소프트웨어의 재사용 및 유지보수를 위한 컴포넌트 기반 소프트웨어 구조 설계

Component based Software Architecture Design for Software Reuse and Maintenance

온용훈, 강병도
대구대학교 컴퓨터정보공학부

Yong-Hoon On, Byeong-Do Kang
Department of Computer Information Engineering
Taegu University

요 약

근래에 들어와서 소프트웨어 개발 방법론에 있어 소프트웨어의 재사용 및 유지보수를 위한 여러 방안이 제시되었다. 이를 위해 분석 및 설계에 초점을 두어 여러 객체 지향적인 개발 방안이 나왔는데, 일반적으로 컴포넌트를 기반으로 모형화하고 있으며, 이에 대한 연구가 현재 활발히 진행되고 있다.

본 논문에서는 소프트웨어 개발 과정에서 컴포넌트를 기반으로 소프트웨어 구조를 설계하고, 모형화하기 위한 도구로 HappyWork을 구현하였으며, 이를 기술하기 위한 언어로 HWL(HappyWork Language)을 만들었다. 우리는 이를 이용해 소프트웨어를 구조적인 차원에서 심도있게 분석하고 성능을 평가해 보고자 한다.

1. 서 론

소프트웨어 개발의 편의성과 재사용성, 그리고 개발기간의 단축을 위해 최근의 시스템 설계는 컴포넌트 기반으로 이루어지고 있다. 따라서 이것을 명세하기 위한 하나의 공통적인 기술언어가 요구되며 이것을 위한 여러 객체지향 개발 방법론이 제시되고 있다[1]. 현재 객체지향 시스템 개발 기법으로 과거 OMT(Object Modeling Technic), OOSE(Object-Oriented Software Engineering) 등을 새로이 집대성한 이론으로 UML(Unified Modeling Language)이 있으며, 현재의 CORBA(Common Object Request Broker Architecture), JAVA를 이용한 시스템 설계시에 이 이론은 필수적으로 이용되고 있다[2]. 또한, UML 표기 이론에 따라 설계를 하면 자동으로 JAVA나 C++ 코드로 생성하는 도구도 나와 있다[3].

하지만 우리는 단지 시스템을 모델링하고 실행가능한 코드를 생성하는 것뿐만 아니라 시스템을 분석하고 평가할 필요성이 있다. 또한 재사용 및 유지보수를 위한 더욱 세심한 배려가 요구된다. 이러한 요구에 부합하여 소프트웨어 구조 기술 언어들(Software Architecture Description Languages)이 심도있게 연구되고 있으며[4][5][6][7][8][9], 본 논문은 소프트웨어 구조에 대해 새로운 관점에서 분석함으로서 그 특성을 파악하고 성능을 평가하며 더욱 양질의 소프트웨어 개발을 위한 방향으로 우리가 개발한 HappyWork를 제시한다.

2. HappyWork 소개

HappyWork는 시스템의 구조에 대한 프로토타이핑을 지원하고, 이를 명세화한 것이 HWL(HappyWork Language)이다. 소프트웨어의 구조적인 측면을 강조하며, 컴포넌트간의 메시지 이동과 이를 통제하기 위한 프로토콜, 그리고 제약사항을 명시한다. 이를 구체화한

것이 커넥터이며, 시스템 전체에 대한 제어구조를 나타내고, 컴포넌트간의

① 상호작용 문제
② 통신 프로토콜
③ 동기화 문제
④ 자료 접근에 대한 제약 및 방법의 문제
⑤ 오류 인식 및 복구의 기능
⑥ 성능 개선의 문제

등을 표현한다.

메시지 전달에 대하여 구체적인 컴포넌트의 수행은 은닉하고, 응답으로 신호나 정형화된 데이터를 전달함으로서, 전체적인 시스템의 인터페이스에 대한 분석을 할 수 있다. 따라서 HappyWork의 주된 기능은 분산시스템을 설계시 적용될 문제, 즉 위에서 나열한 6가지 문제에 대해 컴포넌트와 커넥터를 사용하여 시스템 전체에 대한 인터페이스를 구성하고, 이러한 구성을 명세하는 HWL(HappyWork Language)를 이용해 기술한다.

2.1 HappyWork의 특징

HappyWork는 소프트웨어 구조를 시각화하고, 자동으로 HappyWork Language로 명세화하는 모델링 환경이다. HappyWork와 HappyWork Language의 두드러진 특징들을 살펴보면 다음과 같다.

① 다양한 자료형을 제공한다. C 언어를 기반으로 하며, 시스템 및 컴포넌트도 자료형이 될 수 있다.

② 컴포넌트간의 메시지 전송을 지원한다. 메시지 전달 방식에 따라 다양한 형태의 제약사항이 있으며, 이것을 통해 컴포넌트간의 상호작용이 가능하다.

③ 소프트웨어를 구조적인 측면에서 명세하는 HappyWork 언어를 생성한다.

④ C 언어의 서브루틴 또는 DLL(Dynamic Linked Library)과 같은 리소스를 호출할 수 있다.

⑤ 엄격한 분석도구를 제공하여 시스템 설계시 신뢰성 있는 구성을 할 수 있다.

⑥ 생성된 코드는 재사용 가능하다.

⑦ HappyWork는 사용하기 쉽도록 제작된 도구이다.

⑧ HWL은 소프트웨어 구조를 효과적으로 명세할 수 있도록 제작된 간결한 언어이다.

2.2 HappyWork에서 사용하는 다이어그램

HappyWork의 다이어그램은 사용자, 컴포넌트, 시스템, 커넥터 등과 같은 소프트웨어 구조의 표현 양식을 그래프 형태로 나타낸 것이다. 사용자는 다양한 관점에서 소프트웨어 구조를 바라볼 수 있으며, HappyWork에서는 소프트웨어 구조를 구성하기 위해 다음과 같은 세 가지 다이어그램을 정의한다.

System Context Diagram

System Context Diagram은 사용자, 설계자 및 개발자의 관점에서 바라본 소프트웨어 구조의 표현 양식을 그래프 형태로 나타낸 것이다.

Component Diagram

System Context Diagram에서 구성한 User와 System간의 관계를 더욱 구체화한 것으로, 이들 컴포넌트간의 상호작용을 표현한다. 이 다이어그램은 상호 작용하는 컴포넌트들을 포함하고 있고, 이들간에 전달되는 메시지들을 통해 그 연관성을 분석할 수 있다. 따라서 컴포넌트 다이어그램은 시스템의 정적인 부분을 표현하면서도 내부적으로 동적인 부분을 포함하기도 한다.

Component Sequence Diagram

Component Sequence Diagram은 컴포넌트 간의 상호작용을 순차적으로 보여준다. 이 다이어그램은 시간의 흐름에 따라 메시지들이 컴포넌트간에 어떻게 전달되는지를 나타낸다. 이것은 시스템 내의 통신을 시각화하는데 사용될 수 있으며, 이해하기가 쉽게 표현되고 있다.

2.3 소프트웨어 구조를 구성하는 요소

소프트웨어 구조를 구성하기 위해 세 가지 다이어그램이 사용되는데, 각각의 다이어그램에 User, System, Component, Connector등과 같은 구성요소가 사용된다. 시스템의 구체적인 설계를 위한 일반 모델링 도구와는 달리 이것은 전체적인 소프트웨어의 구조를 명세하기 위한 최소의 구성요소이며, 분석과 성능평가에 대한 기능을 위해 설계되었다.

User

User는 System과 상호 작용하는 어떤 사람이거나 또 다른 시스템이다. 시스템에게 User는 메시지를 보내고 그 결과를 받는 요구와 응답의 관계이다. 소프트웨어 구조를 설계할 때, 사용자와 개발자간의 요구분석이 이루어지고, 이를 토대로 User와 시스템간의, 또는 시스템과 기존의 시스템간의 구성을 설계할 수 있다. 여기서 User는 시스템에게 커넥터를 통하여 요구를 할 수 있고, 시스템은 이를 통해 응답을 해 줌으로써, 전체적인 소프트웨어 구조에 대한 윤곽을 표현할 수 있다.

System

HappyWork는 소프트웨어 구조를 계층화된 구조로 표현하며, 시스템은 서브시스템 또는 컴포넌트 및 커넥터로 구성된 하위 계층을 포함할 수 있다. 시스템은 특수한 기능을 수행하기 위해 구성된 일련의 요소들의 집합이라 할 수 있으며, 시스템은 서브시스템의 집합체로 분해될 수 있다. 시스템의 경계를 정의하는 것은 시스템의 내부 요소와 외부 요소를 식별하고 시스템의 기능을 정의하는 것이므로 매우 중요하다.

Component

컴포넌트는 현재 크게 관심을 갖는 대상이며, 컴포넌트라는 용어는 재사용 가능하고 추상화된 소프트웨어 제작물로 정의될 수 있다. 즉 컴포넌트는 클래스에 의해 정의될 수 있는 하나의 인스턴스이며, 일련의 인터페이스에 대한 실현을 제공한다. HappyWork에서 컴포넌트는 가장 기본적인 설계단위이며, 시스템에서 작업을 수행하기 위한 하나의 요소이다. 컴포넌트는 독립된 소프트웨어이거나 하드웨어 제어를 위한 컨트롤러일 수도 있으며, 다른 컴포넌트와 상호연결 가능하다면 어떤 구성이든지 간에 하나의 컴포넌트로 표현 가능하다.

HappyWork에서 컴포넌트는 그 유형에 따라 Active, Passive, Database로 나타나며, 다이어그램에서 서로 다른 박스형태로 표현된다. 여기서 Active 속성은 컴포넌트가 Request를 할 수 있는 능동적인 객체임을 뜻하고, Passive속성은 파일과 같이 스스로 연산이나 제어와 같은 작업을 수행할 수 없는 수동적인 객체를 나타낸다. Database의 경우 Active와 Passive의 두 가지 특성을 모두 가지므로 따로 속성을 지정하였다.

전형적인 컴포넌트의 예로 클라이언트, 서버, 필터, 파일시스템 등을 들 수 있는데 이들 간의 인터페이스는 Port에 의해 정의된다. 각각의 Port는 컴포넌트와 다른 컴포넌트간의 상호작용을 위한 연결점을 제공한다. 또한 컴포넌트는 다른 타입의 Port를 사용하여 다중 인터페이스를 제공할 수 있다. Port는 단순하게 하나의 Procedure Signature에 대해 나타낼 수도 있지만, 여러 Procedure Call에 대해 처리하거나, Multi-Cast Event와 같은 메시지 처리도 수행할 수 있어야 한다.

컴포넌트는 기본적인 설계의 단위이므로, 컴포넌트간의 메시지 이동이나 상호관계를 통해 그 기능 및 역할을 쉽게 분석할 수 있다. 또한 이러한 컴포넌트와 커넥터를 하나의 시스템으로 구성하여 그 정보를 은닉화할 수도 있다. 이렇게 그룹화 및 계층화된 구조를 통하여 HappyWork는 소프트웨어 구조를 설계한다.

Connector

User, System, Component간의 상호작용을 정의한다. 상호작용은 구체적인 목적을 수행하기 위해 요소들간에 교환되는 메시지로 구성되는 행위이며, 소프트웨어 구조의 동적인 측면이다. 이러한 상호작용은 메시지 전달에 대한 규약을 준수하며, 제약사항에 따라 요청과 응답을 반복한다. 대부분의 경우 메시지는 신호나 오퍼레이션의 호출이며, 이를 위해 데이터를 전달하기도 한다. 따라서 소프트웨어 설계자 및 개발자는 System Context Diagram, Component Diagram, Component Sequence Diagram 내의 제어 흐름을 모델링하기 위해 커넥터를 이용한다. System Context Diagram과 Component Diagram에서는 구성요소들간의 구조적인 관계를 정의하기 위해 사용되고, Component Sequence Diagram에서는 시간의 흐름에 따른 메시지 전달 경로를 정의하기 위해 사용된다.

전형적인 커넥터의 예로 Pipe, Procedure Call, Event, Broadcast 등을 들 수 있는데 인터페이스는 Role에 의해 정의된다. 각각의 Role은 커넥터에 의해 표현된 상호작용에 관여한다. 즉, 일반적인 RPC(Remote Procedure Call) 커넥터의 경우 Sender와 Receiver와 같은 두개의 Role을 가진다. 물론 다른 경우에 있어서 2개 이상의 Role을 가질 수 있다. 예를 들어 Event Broadcast 커넥터의 경우 하나의 Sender와 여러 개의 Receiver를 가지게 된다.

3. HWL(HappyWork Language)

HWL은 분산환경에서의 시스템을 명세하기 위한 언어이며, HappyWork의 코드 생성기를 통해 만들어진다. HappyWork는 System Context Diagram과 Component Diagram, 그리고 Component Sequence Diagram을 통해서 구성된 각각의 Element에 대해서 그 연관과 상호작용 및 제약사항에 대해서 본 논문에서 제시한 HappyWork Language를 생성한다.

[그림 1]은 HappyWork의 코드 생성기를 통해 만들어진 컴포넌트와 커넥터의 명세를 보이고 있다.

4. 시스템 설계와 소프트웨어 재사용

본 논문에서 구현한 HappyWork는 기존의 설계도구에 모델링 기능의 편의성과 분석기능을 유기적으로 결합시킨 것으로서 상용 도구들이 가지는 설계의 정밀성이나 인터페이스의 복잡성을 유연하게 일관화하여 소프트웨어 구성에 있어 그 연관성이나 재사용 가능성의 구현에 더 중점을 두었다.

[그림 2]는 HappyWork를 사용하여 소프트웨어를 설계하는 화면으로 세 가지 다이어그램에 대해서 여러 구성요소가 존재하고 분석과 성능평가를 위한 창을 제공하고 있다.

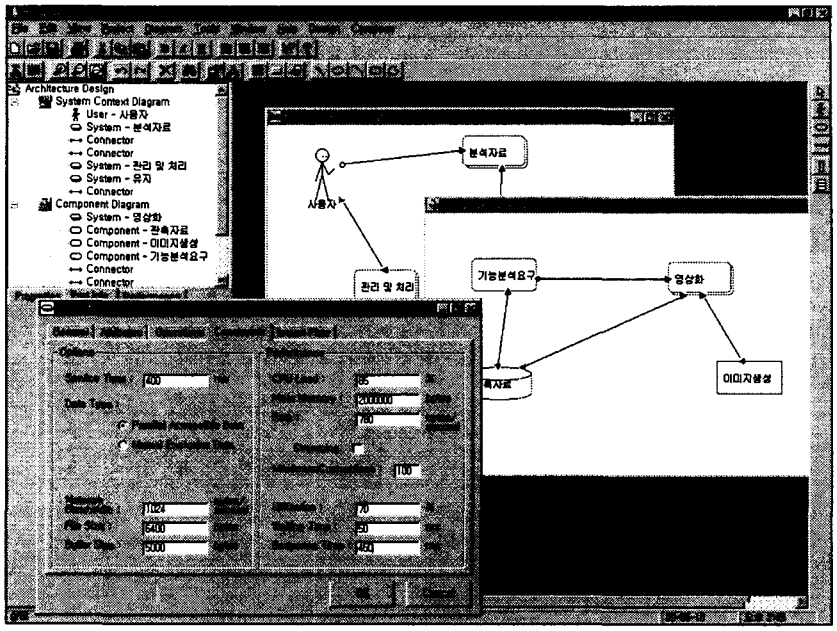
5. 결 론

컴포넌트를 기반으로 한 HappyWork Language와 HappyWork는 소프트웨어 구조에 대해 표현하거나 그 기능을 추출해 내는데 유용하였다. 또한 HWL과 HappyWork를 통하여 소프트웨어 구조 차원에서 심도있는 분석과 성능평가를 할 수 있었다. 이러한 시도는 소프트웨어의 빠른 개발에 중점을 두고있는 일반 RAD(Rapid Application Development) 도구들과 비교해 소프트웨어 재사용 및 관리를 위한 개발환경 및 명세언어를 제시했다는 점에 의의가 있다.

그러나 다양한 응용환경에 대한 일관성을 제공하지 못하고, 다른 소프트웨어 개발 도구와의 연계가 되지 않는 문제점 등이 향후 과제로 남아있다.

```
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
                Utilization = 90
                WaitingTime = 50
                ResponseTime = 160
                ServiceTime = 110
                DataType = Parallel Accessible Data
                NetworkBandwidth = 1024
                FileSize = 450
                BufferSize = 200
        }

        Import Files
        {
                "CommonLib.Dll"
                "Library.Dll"
                "Tokenize.Dll"
        }

        Description {...}
}

Connector FindAddress Type Request / Response Data Flow
{
        Sender s1 = RequestAnalysis.p1
        Receiver r1 = AddressDB.p5

        Constraints
        {
                DelayTime = 100
                Rate = 400
                Reliable = True
        }

        Description {...}
}
```

[그림 1] HappyWork Language의 예 (컴포넌트, 커넥터)



[그림 2] HappyWork를 이용한 소프트웨어 구조 설계

감사의 글

이 논문은 정보통신연구관리단 1998년도 대학기초연구지원사업의 지원 (C1-98-0771-0 0)에 의한 연구결과입니다.

참고문헌

[1] J. Rumbaugh, M.Blaha, W.Premerlani, F.Eddy, W.Lorensen, "Object-Oriented Modeling and Design," Prentice Hall, 1991

[2] 왕창종, 이세훈, "분산객체 컴퓨팅 기술 CORBA 프로그래밍," 대림 출판사

[3] Hans-Erik Eriksson and Magnus Penker, "UML Toolkit," Wiley Computer Publishing, 1998.

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

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

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

[7] Lockheed Martin Advanced Software Technology, http://www-ast.tds-gn.lmco.com/

[8] Carnegie Mellon University's ABLE Project, http://www.cs.cmu.edu/~able/

[9] The Software Technology for Adaptable, Reliable Systems (STARS), http://www.asset.com/stars/.