제 18 회 한국정보처리학회 추계학술발표대회 논문집 제 9 권 제 2 호 (2002.11)

컴포넌트 조립을 지원하는 워크플로우 기반의 아키텍처 모델

서효길*, 홍찬기**
*관동대학교 전자계산공학과
**관동대학교 컴퓨터공학과
e-mail : cbells@kwandong.ac.kr
chankih@kwandong.ac.kr

The Workflow based Architecture Model for Component Assembly

Hyo-Gil Seo*, Chan-Ki Hong**
*Dept. of Computer Science , Kwandong University
**Dept. of Computer Engineering, Kwandong University

요 약

컴포넌트 기반 소프트웨어 개발에 있어서 필수 요소라 할 수 있는 아키텍처의 중요성이 부각되면서 아키텍처 상에서 컴포넌트를 조립할 수 있는 다양한 아키텍처 모델이 제안되었다. 대부분의 아키텍처 모델들이 메시지를 이용하여 컴포넌트를 이용하는 방안을 제시하였다. 본 논문에서는 메시지를 이용하지 않고, 업무 흐름에 따라 컴포넌트를 직접 호출하는 방법으로 컴포넌트의 성능을 최대한 발휘하면서, 소프트웨어 개발의 시간과 비용을 줄일 수 있는 아키텍처 모델을 제안한다.

1. 서론

소프트웨어의 수요는 기하 급수적으로 증가하는 추세에 있으나, 개발자들의 생산성과 개발 인력의 공급은 상대적으로 미미한 수준의 신장을 보이고 있다. 그러나 경영환경이 빠르게 변화하면서 이를 반영하는 정보 시스템 역시 점점 더 고도화되고 복잡해지고 있다. 이로써 서비스 공급자나 개발자들이 사용자의 요구에 맞는 시스템을 개발해서 제때에 공급하는 것은 점점 더 어려워지고, 시스템에 새로운 기능을 추가함에 따라 변경의 파급이라는 문제점을 발생시킬 위험성도 증가하고 있다.

이러한 소프트웨어의 한계성을 극복하기 위하여 재사용 가능한 소프트웨어 컴포넌트의 개발과 이를 이용하는 응용 소프트웨어의 개발에 관한 많은 연구가 진행되어 왔다[1][4].

그러나, 독립적 재사용 가능한 컴포넌트 만으로 전체 응용 소프트웨어의 복잡도를 해결할 수 없기 때문에 아키텍처를 기반한 개발 소프트웨어 프로세스 모델이 등장하고 있다. 컴포넌트 기반 소프트웨어 개발(Component-Based Software Development; CBSD)에 있어서 시스템에 대한 상위 수준의 추상화를 나타내는 소프트웨어 아키텍처(Software Architecture)는 소프트웨어 개발에 참여하는 사람들간의 원활한 의사 소통과 시스템 설계 결정에 대한 합리적 판단을 가능하게 하며, 소프트웨어 개발에 참여하는 사람들이 모두 인지 할 수 있는 형태로 시스템 구조와 그 시스템을 구성하는 컴포넌트들의 동작을 보여 준다. 이러한 장점 때문에 고품질 소프트웨어를 개발하는데 있어서 필수적이라고 할 수 있다[4][6].

지금까지 제안된 대부분의 아키텍처 모델들은 메시지를 이용하여 원격(Remote) 컴포넌트를 이용하는 방안을 제시하였다. 이러한 모델들은 지역(Local) 컴포넌트 또한 네트웍을 통해 원격으로 요청해야 하며, 메시지 방식의 간접 호출이기 때문에, 메시지 처리에 의한 작업 수행시간의 지연이 발생한다.

본 논문에서는 메시지를 사용하지 않고, 업무 흐름에 따라 컴포넌트를 직접 호출하는 방법으로 컴포넌트의 성능을 최대한 발휘하면서, 소프트웨어 개발에 필요한

시간과 비용을 줄일 수 있는 아키텍처 모델을 제안한다.

2. 관련 연구

이 장에서는 컴포넌트 기반 소프트웨어 개발 방법, 소프트웨어 아키텍처의 정의와 아키텍처 명세들에 간략하게 정리한다.

2.1 컴포넌트 기반 소프트웨어 개발

컴포넌트는 독립적 실행 단위의 소프트웨어 모듈이다.

컴포넌트는 인터페이스에 의해서만 서비스를 제공함으로써 구현의 변경이 쉽고, 인터페이스 기반의 구현이 가능하다.

컴포넌트 기반 소프트웨어 개발이란 해당 업무의 수행에 필요한 기능을 담당하는 하나 이상의 컴포넌트를 결합하여 해당 업무를 위한 소프트웨어를 개발하는 기술을 의미한다. 이러한, 컴포넌트 기반 소프트웨어 개발이 갖는 장점을 정의하면 다음과 같다.

첫째, 개발자는 자신의 요구에 부합하는 품질 좋은 컴포넌트를 선택하여 재사용할 수 있다.

둘째, 기존에 개발된 컴포넌트를 조립해 새로운 어플리케이션을 개발함으로써 기간을 단축할 수 있으며 기존의 컴포넌트를 재사용 할 수 있어 생산성과 경제성을 높일 수 있다.

셋째, 컴포넌트는 분산환경에서 반복적 분산 배치가 가능하므로 시스템의 대형화와 통합화에 유연하고 신속하게 대처할 수 있다.

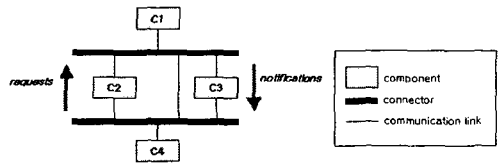
넷째, 기존 시스템의 보완개발과 통합 문제에 쉽게 접근할 수 있다.

2.2 소프트웨어 아키텍처와 명세 언어

소프트웨어 아키텍처는 컴포넌트들의 구조, 상호 관계, 설계 및 변경을 가이드 하는 원칙 등을 나타내는 시스템에 대한 상위 수준의 추상화이다[6].

(1) C2-Architecture

C2-Architecture Style 은 UCI(University of California in Irvine)에서 본래 GUI(Graphic User Interface)를 갖는 응용 소프트웨어 개발을 지원하기 위하여 설계한 아키텍처 스타일이지만 다른 타입의 응용 소프트웨어 개발도 지원한다.



[그림 1] C2-Architecture Style

C2-Architecture Style 은 메시지 기반의 계층적 아키텍처 스타일로 [그림 1]에 정의한 것과 같이 컴포넌트와 커넥터들로 구성된다.

C2-Architecture 에서의 컴포넌트는 일반적 컴포넌트에 두 개의 포트(Top-port, Bottom-port)를 추가하여 확장하였다. 확장된 컴포넌트는 두 개의 포트를 통해 들어오는 메시지에 대해 내부적으로 선언된 메소드를 호출(Invocation)하거나 포트를 통해 나갈 메시지를 생성하여 요청(Request) 메시지를 보내거나, 통지(Notification) 메시지를 받는다.

아키텍처를 구성하는 각 컴포넌트들 간의 상호작용은 메시지의 교환으로 이루어진다.

아키텍처내의 커넥터(Connector)는 컴포넌트 간에 교환되는 메시지를 라우팅(Routing), 브로드캐스팅(Broadcasting) 또는 메시지 필터링(Message Filtering)하는 역할을 한다[2].

각 컴포넌트는 포트에 메시지를 명시하여, 커넥터를 통해 다른 컴포넌트의 포트에 전달하는 형태의 간접 통신을 지원하며, 컴포넌트 간의 직접 통신은 지원하지 않는다. 커넥터는 복수 개의 컴포넌트들 또는 또 다른 커넥터들과 연결될 수 있다.

(2) Rapide

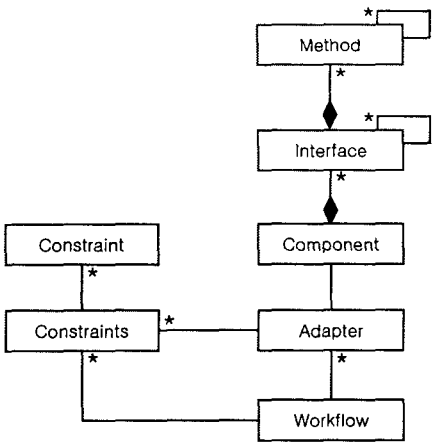
Rapide 는 1990 년에 ARPA(Advanced Research Projects Agency) 에 의해 분산 시스템 구조의 시뮬레이션과 분석 프로토타이핑을 위해 설계된 이벤트 기반 동시 객체 지향 언어(Event-Based Concurrent Object-Oriented Language)이다[3]. Rapide 의 주된 설계 목표는 구현을 결정하기 위한 시뮬레이션을 위해 시스템 구조를 실행 가능한 형태로 표현할 수 있도록 하는 아키텍처를 제공한다. 그리고, 분산 행위와 타이밍을 정확하게 찾아내는 실행 모델(Execution Model)을 채택했다. 또한, 참조 아키텍처의 제약 기반 정의와 구조적 표준과의 일치를 확인하도록 정형화된 제약 조건과 매핑을 제공한다. Rapide 에서 아키텍처는 모듈에 대한 명세 집합, 인터페이스간 직접적인 통신을 정의하는 연결 규칙의 집합 그리고 통신 패턴을 정의하는 정형화된 제약 조건으로 구성된다. 인터페이스는 함수와 같은 다른 모듈 인터페이스에 제공되는 특징들에 대해 정의하는 것으로써 모듈의 행위에 대한 추상적인 정의가 가능하다. 또한, 인터페이스 행위는 모듈에 의해 수신된 데이터와 생성된 데이터 간의 관계를 명세한다[3][5].

연결은 인터페이스간 데이터에 대한 동기적, 비동기적 통신에 대해 정의한다. 그리고, 아키텍처 제약 조건은 POSET(Partially Ordered SET)모델이라고도 불리는 이벤트 기반 실행 모델을 가진다. Rapide 에서 아키텍처는 모듈을 기술하기 전에 정의하고, 모듈들을 결합하기 위한 프레임워크이며, 모듈간 통신을 제약한다[5].

3. 워크플로우 기반 아키텍처 모델

소프트웨어 개발이란 업무 흐름이나 규칙을 정의해 나아가는 것이라 할 수 있다. 이러한 흐름과 규칙이 미리 정의되어 있다면, 개발자들은 관련 컴포넌트를 찾아내어 정의한 흐름과 규칙에 맞게 조립을 하면 된다.

컴포넌트를 조립할 수 있도록, 정의된 흐름과 규칙을 이용하여 소프트웨어 아키텍처를 만들고, 이렇게 만들어진 아키텍처에 컴포넌트를 쉽게 조립할 수 있는 방법으로써, [그림 2]와 같은 아키텍처 개념 모델을 제안한다.



[그림 2] 워크플로우 기반 아키텍처 모델

각각의 구성요소의 의미와 명세 구문을 정의하면 다음과 같다.

(1) 컴포넌트(Component)

본 논문에서는 분산 컴포넌트 및 지역 컴포넌트 모두를 대상으로 하며 JavaBeans 와 EJB 모델의 컴포넌트를 의미한다. [리스트 1]은 컴포넌트를 명세하기 위해 정의한 명세 구문이다.

[리스트 1] 컴포넌트 명세를 기술하기 위한 명세 구문

```
Component
   Component name : <component_name>
   Specification : <component_spec>
   Package : <component_package>
   Constructor : <constructor>
```

(2) 인터페이스(Interface)

인터페이스는 아키텍처와 컴포넌트 간의 통신 수단을 제공한다. 인터페이스 기술시에는 인터페이스와 메소드들의 정보를 기술하며, 기술된 메소드들은 워크플로우 명세 기술시에 사용된다. [리스트 2]는 인터페이스를 명세하기 위해 정의한 명세 구문이다.

[리스트 2] 인터페이스 정보를 기술하기 위한 명세 구문

```
Interface
   Interface name : <interface_name>
      Method name : <method_name>
      Invoke method : <invoke_method_name>
      Parameter : <para_list>
      Return : <return_type>
      Constraint : <constraint_name>
```

(3) 제약조건(Constraint)

제약조건은 어떠한 작업을 수행하기 위해 필수적으로 만족해야 할 상태 들을 정의한다.

제약조건은 아키텍처 내에서 재사용될 수 있도록 별도로 관리하며, 워크플로우에서 호출한 어댑터 내와 워크플로우 명세 기술시에 사용하게 된다. [리스트 3]은 제약조건을 명세하기 위해 정의한 명세 구문이며, [리스트 4]는 제약조건 명세에 기술된 각각의 제약조건 항목을 명세하기 위해 정의한 명세 구문이다.

[리스트 3] 제약조건 기술을 위한 명세 구문

```
Constrains
   Constraint name : <constraint_name>
   Parameter : <para_list>
   ConstraintItem :
      void
        | {<vari_name> '::=' <constraint_item_name>
                                    '('<para_name_list>')';}
```

[리스트 4] 제약조건 항목 기술에 필요한 명세 구문

```
ConstraintItem
      ConstraintItem name : <constraint_item_name>
      Parameter : <para_list>
      Logic
            <constraint_logic>
```

여기서 정의되는 모든 제약조건의 조건 검사 후 반환되는 값은 해당 제약조건을 만족하는지의 여부만을 나타내며, boolean 형으로 True/False 값만을 반환한다.

(4) 어댑터(Adapter)

어댑터는 아키텍처와 컴포넌트를 연결하여 주는 역할을 하며, 컴포넌트의 객체 생성 및 소멸을 관리하는 역할도 포함한다. [리스트 5]는 지역 컴포넌트를 어댑팅하기 위해 정의한 어댑터 명세 구문이며, [리스트 6]은 원격 컴포넌트를 명세하기 위해 정의한 어댑터 명세 구문이다.

[리스트.5] 지역 서버에 있는 컴포넌트를 어댑팅하기 위한 LocalAdapter 의 명세 구문

```
LocalAdapter
   Adapter name : <adapter_name>
   Component
        <component_info>
```

[리스트.6] 원격 서버에 있는 컴포넌트를 어댑팅하기 위한 RemoteAdapter 의 명세 구문

```
RemoteAdapter
   Adapter name : <adapter_name>
   Server : <server_name>
   Port : <port_number>
   Component
        <component_info>
```

컴포넌트를 새로운 것으로 변경 및 컴포넌트의 수정으로 인한 인터페이스의 이름 변경으로 시스템을 유지보수 해야 할 경우에는 어댑터의 수정만으로 해결이 가능하다.

(5) 워크플로우(Workflow)

특정 작업에 대한 인터페이스 호출 흐름을 관리하며, 이 부분에서 정의되는 워크플로우들이 실제 조립된 컴포넌트를 이용할 사용자에게 제공되어지는 인터페이스가 된다.

워크플로우가 수행시에 메소드의 수행 순서는 메소

드 플로우에 기술된 순서대로 순차적으로 수행된다. [리스트 7]은 워크플로우를 명세하기 위해 정의한 명세 구문이다.

[리스트 7] 워크플로우 명세 구문

```
Workflow
    Workflow name : <workflow_name>
    Return : <return_type>
    Parameter : <para_list>
    User variable : <vari_list>
    Method flow
        <method_flow>
```

메소드 호출 후 해당 결과값을 다른 메소드의 파라미터로 사용할 수 있으며. 메소드 호출시 결과값의 저장은 선택적으로 할 수 있다.

또한, 실제 작업에 대한 메소드 호출 흐름을 기술하는 부분이기 때문에 특정 메소드의 반복 호출([리스트 8])이나 메소드 호출 후의 반환결과값을 이용한 분기([리스트 9])에 대한 부분 또한 기술할 수 있다.

[리스트 8] 분기를 표현하기 위한 명세 구문

```
<branch_flow>
  ::=  [<method_flow>]
       BranchConstraint.<constraint_name>
       MasterBranch
            [<method_flow>]
       AlternativeBranch
            [<method_flow>]
       EndBranch
       [<method_flow>]
```

[리스트 9] 반복을 표현하기 위한 명세 구문

```
<loop_flow>
  ::=  [<method_flow>]
       Loop
            [<method_flow>]
            BreakConstraint.<constraint_name>
            [<method_flow>]
       EndLoop
       [<method_flow>]
```

위의 분기 명세와 반복 명세를 이용하여 분기와 반복이 혼합된 혼합 명세도 가능하다.

[리스트 10]은 정의된 각 명세 구문에서 반복적으로 사용되는 명세 구문을 정리한 것이다.

[리스트.10] 반복되는 명세 구문

```
<method_flow>
  ::= {<branch_flow> | <loop_flow> | <method>}
<method>
  ::= [<vari_name> '::=' ]
      <adapter_name>.<method_name>'('<para_name_list>')';
<para_name_list>
  ::= void
      |(<para_name>|<vari_name>)
        {, (<para_name>|<vari_name>)}
<para_list>
  ::= void
      | <para_type> <para_name>{, <para_type> <para_name>}
<vari_list>
  ::= void
      | <vari_type> <vari_name>{, <vari_type> <vari_name>}
```

4. 결론 및 향후 연구

본 논문은 소프트웨어 시스템 개발시에 아키텍처를 통해 컴포넌트의 기능 수행시간을 최소화 시키면서 합성할 수 있는 아키텍처 모델을 제안하였다.

C2 나 Rapide 와 같이 메시지 패싱을 하는 아키텍처는 컴포넌트의 기능을 수행하기 전에 해당 기능을 수행하려는 메시지인지를 식별하는 시간이 추가된다. 게다가, C2 의 경우에는 커넥터를 통한 브로드케스팅 방식의 호출이기 때문에 메시지를 커넥터에 연결된 많은 컴포넌트들이 받아 자신이 처리 가능한 메시지인지를 식별하며, 최악의 경우는 컴포넌트에 연결된 커넥터에 다시 요청하고, 커넥터가 다시 각 컴포넌트들에게 요청하는 형태가 반복적으로 일어날 수 있기 때문에, Rapide 와 같이 직접 호출을 하는 것보다 더 시간적 효율이 떨어지게 된다. 이러한 이유로 제안하는 아키텍처는 메시지를 이용하지 않고, 메소드를 직접 호출하는 방식을 사용하여 시간적 효율성을 높였다. 또한, C2 의 경우 메시지 흐름이 복잡해 업무 흐름을 한눈에 파악하기 힘든 단점을 워크플로우를 아키텍처에 적용하여, 업무 흐름 파악 및 아키텍처 구성을 쉽게 할 수 있도록 하였다.

향후 명세 구문의 확장 및 수정을 쉽게 할 수 있도록 명세 구문을 XML 형태로 변환하여 적용할 예정이다.

참고문헌

[1] Butler Group, Component-Based Development: Application Delivery and Integration Using Componentised Software, 1998.

[2] Nenad Medvidovic, "Formal Definition of the Chiron-2 Software Architectural Style," Techincal Report

[3] Rapide Design Team, "DRAFT Guide to the Rapide 1.0 Language Reference Manuals," Program Analysis and Verification Group, Computer System Lab, Stanford University, July, 1997.

[4] David Garlan and Mary Shaw, "An Introduction to Software Architecture," Technical Report CMU-CS-94-166, Carnegie Mellon University, January, 1994.

[5] 전주현, "아키텍처 기반의 컴포넌트 검색 시스템", 공학 박사 학위 논문, 관동대학교, 2002.

[6] 심우곤 외 3 명, "CBSD 활성화를 위한 확장된 부가가치 중개 개념", 정보처리학회지 제 8 권 제 6 호, 2001.