한국산업정보학회논문지 제6권 제4호 (2001. 12)

실시간 시스템을 위한 소프트웨어 구조 설계 환경†

(A Software Architecture Design Environment for Real-time Systems)

강 병 도*
(Byeong-Do Kang)

요 약 소프트웨어 구조는 시스템 컴포넌트들과 그들의 상호관계를 이해하기 위한 프레임워크로서 제공된다. 또한 소프트웨어 구조는 낮은 비용, 높은 생산성 및 일관된 품질을 이루기 위해 재사용 될 수 있다. 본 논문에서는 우리가 개발한 소프트웨어 구조 설계 환경인 HappyWork의 구조와 기능을 설명한다. HappyWork는 두 가지의 주요한 기능을 가진다. 첫째로 HappyWork는 소프트웨어 구조 다이어그램의 모델링을 위한 그래픽 에디터를 제공하고, 둘째로 HWL(HappyWork language)라는 소프트웨어 구조 기술 언어를 제공한다.

Abstract Software architecture serves as a framework for understanding system components and their interrelationships. Software architectures can be reusable assets to achieve low costs, high productivity, and consistent quality. We have developed a software architecture design environment, called HappyWork. In this paper, we would like to present the structure and functions of HappyWork. HappyWork has two main functions. First, it provides a graphic editor for modeling of software architecture diagram. Second, it provides an ADL, called HWL(HappyWork language). HWL is a language that describes software architecture.

1. 서 론

소프트웨어 재사용의 부족은 소프트웨어 개발자로 하여금 낮은 비용, 높은 생산성 및 일관된 품질을 이루기 어렵게 한다. 재사용의 가능성은 소프트웨어 구조 단계에서의 명세를 최소화 할 때 가장 크다. 소프트웨어 구조는 시간에 대해 비교적 일정한 시스템 특성을 포함하기 때문에, 구조는 시스템 버전들 사이에서 일관되고 같은 영역내의 교차된 어플리케이션은 재사용될 수 있다.

소프트웨어 구조는 그들 자체로 재사용될 수 있을 뿐만 아니라 표준화된 인터페이스, 프로토콜, 추상화 단계에서의 기능 패키지를 통하여 설계 및 코드 컴포넌트의 재사용도 지원한다.

본 논문에서는 우리가 개발한 소프트웨어 구조 설계 환경인 HappyWork의 구조와 기능을 설명한다. 2장은 소프트웨어 구조의 개념을 설명하고, 3장은 소프트웨어 구조에 대한 몇 가지 관련연구를 소개한다. 4장은 HappyWork의 환경을, 5장은 HappyWork와 관련연구 사이의 특징 비교를 보여주며, 마지막으로 6장에서 결론을 맺는다.

2. 소프트웨어 구조

소프트웨어의 크기와 복잡성이 증가함에 따라, 시스템 전체 구조의 설계와 명세는 알고리즘과 자료구조의 선택

* 대구대학교 컴퓨터정보공학부
† 이 논문은 2000년도 대구대학교 학술연구비 지원에 의한 것임.

보다 더 중요한 문제가 되었다. 구조적 문제는 컴포넌트 합성으로서의 시스템 구성, 전체적인 제어 구조 및 통신, 동기화, 데이터 액세스를 위한 프로토콜, 설계 요소에 대한 기능 할당, 물리적 분포, 크기와 성능, 발전 정도, 그리고 설계 방안 사이에서의 선택을 포함한다. 이것은 소프트웨어 구조 단계이다[1].

소프트웨어 구조는 만들어진 시스템으로부터의 요소 기술 및 그들 요소들 사이의 상호작용과 그들의 합성을 이끄는 패턴 그리고 이들 패턴들에서의 제약을 포함한다. 일반적으로, 특정 시스템은 컴포넌트의 모음과 그들 컴포넌트 사이의 상호작용에 의하여 정의된다. 그러한 하나의 시스템은 커다란 시스템에서의 하나의 요소로 사용될 것이다.

소프트웨어 구조는 시스템 컴포넌트들과 그들의 상호관계를 이해하기 위한 프레임워크로서 제공된다. 이러한 이해는 기존 시스템의 분석을 위해서와 미래 시스템의 통합을 위해 필요하다. 분석의 지원에서, 구조는 분야 지식과 커뮤니티 일치를 포함하고, 컴포넌트 설계와 실행의 평가가 용이해야 하고, 시뮬레이션과 프로토타이핑이 쉬워야 한다. 통합의 지원에서, 구조는 생산 라인 확립과 예측 방법에서 모듈, 서브 시스템 및 시스템의 구성과 유지를 위한 분야 지식 사용을 위한 기초를 제공한다.

3. 관련연구

현재까지 소프트웨어 구조를 기술하는 많은 언어들이 제안되었다. 이것들은 범용 혹은 특정 영역에 적합하게 적용되도록 설계되어 있으며, 사용 영역에 따라 다양한 분석환경을 제공한다. 그 중에서 대표적인 소프트웨어 구조 기술 언어를 살펴보도록 한다.

- ArTek(ARDEC/Teknowledge)[2] developed by Hayes-Roth, et al, of Teknowledge for the DSSA program. ArTek은 지능적인 분산처리의 제어와 관리를 위한 소프트웨어 구조 기술 언어이며, ArTek을 위한 도구환경으로 DADSE(DSSA Application Development Support Environment)가 있다.
- GenVoca LE Language[3] developed by Don Batory, et al, of the University of Texas at Austin and in use on the Army STARS Demo Project. GenVoca는 Genesis와 Avoca에 기반을 두고 있으며 재사용 가능한 컴포넌트들의 구성으로 확장 가능한 계층적 시스템을 정의하는 도메인 독립 모델(Domain-Independent Model)이다. GenVoca는 큰 규모의 시스템을 구성하는 메타모델(Meta-Medel)의 특성을 지니며, 계층적 구조를 가지는 소프트웨어를 구성함에 있어 표준화된 매개변수 전달, 호환성, 컴포넌트들로 구성된 계층(layer)들의 재사용성을 제공한다. 또한 표준화된 인터페이스를 지원하여 컴포넌트 간의 구성 및 소프트웨어 시스템 확장성을 가능 하도록 한다.
- LILEANNA(Library Interconnect Language Extended with Annotated Ada)[4] developed by Loral team and Don Batory for the DSSA program. LILEANNA는 항공전자공학 부문에 적용되고 있으며, Ada를 위한 모듈 제작 언어인 LIL과, 폼과 Ada 코드의 구성에 대해 자동화된 분석을 제공하는 언어인 ANNA를 이용한다.
- MetaH[5] developed by Steve Vestal, et al, of Honeywell for the DSSA program. MetaH는 분석, 검사, 실시간 생성, 오류방지, 보안, 다중처리, 임베디드 소프트웨어 제작에 적합한 ADL이다.
- ControlH[6] developed by Steve Vestal, et al, of Honeywell for the DSSA program. ControlH는 간결하고도 명확한 방법으로 유도 장치나 항해 또는 제어 장치(GN&C: Guidance, navigation and Control) 알고리즘을 이용하는 ADL이다.
- Rapide[7] developed by the Stanford team for the DARPA Prototech project. Rapide는 분산 시스템에서의 프로토타이핑을 위한 객체지향 언어이며, 이벤트 처리 방식의 컴포넌트 구성에 있어 시스템 구조의 명세와, 분석 그리고 검사 기능을 제공한다.
- UNAS(Universal Network Architecture Services)[8] developed by TRW and Rational. UNAS는 상업적인 제품으로서 분산 시스템 및 이질적인 소프트웨어 시스템을 구성하기 위해 이미 정의되고, 재사용 가능한 Ada 모듈 및 서비스를 이용한다. UNAS는 분산 시스템을 위한 핵심적인 실행 함수들을 제공하기 위해 개발되었다. 예를 들자면, 초기화, 시스템 모듈 제어, 재구성, 오류 방지, 상태 감시, 프로세스간 통신 등이 있다. UNAS는 미 공군 STARS 데모 프로젝트(Demo Project)에 사용되었고, Rational사에 의해 Ada, C++에 적용 가능한 상업적인 제품으로도 활용되고 있다.
- UniCon(language for Universal Connector support)[9] developed by Shaw, et al, of Carnegie-Mellon University. UniCon은 소프트웨어 아키텍처의 구조적인 측면을 강조한다. 또한, 컴포넌트와 커넥터 상호 보완적인 구성에 기초를 둔다.
- Aesop[10] developed by the ABLE project at Carnegie-Mellon University. Aesop은 아키텍처 분석을 위해 일반적인 도구모음과 통신구조를 제공하여 소프트웨어 구조 설계 환경을 생성한다.
- Armani[11] developed by Robert Monroe of

Carnegie-Mellon University. Armani는 소프트웨어 구조 설계 기술과 명세를 포함하는 선언적 언어로 시스템 구조의 묘사와 추상화 구조 설계 기술을 가지기 위해 단일 통합 언어를 사용한다.

· Acme[12] developed by David Garlan, Bob Monroe and Drew Kompanek of Carnegie-Mellon University and David Wile of USC/Information Sciences Institute. Acme는 구조 설계 툴을 위한 공통된 교환 형식과, 새로운 구조 설계 개발과 분석 툴을 위한 기초이다.

4. HappyWork

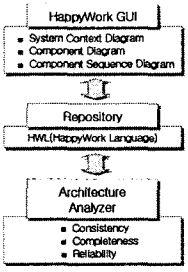
4.1 HappyWork 구조

우리가 개발한 HappyWork는 소프트웨어 구조 설계 환경으로서, 소프트웨어 구조를 설계하고 모델링 하기 위한 기능과 환경을 제공한다. 또한 소프트웨어 구조 모델로부터 자동적으로 HappyWork Language를 생성할 수 있다.

HappyWork는 두 가지 주요한 기능을 가지고 있다. 첫째, HappyWork는 소프트웨어 구조 다이어그램의 모델링을 위한 그래픽 에디터를 제공한다. 이것은 소프트웨어 구조를 기술하는 세 가지 다이어그램 즉, System Context Diagram, Component Diagram, Component Sequence Diagram을 이용하여 소프트웨어의 동적 및 정적인 면을 표현할 수 있다. 또한 각각의 다이어그램은 User, System, Component, Connector의 소프트웨어 구조적 특성에 대한 네 가지 요소로 표현될 수 있다.

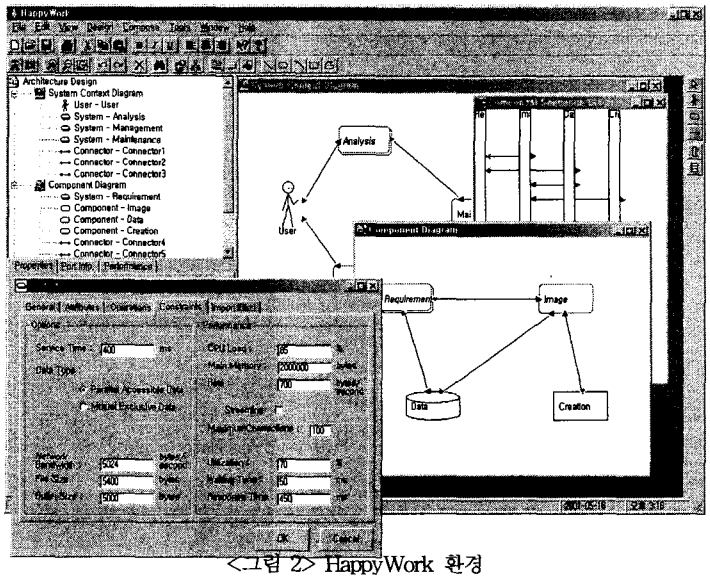
둘째, HappyWork는 HWL(HappyWork language)라 불리는 소프트웨어 구조 기술 언어(ADL)를 제공한다. HWL는 HappyWork의 코드 생성기로부터 생성된다. HWL는 소프트웨어의 구조적인 면과 기능적인 면을 함께 나타냄으로써, 사용자가 HWL 파싱을 통해 소프트웨어를 쉽게 분석하고 이해할 수 있도록 한다.

HappyWork는 <그림 1>에서 보는 바와 같이 HappyWork GUI(Graphic User Interface), Repository, Architecture Analyzer의 세 가지 모듈로 구성된다.



<그림 1> HappyWork 구조

소프트웨어 설계자는 HappyWork GUI에서 제공하는 System Context Diagram, Component Diagram, Component Sequence Diagram의 세 다이어그램을 사용하여 소프트웨어를 표현할 수 있다. 이들 다이어그램들은 소



<그림 2> HappyWork 환경

프트웨어 구조의 특성과 소프트웨어의 동적이고 정적인 면을 함께 포함한다.

<그림 2>는 HappyWork GUI를 나타낸다. 소프트웨어 구조 다이어그램은 HWL로 바뀔 수 있고, HWL는 Repository를 통해 저장된다. 또한 HWL는 HappyWork GUI를 통해 다시 다이어그램으로 표현될 수 있다. Architecture Analyzer는 HWL 파싱을 통해 소프트웨어 구조의 일관성, 완전성, 신뢰성을 분석 할 수 있다.

4.2 소프트웨어 구조 모델링 표기법

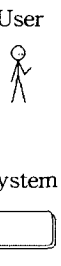
소프트웨어 구조를 기술하기 위하여 소프트웨어 설계자들은 세 가지 다이어그램 즉, System Context Diagram, Component Diagram, Component Sequence Diagram을 개발한다.

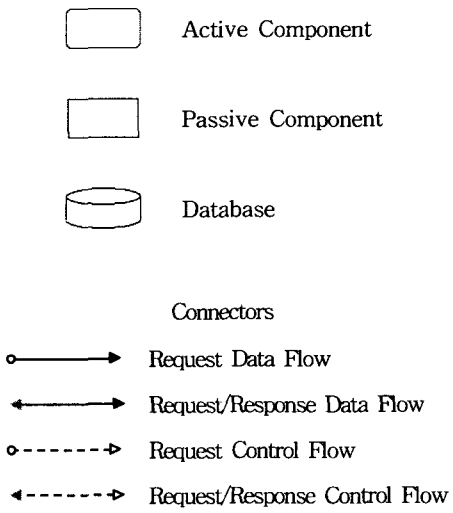
4.2.1 그래픽 표기법

HappyWork는 다음의 세 가지 다이어그램을 사용한다.

- System Context Diagram
- Component Diagram
- Component Sequence Diagram

각각의 다이어그램은 소프트웨어 구조의 서로 다른 면을 표현한다. 비록 모든 다이어그램은 서로 관계를 가지지만 그 역할은 엄격히 분리되어 있다. 다이어그램들은 다양한 요소로 구성되어 있다. <그림 3>은 요소들의 그래픽 표기법을 보여준다.





<그림 3> 아키텍처 구성 요소

User는 시스템과 상호작용하는 누군가이거나 다른 시스템이다. User는 시스템과 메시지를 주고받거나 정보를 교환한다.

System은 특정 기능을 수행하기 위한 요소들의 집합이고, 이것은 또한 서브시스템으로 나뉘어 질 수 있다.

컴포넌트는 Active Component, Passive Component, Database로 구성된다. Active Component는 스스로 요구하거나 일할 수 있다. Passive Component는 파일처럼 스스로 일할 수 없는 것을 의미한다. Database는 다른 종류의 Passive Component 이다. 이것은 정적 데이터의 저장소를 표현한다.

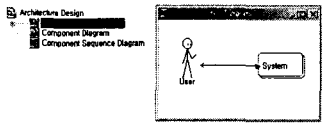
하나의 시스템 또는 컴포넌트의 특성은 Attributes, Operations, Constraints와 같은 속성으로 표현된다.

커넥터의 종류는 Request Data Flow, Request/Response Data Flow, Request Control Flow, Request/Response Control Flow 이다. Request Data Flow는 단방향 데이터 전송을 표현하고, Request/Response Data Flow는 양방향 데이터 전송을 말한다. Request Control Flow는 단방향 제어 흐름을 Request/Response Control Flow는 양방향 제어 흐름을 나타낸다. 커넥터의 특성은 Types과 Constraints 같은 속성으로 나타낸다.

4.2.2 System Context Diagram

<그림 4>와 같이 System Context Diagram은 소프트

웨어 구조의 가장 높은 단계를 표현한 것으로, 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 Diagram을 통해 구체화될 수 있다.



<그림 4> System Context Diagram

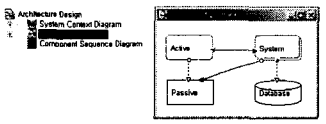
이 다이어그램은 다음과 같은 구성 요소를 포함한다.

- User
- System
- Connector

4.2.3 Component Diagram

<그림 5>와 같이 Component Diagram은 System Context Diagram에서 나타난 System의 구조를 자세히 나타낸다. 세부 구조는 컴포넌트와 컴포넌트 사이의 상호작용을 통해 표현된다. 하나의 시스템은 몇 개의 컴포넌트 또는 시스템으로 나뉘어질 수 있고, 이러한 컴포넌트는 Active 또는 Passive 이다.

하나의 컴포넌트는 관련된 기능들의 집합을 나타내며, 컴포넌트 사이의 커넥터는 통신을 위한 메시지를 표현한다.



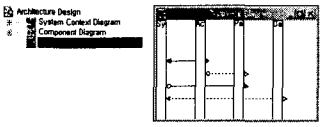
<그림 5> Component Diagram

이 다이어그램은 다음과 같은 구성요소를 포함한다.

- System
- Component
- Connector

4.2.4 Component Sequence Diagram

<그림 6>와 같이 Component Sequence Diagram은 Component의 상호작용을 순차적으로 보여준다. 이 다이어그램은 시간의 흐름에 따라 메시지들이 Component간에 어떻게 전달되는지를 나타낸다.



<그림 6> Component Sequence Diagram

이 다이어그램은 다음과 같은 구성요소를 포함한다.

- System
- Component
- Connector

4.3 소프트웨어 구조 모델링 방법

HappyWork를 이용한 아키텍처 모델링 방법은 6단계로 나누어진다. 6단계는 요구분석, 시스템 설계, 컴포넌트 설계, 기능 분석 및 성능 평가, 개발, 유지보수와 재사용을 포함한다. 하지만 모든 시스템이 6단계로 구분한 개발순서에 일치하지 않을 수도 있다. 시스템의 특성이나 복잡도에 의해 더욱 구체화될 필요성이 있거나, 중요하지 않은 단계도 있을 수 있다.

4.3.1 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 것으로 세부 단계는 다음과 같다.

- 단계1: 사용자의 요구 및 제약사항을 결정한다.
- 단계2: 서비스의 형태 및 방법을 구상한다.
- 단계3: 필수 요구사항과 선택 요구사항을 구분하고, 요구의 순위를 결정한다.
- 단계4: 서비스의 순서 등 기본적인 문제들을 명확히 한다.

4.3.2 System 설계

시스템 설계는 여러 단계에 걸쳐 이루어진다. 각 단계는 다른 설계 관심을 다루기 때문에, 시스템 설계는 이 단계들 사이의 명확한 차이를 만들기 위해 유용하다. 각 단계마다 우리는 원형 및 합성의 컴포넌트와 비원형의 컴포넌트 또는 시스템의 합성 규칙, 그리고 시스템에 의미를 부여하는 행동 규칙들을 발견할 수 있다. 또한 각 단계마다 다른 표기법과 설계 문제, 분석 기법을 볼 수 있다. 결과적으로, 각 단계에서의 설계는 실제로 자율적으로 이루어진다.

설계 문제가 컴포넌트와 관련하여 전체적인 시스템 성능을 수반할 때, 컴포넌트는 모듈들이고, 모듈들 사이의 상호작용은 다양한 방법으로 처리된다.

시스템 설계 단계에서 분석의 결과는 기술적인 솔루션을 통해 설명된다.

시스템의 구조, 기능, 제약 그리고 상호작용은 이 단계에서 고려되며, 세부 단계는 다음과 같다.

- 단계1: System의 형태, 기능, 비용, 특성을 결정한다.
- 단계2: System의 수행 능력, 문제 발생 요인을 예측한다.
- 단계3: Hardware의 성능, 데이터 저장 공간, 외부와의 연결 등 Hardware 조건을 고려한다.
- 단계4: 기존의 System과의 관계를 표현한다.
- 단계5: System 최적화를 위한 고려사항을 점검한다.

4.3.3 Component 설계

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

- 단계1: 요구분석 단계에서의 기능들을 Component의 Operations에 추가한다.
- 단계2: Component간의 상호관계를 정의한다.
- 단계3: Component의 Properties를 결정한다.
- 단계4: 제약사항을 명시하고 일관성을 확인한다.

4.3.4 기능분석 및 성능평가

소프트웨어 기능을 분석하고 일관성, 완전성, 신뢰도 및 수행능력을 평가한다. 세부 단계는 다음과 같다.

- 단계1: 일련의 기능들에 대해 테스트한다.
- 단계2: 오류에 대해 어떻게 처리하는지 살핀다.
- 단계3: 소프트웨어 전체 효율과 지연시간은 얼마인지 확인한다.
- 단계4: 문제점을 제거하고 Design을 최적화 한다.

4.3.5 개발

지금까지 단계의 분석 및 설계를 바탕으로 소프트웨어를 구현한다. 세부 단계는 다음과 같다.

- 단계1: 각 단계에서의 구성을 바탕으로 소프트웨어 구현에 적용한다.
- 단계2: 제품을 위한 Package를 작성한다.

4.3.6 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. 세부 단계는 다음과 같다.

- 단계1: 결점 및 보완점을 찾는다.
- 단계2: 새로운 시스템 구축 시 재구성하거나 참조한다.

4.4 구조 분석

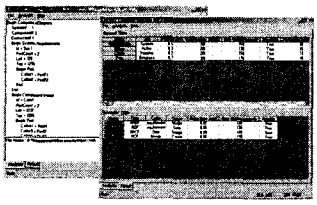
소프트웨어 구조 차원에서의 시스템 평가는 조금은 다른 시각에서 이루어진다. 여기서 일관성, 완전성 및 성능이라 하면 시스템을 구성하는 요소들간의 연관성 및 효율성 측면에서 평가된다. HappyWork에서는 다음과 같은 분석 기능을 가지는 분석기를 제공한다.

- 일관성(Consistency) : 컴포넌트의 구성과 커넥터의 프로토콜간의 문제가 서로 잘 대응되는지를 분석한다.
- 완전성(Completeness) : 요소들의 기술에서 누락된 것이 없는지를 살핀다. 즉 심각한 오류는 없는지, 또는 오류가 나타날 경우에 적절히 대처하고 있는지 평가한다.
- 성능(Performance) : 각 연산이나 시스템 전반의 성능을 평가한다.

4.4.1 HappyWork Analyzer

HappyWork Analyzer는 소프트웨어 구조 성능 분석 도구이다. 그것은 소프트웨어 구조 다이어그램에 대한 오류를 분석하고 시스템 전반에 대한 성능을 평가한다.

<그림 7>은 소프트웨어 구조의 성능 분석을 위한 HappyWork Analyzer 환경이다.



<그림 7> HappyWork Analyzer 환경

HappyWork Analyzer는 소프트웨어 구조의 일관성과 완전성을 체크하기 위해 HappyWork 구조 기술 언어 문법에 따라 HWL를 파싱한다. 이때 에러가 발견되면 에러 메시지를 출력한다.

4.4.2 성능 분석

HappyWork Analyzer는 소프트웨어 구조 성능 분석의 결과로 다음과 같은 두개의 테이블을 제공한다.

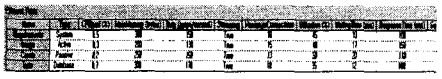
- Element Table
- Connector Table

각 테이블은 소프트웨어 구조 모델링에서 시스템, 컴포넌트, 커넥터의 일반적인 성능을 나타낸다. 그리고 그 테이블에 기반하여 소프트웨어 전체의 효율과 지연시간을 구함으로써 신뢰성 분석을 용이하게 한다.

a. Element Table

Element Table은 시스템과 컴포넌트에 대한 constraints를 나타낸다. 이 Table의 필드는 다음과 같이 구성되고, <그림 8>는 Element Table의 한 예를 나타낸다.

- Name: 시스템 또는 컴포넌트의 이름
- Type: System | Active | Passive | Database
  System - system의 type
  Active - request할 수 있는 능동적인 객체
  Passive - File과 같이 스스로 연산이나 처리와 같은 작업을 수행할 수 없는 수동적인 객체
  Database - Active와 Passive 두 가지 특성을 모두 가짐
- CPUload: CPU 사용량(단위:%)
- MainMemory: Memory 사용량(단위:bytes)
- Rate: 입출력의 빈도(단위:bytes/second)
- Streaming: stream data가 계속적으로 전송되는지 여부 (단위:True | False)
- MaximumConnections: 연결의 최대 허용 개수, Port 마다 연결된 Connector의 합
- Utilization: 이용률, 단위시간당 사용시간 (단위:%)
- WaitingTime: 서비스를 받기 위한 평균 대기 시간 (단위:ms)
  Response Time = Waiting Time + Service Time
- ResponseTime: 서비스 요구에 대한 Server 측의 응답 시간 (단위:ms)
  네트워크상의 오고 가는 지연시간은 제외
- ServiceTime: CPU상에서 수행되는 시간 (단위:ms)
- DataAccessType: 공유 데이터에 접근하는 방식 선택
  Parallel Accessible Data - 동시접근 가능
  Mutual Exclusive Data - 데이터 점유
- NetworkBandwidth: 네트워크 대역폭 (단위:bytes/second)
- FileSize: Element의 물리적인 실제 크기 (단위:bytes)
- BufferSize: Port들의 Queue 행렬을 위한 버퍼 크기 (단위:bytes)



<그림 8> Element Table

b. Connector Table

Connector Table은 커넥터에 대한 constraints를 나타낸다. 이 Table의 필드는 다음과 같이 구성되고, <그림 9>는 Connector Table의 한 예를 나타낸다.

- Name: 커넥터의 이름
- Type
  Request Data Flow - 데이터 전송(data send)
  Request/Response Data Flow - 데이터 전송, 데이터 수신(data send, data receive)
  Request Control Flow - 신호 전송(signal send)

Request/Response Control Flow - 신호 전송, 데이터 수신(signal send, data receive)

- Caller: 서비스를 요구하는 Component 또는 System의 Port 지정; Connector의 시작점
- Callee: 서비스를 요구 받는 Component 또는 System의 Port 지정; Connector의 종점
- DelayTime: 네트워크 상의 물리적인 지연시간
- Rate: 초당 전송되는 데이터량 (단위:bytes/second)
- Reliable: 신뢰도(단위:True | False)

<그림 9> Connector Table

5. 비 교

이 장에서는, 모델링 방법과 분석 기능에 기반 하여, 우리가 개발한 Happy Work와 다른 ADL들을 비교한다. <표 1>은 ADL들의 특징을 보여준다.

<표 1> 구조 기술 언어의 비교

ADLs	GUI	Analysis Tool	특 징
HappyWork	○	○	- 3가지 다이어그램의 지원 · System Context Diagram · Component Diagram · Component Sequence Diagram - HWL와 통합된 환경 - HappyWork analyzer 제공 (constraint를 통한 분석) · Consistency · Completeness · Reliability
UniCon	○	×	- 다양한 type의 지원 - 외부 툴을 이용한 분석 지원
Aesop	○	×	- Style-Specific 소프트웨어 구조 설계 환경 - 외부 툴을 이용한 분석을 지원
Wright	×	○	- Architectural Configuration과 Styles를 상세하게 기술 - 분석 · consistency · application · equivalence
ControlH	○	○	- ControlH는 Ada 또는 C 코드를 생성 - Simulation을 통해 분석
MetaH	○	○	- 항공전자공학 분야에 적절 - 분석 · schedulability · reliability · secure partitioning
Rapide	○	○	- Raparch Tool 제공 - Raptor라는 Simulation Animator를 이용한 분석
Acme	○	×	- AcmeStudio Tool 제공 - 다른 Languages로의 매핑을 통해 분석
Armani	×	×	- 소프트웨어 아키텍처 designs, constraints를 위한 풍부한 언어를 제공

6. 결 론

재사용의 가능성은 소프트웨어 구조 단계에서의 명세를 최소화 할 때 가장 크다. 소프트웨어 구조는 만들어진 시스템으로부터의 요소 기술 및 그들 요소들 사이의 상호작용과 그들의 합성을 이끄는 패턴 그리고 이들 패턴들에서의 제약을 포함한다. 일반적으로, 특정 시스템은 컴포넌트의 모음과 그들 컴포넌트 사이의 상호작용에 의하여 정의된다.

우리가 개발한 HappyWork는 소프트웨어 구조 설계 환경으로서, 소프트웨어 구조를 설계하고 모델링 하기 위한 기능과 환경을 제공한다. HappyWork는 HappyWork GUI(Graphic User Interface), Repository, Architecture Analyzer의 세 가지 모듈로 구성된다. 소프트웨어 구조를 기술하기 위하여 소프트웨어 설계자들은 세 가지 다이어그램 즉, System Context Diagram, Component Diagram, Component Sequence Diagram을 개발한다.

소프트웨어 구조 다이어그램은 HWL로 바뀔 수 있고, HWL는 Repository를 통해 저장된다. 또한 HWL는 HappyWork GUI를 통해 다시 다이어그램으로 표현될 수 있다. Architecture Analyzer는 HWL 파싱을 통해 소프트웨어 구조의 일관성, 완전성, 신뢰성을 분석 할 수 있다.

참 고 문 헌

[1] Mary Shaw, and David Garlan, "Software Architecture: Perspectives on an Emerging Discipline," Prentice-Hall, Inc., 1996.

[2] A. Terry, R. London, G. Papanagopoulos, and M. Devito, "The ARDEC/Teknowledge Architecture Description Language (ArTek), Version 4.0," Technical Report, Teknowledge Federal Systems, Inc. and U.S. Army Armament Research, Development, and Engineering Center, July 1995.

[3] Don Batory, Lou Coglianese, Mark Goodwin, and Steve Shafer, "Creating Reference Architectures: An Example From Avionics," IBM technical report, 1993.

[4] Will Tracz, "Parametrized programming in LILEANNA," Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing, pp. 77-86, February 1993.

[5] Agrawal, A., Krause, J. and Vestal, S., "Domain Specific Software Architectures for Intelligent Guidance, Navigation and Control," IEEE Symposium on Computer Aided Control System Design (CACSD), pp. 110-116, March 1992.

[6] Englehart, M., and Jackson, M. "ControlH: A Fourth Generation Language for Real-time GN&C Application," IEEE/IFAC Joint Symposium on Computer-Aided Control System Design, pp. 261-270, March 1994.

[7] David C. Luckham, John J. Kenney, Larry M. Augustin, James Vera, Doug Bryan, and Walter Mann, "Specification and analysis of system architecture using Rapide," IEEE Transactions on Software Engineering, 21(4), pp. 336-355, April 1995.

[8] Royce, Walker, and Winston Royce, "Software Architecture: Integrating Process and Technology," TRW-TS-91-04, December 1991.

[9] Mary Shaw, Robert DeLine, Daniel Klein, Theodore Ross, David Young and Gregory Zelesnik, "Abstractions for software architecture and tools to support them," IEEE Transactions on Software Engineering, 21(4), pp. 314-335, April 1995.

[10] David Garlan, Robert Allen and John Ockerbloom, "Exploiting Style in Architectural Design Environments," Proceedings of the second ACM SIGSOFT symposium on Foundations of software engineering, pp. 175-188, December 1994.

[11] Robert T. Monroe, "Capturing Software Architecture Design Expertise with Armani," Technical Report CMU-CS-98-163, October 1998.

[12] David Garlan, Robert T. Monroe and David Wile, "Foundations of Component-Based Systems," Cambridge University Press, pp. 47-68, 2000.

강 병 도 (Byeong-do Kang)

1986년 2월 서울대학교 계산통계학과 졸업(이학사)

1988년 2월 서울대학교 대학원 전산과학과 졸업(이학석사)

1995년 2월 서울대학교 대학원 전산과학과 졸업(이학박사)

1988년 6월~1998년 2월 한국전자통신연구원 선임연구원

1998년 3월~현재 대구대학교 교수로 재직중

관심분야 : 소프트웨어 개발방법론, 소프트웨어 구조, 소프트웨어 프로세스