를 제시함으로서 소프트웨어를 구조적인 차원에서 모델링 한다. 둘째로 HappyWork Language와의 통합된 환경이다. HappyWork Language는 분산환경에서의 소프트웨어 구조를 명세하기 위한 언어이며, HappyWork의 코드 생성기를 통해서 만들어진다. 이것은 소프트웨어를 구조적인 면과 기능적인 면으로 구분하여 소프트웨어를 분석하고 이해하기 쉽도록 도와준다.

3.2. HappyWork의 기능

HappyWork는 소프트웨어를 표현하는 각각의 다이어그램을 제공하고 있으며, 다이어그램을 구성하는 User, System, Component, Connector를 이용하여 소프트웨어를 모델링 하는 간편한 인터페이스를 제공하므로, 표기법의 교육 및 이해가 쉽게 되어 있다. 그리고 소프트웨어를 구성하는 각각의 요소들에 있어, 일반적인 패턴들 즉, General Properties, Attributes, Operations, Constraints, Import Files를 기술할 수 있도록 하여 소프트웨어를 분석하고 평가하는데 도움을 주고 있다.

[그림 2]는 분산환경에서의 소프트웨어 구조 설계를 지원하고 모형화 하는 HappyWork 소프트웨어 개발환경이다.



[그림 2] HappyWork 소프트웨어 모델링 환경

3.3. 아키텍처 모델링 방법

HappyWork를 이용한 아키텍처 모델링 방법은 6단계로 나누어진다. 하지만 모든 시스템이 6단계로 구분한 개발 순서에 일치하지 않을 수도 있다. 시스템의 특성이나 복잡도에 의해 더욱 구체화될 필요성이 있거나, 중요하지 않은 단계도 있을 수 있다. 하지만 이것은 많은 객체지향 개발 방법론의 장점을 적용하고 있으며, 일반적인 소프트웨어의 개발방법에 비추어 볼 때 다음의 6단계에 대해 비교적 일관성이 유지될 수 있다.

3.3.1. 요구분석

요구분석은 시스템 개발의 목표와 기능을 결정하는 단계다. 분석의 근본적인 목적은 사용자가 요구하는 문제에 대해 개발자 측면에서 정확하게 파악하여 각각의 문제에 대해 이해하기 쉽고 원활히 접근하고자 하는데 있다. 즉 실생활을 시스템에 적용하려는 것이다.

세부단계

1) 사용자의 요구 및 제약사항을 결정한다.
2) 서비스의 형태 및 방법을 구상한다.
3) 필수 요구사항과 선택 요구사항을 구분하고, 요구의 순위를 결정한다.
4) 서비스의 순서 등 기본적인 문제들을 명확히 한다.

3.3.2. System 설계

System 설계는 소프트웨어 전반에 걸친 System 구현방법 및 적용문제를 결정하는 단계다. System의 구조뿐만 아니라 System의 기능 및 제약사항, 상호관계 등도 고려된다.

세부단계

1) System의 형태, 기능, 비용, 특성을 결정한다.
2) System의 수행능력, 문제발생 요인을 예측한다.
3) Hardware의 성능, 데이터 저장공간, 외부와의 연결 등 Hardware 조건을 고려한다.
4) 기존의 System과의 관계를 표현한다.
5) System 최적화를 위한 고려사항을 점검한다.

3.3.3. Component 설계

Component 설계시 가장 중요한 것은 실세계의 모델을 컴퓨팅 환경에 구현하기에 가장 적합한 모델로 변환시키는 것이다. 이번 단계는 여러 Attributes, Operations, Interfaces 및 Constraints를 가지는 Component를 생성한다.

세부단계

1)요구분석 단계에서의 기능들을 Component의 Operations에 추가한다.
2) Component간의 상호관계를 정의한다.
3) Component의 Properties를 결정한다.
4) 제약사항을 명시하고 일관성을 확인한다.

3.3.4. 기능분석 및 성능평가

소프트웨어 기능을 분석하고 일관성, 완전성, 신뢰도 및 수행능력을 평가한다.

세부단계

1) 일련의 기능들에 대해 테스트한다.
2) 오류에 대해 어떻게 처리하는지 살핀다.
3) 소프트웨어 전체 효율과 지연시간은 얼마인지 확인한다.
4) 문제점을 제거하고 Design을 최적화 한다.

3.3.5. 개발

지금까지 단계의 분석 및 설계를 바탕으로 소프트웨어를 구현한다.

세부단계

1) 각 단계에서의 구성을 바탕으로 소프트웨어 구현에 적용한다.
2) 제품을 위한 Package를 작성한다.

3.3.6 유지보수 및 재사용

유지보수 및 재사용은 시스템 분석 및 설계단계에 의해 그 효과가 큰 차이로 나타난다. 즉 시스템 분석 및 설계단계에서 잘 구성된 시스템은 유지보수 비용을 절감하고 재사용 가능성이 증대되며, 시스템 가독성을 향상시킨다.

세부단계

1) 결점 및 보완점을 찾는다.
2) 새로운 시스템 구축시 재구성하거나 참조한다.

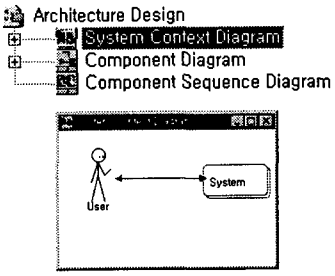
3.4. HappyWork의 다이어그램

HappyWork는 소프트웨어 아키텍처를 기술하기 위해 세 가지 다이어그램을 사용한다.

- System Context Diagram
- Component Diagram
- Component Sequence Diagram

각각의 다이어그램은 소프트웨어 개발 전반에 걸쳐 서로 다른 소프트웨어에 대한 표현법을 제공하여 소프트웨어를 구조적이면서 동적 및 정적인 기능으로 표현한다. 모든 다이어그램은 서로 연관성을 지니며, 상호간의 관계 및 명령의 흐름을 나타내지만 각각의 다이어그램의 역할은 엄격하게 분리되어 있다. 소프트웨어의 구조적인 모델링과 소프트웨어 재사용 및 분석, 관리를 위해 위의 세 가지 다이어그램은 중요한 역할을 담당한다.

3.4.1. System Context Diagram

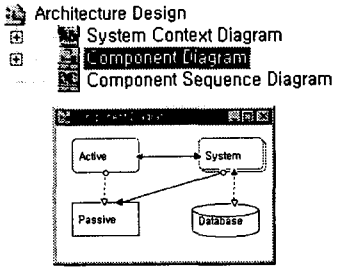


[그림 3] System Context Diagram

System Context Diagram은 사용자, 설계자 및 개발자의 관점에서 바라본 소프트웨어 구조의 표현 양식을 그래프 형태로 나타낸 것이다. 사용자 및 다른 시스템과의 관계를 나타내며 현재 개발하는 시스템의 기능을 표현한다. 그리고 이것은 Component Diagram, Component Sequence Diagram으로 구체화될 수 있으며, 다음과 같은 구성요소를 포함한다.

- User
- System
- Connector

3.4.2. Component Diagram

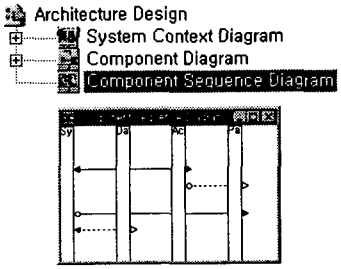


[그림 4] Component Diagram

System Context Diagram에서 구성한 User와 System간의 관계를 더욱 구체화한 것으로, Component간의 상호작용을 표현한다. 이 다이어그램은 상호 작용하는 Component간에 전달되는 메시지들을 통해 그 연관성을 분석할 수 있다. 따라서 Component Diagram은 System의 정적인 부분을 표현하면서도 내부적으로 동적인 부분을 표현한다. 이 다이어그램을 통해 시스템의 전체적인 구성을 살펴볼 수 있으며, 다음과 같은 구성요소를 포함한다.

- System
- Component
- Connector

3.4.3 Component Sequence Diagram



[그림 5] Component Sequence Diagram

Component Sequence Diagram은 Component의 상호작용을 순차적으로 보여준다. 이 다이어그램은 시간의 흐름에 따라 메시지들이 Component간에 어떻게 전달되는지를 나타낸다. 이것은 시스템 내의 통신을 시각화하는데 사용될 수 있으며, 이해하기가 쉽게 표현되고 있다. 그리고 다음과 같은 구성요소를 포함한다.

- System
- Component
- Connector

3.5. HWL(HappyWork Language)

HappyWork는 범용 컴퓨팅 환경에 대한 아키텍처를 기술하는 언어인 HappyWork 언어를 생성한다. HWL은 시스템의 기능과 구조적 특성을 기반으로 하여 패턴을 정형화하고 구성 및 속성을 구체화한다. HappyWork의 설계도구를 이용하여 소프트웨어의 아키텍처를 디자인하고 코드 생성기를 통해 HWL를 생성할 수도 있으며, 또한 일반 편집기 또는 HappyWork가 제공하는 편집기를 통해 디자인 및 수정할 수도 있다.

HappyWork로부터 구체화된 아키텍처가 실제로 구현되기 위해서는 하드웨어 구성과 소프트웨어 아키텍처간의 관계를 정확히 정의하는 과정이 필요하다. 지금의 HWL은 소프트웨어 아키텍처에 대한 기본적인 명세로 쓰일 수 있으며, 구현으로 실체화되기에는 아직 미흡하다.

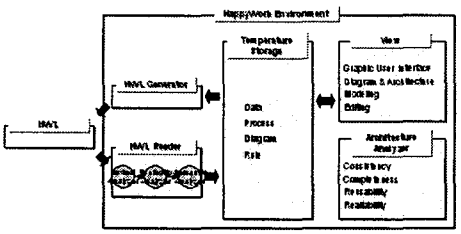
다음 [그림 6]은 HWL 구문의 간단한 예이다.

```
ComponentDiagram Sample
{
        Component RequestAnalysis Type Active
        {
                Port p1
                Port p2

                Attributes
                {
                        int MaxRequest
                        long Count
                        char[100] ResultData
                }

                Operations
                {
                        char[20] Response(long RequestID)
                        int GetCurrentRequest()
                        int GetServiceNum()
                        int Connect()
                }

                Constraints
                {
                        CPUload = 70
                        MainMemory = 3000
                        Rate = 120
                        Streaming = False
                        MaximumConnections = 100
                                :
```

[그림 6] HappyWork Language

3.6. HappyWork 구조

HappyWork는 [그림 7]과 같이 View, Architecture Analyzer, Temperature Storage, HWL Generator, HWL Reader 이렇게 5개의 기능으로 구분할 수 있으며, View는 소프트웨어 아키텍처 설계를 위해 모델링 도구 및 편집기 등 그래픽 사용자 인터페이스를 제공하며, Architecture Analyzer는 Architecture의 오류 검출 및 일관성, 완전성, 재사용성, 신뢰성을 분석하고, Temperature Storage는 설계 및 분석을 위해 임시저장소를 제공하며, HWL Generator와 HWL Reader는 각각 HappyWork 언어를 생성하거나 번역한다.



[그림 7] HappyWork 구조

4. 아키텍처 성능분석

일반적으로 성능이란 사용자의 시각에서 평가하는 시스템의 반응시간(Response Time)이나 처리율(Throughput)을 말하며, 시스템을 이용하는 사용자의 효과 및 생산성 측면에서 정의된다.

시스템의 성능을 표현할 수 있는 척도들은 다양하다. 편의성, 가독성, 구조성, 정확성 등 구체화시키기 힘든 질적 척도들이 있는가 하면, 시스템 성능평가의 중심이 되는 계수 척도들도 있다[10]. 계수척도들은 다음과 같다.

① 생산성(Productivity) : 주어진 단위시간 내에 시스템에 의해 처리되는 정보량을 뜻한다. 처리율(Throughput Rate), 생산률(Production Rate), 용량(Capacity: 최대처리율), 명령어 수행률(Instruction Execution Rate), 자료처리율(Data Processing Rate) 등으로 표현된다.

② 응답성(Responsiveness) : 시스템에 입력이 주어진 시각으로부터 시스템이 이와 관련된 출력을 생성하기 시작한 시각까지의 시간을 뜻한다. 응답시간(Response Time), 총 처리시간(Turnaround Time) 및 반응시간(Reaction Time) 등으로 설명된다.

③ 활용성(Utilization) : 주어진 시간동안 시스템의 특정요소가 이용되는 정도를 말한다. 적용영역에 따라 다양한 요소들이 있다.

소프트웨어 아키텍처 차원에서의 시스템 평가는 조금은 다른 시각에서 이루어진다. 여기서 성능이라 하면 시스템을 구성하는 요소들간의 연관성 및 효율성 측면에서 평가된다. 따라서 소프트웨어 아키텍처 분석을 위한 척도들은 다음과 같다.

① 일관성(Consistency) : 구성 및 표현이 서로 잘 일치하고 있는지를 살핀다. 컴포넌트의 구성과 커넥터의 프로토콜간의 문제가 서로 잘 대응되는지를 분석한다.

② 완전성(Completeness) : 요소들의 기술에서 누락된 것이 없는지를 살핀다. 즉 심각한 오류는 없는지, 또는 오류가 나타날 경우에 적절히 대처하고 있는지 평가한다.

③ 재사용성(Reusability) : 전체 시스템 또는 구성요소가 다른 것으로 재구성되거나, 대치될 수 있는지를 분석한다.

④ 신뢰성(Reliability) : 시스템 전반의 작용 및 성능을 평가한다.

이러한 척도들은 기존의 분석 언어나 도구들을 이용하여 평가할 수 있으며, 몇몇 아키텍처 기술 언어들(Architecture Description Languages)은 효과적인 분석을 위한 분석도구들을 제공하고 있다.

HappyWork는 소프트웨어 아키텍처 구성을 대한 오류를 검출하고, 시간의 흐름에 따른 순차적인 데이터의 흐름 및 이벤트의 이동을 감지하며, 구성요소들의 반응을 통한 성

능분석 및 신뢰성을 평가해 볼 수 있다. 또한, 아키텍처를 기능별 세분화 및 계층화를 통해 코드 재사용의 효과를 극대화하고 있다.

5. 결론 및 향후연구과제

이 논문에서 우리는 소프트웨어 아키텍처를 이용한 소프트웨어 모델링 기법과, HappyWork라는 모델링 및 분석 도구를 살펴보았다. 때때로 아키텍처 기술 언어 및 분석도구들은 RAD(Rapid Application Development) 도구들과 비교가 되지만 각각 소프트웨어를 해석하는 시각을 달리하여 상호보완의 관계로 발전하고 있다.

HappyWork는 시스템의 생명주기에 따른 구성 및 요구사항을 표현하고 분석하는 일반적인 기법과 기능을 수용한 도구이다. 또한 소프트웨어 아키텍처에 대한 기본구조에 충실하였고, 아키텍처 명세를 위한 일반화된 구성요소들을 도입하였다. 그리고 편의성을 위해 사용자 인터페이스를 그래픽 중심으로 개발하였고, 아키텍처 분석을 위해 분석 도구를 제공한다. 우리는 이러한 환경을 이용해 어떻게 소프트웨어 아키텍처가 구성되어 있는지, 어떤 기능을 수행하는지, 얼마만큼의 성능을 기대할 수 있는지를 평가할 수 있다. 그러나 소프트웨어의 품질개선을 위한 최적화 된 방안을 제시하지 못하며, 사용자 및 설계자의 요구에 따른 시스템 구성을 자동화하고 있지 않다.

결론적으로, 소프트웨어 아키텍처를 가장 잘 표현할 수 있는 표준화된 구조 및 기능 분류가 요구되며, 새로운 운용환경에 적절히 대처할 수 있고, 동적인 수행능력 및 기능평가를 위해 향후 과제로 연구하고 있다. 또한 소프트웨어 아키텍처 분석을 위한 많은 연구가 계속되고 새롭고 향상된 분석 및 개발도구가 실현되기를 기대한다.

참 고 문 헌

[1] P. C. Clements, L. M. Northrop, "Software Architecture: An Executive Overview", Technical Report, CMU/SEI-96-TR-003, February, 1996.

[2] Robert T. Monroe, "Modeling and Analysis of Software Architecture", ICSE 99 Tutorial, 1999.

[3] Paul Kogut, Paul Clements, "Software Architecture Renaissance", Crosstalk, November 1994.

[4] LTC Erik Mettala, Mark H. Graham, "The Domain-Specific Software Architecture Program", Proceeding of the DARPA Software Technology Conference. Defense Advanced Research Projects Agency, April 1992.

[5] Paul Kogut, Paul Clements, "Features of Architecture Representation Language(Draft)", Software Engineering Institute Technical Report, 1995.

[6] D. Garlan, M. Shaw, "An Introduction to Software Architecture", Advances in Software Engineering and Knowledge Engineering, Volume I, World Scientific Publishing, 1993.

[7] R. Pressman, "Software Engineering: A Practitioner's Approach", 3rd edition, New York: McGraw-Hill, 1992.

[8] I. Sommerville, "Software Engineering, 4th edition, Reading MA: Addison-Wesley, 1992.

[9] Rick Kazman, Len Bass, Gregory Abowd, Mike Webb, "SAAM: A Method for Analyzing the Properties of Software Architectures", Proceedings of ICSE 16, May 1994, pp. 81-90.

[10] 이주헌, "실용 소프트웨어 공학론", 법영사, 1993.

강 병 도

1986년 서울대학교 계산통계학과 졸업(이학사)
1988년 서울대학교 대학원 이학석사(전산과학 전공)
1995년 서울대학교 대학원 이학박사(전산과학 전공)
1988년 6월 ~ 1998년 2월 한국전자통신연구원 선임연구원
1998년 3월 ~ 현재 대구대학교 교수로 재직중
관심분야 : 소프트웨어 개발론, 소프트웨어 구조, 소프트웨어 프로세스 등