컴포넌트 합성을 위한 워크플로우 기반 S/W 아키텍쳐 모델의 XML 명세

조광윤*, 서효길**, 홍찬기*
*관동대학교 전자계산공학과
**(주)엔텔스
e-mail : doorman78@kwandong.ac.kr
cbells@nate.com
chankih@kwandong.ac.kr

XML Specification of Workflow-Based S/W Architecture for Component Composition

Kwangyun Cho*, Hyogil Seo**, Chanki Hong*
*Dept of Computer Science, Kwandong University
**nTels co., Ltd

요 약

최근 소프트웨어의 수요가 증가함에 따라 수요를 충족시키기 위한 다양한 응용 소프트웨어 개발 방식 중 컴포넌트 기반 소프트웨어 개발(CBSD: Component-Based Software Development) 기술이 빠르게 발전하였고, 이에 실제 개발된 컴포넌트의 합성을 통한 커다란 응용 소프트웨어 개발이 활발히 진행되고 있다. 또한 컴포넌트들이 서로 정확하게 합성되고, 작동할 수 있는 방법으로 소프트웨어 아키텍쳐 기반의 합성 환경에 대한 연구들이 진행되고 많은 방법들이 제안되었다[1].

이에 본 논문에서는 지금껏 제안된 아키텍쳐 모델들이 갖고 있는 소프트웨어 개발에 있어 전체적인 흐름파악의 어려움과 변경의 파급효과라는 문제점들을 해결 및 최소화 시킬 수 있는 방법으로 제안된 워크플로우라는 개념을 사용한 소프트웨어 아키텍쳐 모델에 대한 추가 연구와 더불어 아키텍쳐의 명세를 XML로 정의하므로, 명세 구문의 수정 및 확장이 용이하도록 하였다.

1. 서론

소프트웨어 개발의 규모와 복잡도가 급격히 증가함에 따라 개발자들이 사용자의 요구를 충족시키면서, 이를 적시에 공급할 수 있는 것이 점점 어려워지고 있으며, 기존의 소프트웨어의 새로운 기능을 추가함에 있어 변경의 파급이라는 문제점이 발생할 위험성이 증가되었다.

이에 컴포넌트 기반 소프트웨어 개발(CBSD: Component-Based Software Development)이 위의 문제점을 해결할 방법으로 연구가 활발히 진행되고 있다. CBSD의 목적은 새로운 소프트웨어를 개발하고자 할 때 검증된 컴포넌트를 재사용함으로써 생산성 및 품질을 향상하고자 하는 것이다. 하지만, 독립적이고 재사용 가능한 컴포넌트만으로 소프트웨어 개발의 전체적인 복잡도를 해결할 수 없기 때문에 소프트웨어 아키텍쳐(Software Architecture)를 기반한 개발 소프트웨어 프로세스 모델이 등장하고 있다. 소프트웨어 아키텍쳐는 개발에 참여하는 모든 사람들 간의 원활한 의사소통 및 시스템의 합리적 설계와 함께 전체적 시스템의 구조, 컴포넌트들의 동작 등을 쉽게 인지할 수 있다는 장점 때문에 고품질 소프트웨어 개발하는데 있어서 필수적이라 할 수 있다.

지금까지 제안된 대부분의 아키텍쳐 모델들은 하나의 서비스를 제공하기 위해 각각의 컴포넌트들 내부에 기능들 및 호출 흐름을 정의함으로써 서비스에 대한 전체적 흐름을 쉽게 파악할 수 없고, 흐름 변경시에도 관련 컴포넌트 내부의 정의되어 있는 기능들의 호출 흐름을 재정의 하는 문제점을 안고 있다. 이에 본 논문에서는 이러한 문제점을 보완하기 위해 제안된 워크플로우 기반 소프트웨어 아키텍쳐 모델[5]을 추가 연구를 하고, 이 아키텍쳐 명세를 XML

으로 정의하였다.

2. 관련연구

이 장에서는 CBSD와 소프트웨어 아키텍쳐에 대하여 간략히 정리한다.

2.1 CBSD

컴포넌트 기반 소프트웨어 개발(CBSD)이란, 독립적인 기능을 담당하는 다양한 컴포넌트 소프트웨어의 집합으로부터 해당 업무의 수행에 필요한 기능을 담당하는 하나 이상의 컴포넌트를 결합하여, 해당 업무를 위한 소프트웨어를 개발하는 기술을 의미한다. 이러한 컴포넌트 기반 소프트웨어 개발이 갖는 이점은 다음과 같다.

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

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

셋째, 컴포넌트는 분산 환경에서 반복적 분산 배치가 가능하므로 시스템 대형화와 통합화에 유연하고 신속하게 대처할 수 있다.

넷째, 기존 시스템의 보완개발과 통합 문제에 접근할 수 있다.

2.2 소프트웨어 아키텍쳐

소프트웨어 아키텍쳐는 컴포넌트들의 구조, 상호관계, 설계 및 변경을 가이드하는 원칙 등을 나타내는 시스템에 대한 상위 수준의 추상화이다[6].

소프트웨어 아키텍쳐가 제공하는 이점을 정리하면 다음과 같다.

첫째, 상호 의사소통을 원활하게 한다.

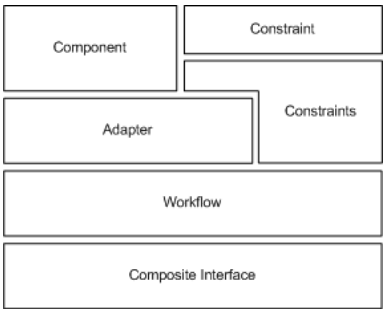
둘째, 시스템의 초기 설계를 포함한다.

셋째, 시스템의 추상화를 제공한다.

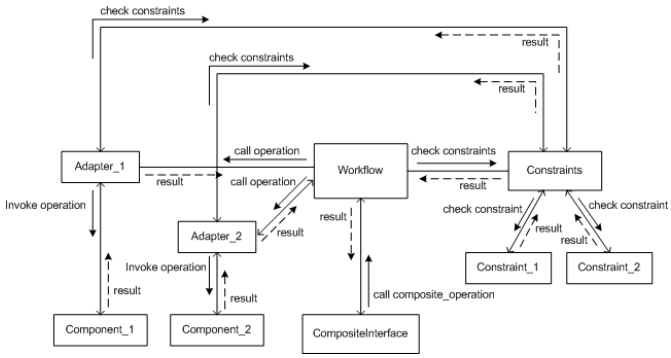
3. 워크플로우 기반 S/W 아키텍쳐

워크플로우란 업무의 흐름을 의미한다. 컴포넌트를 합성하여 업무를 수행하는 소프트웨어를 개발한다는 것은 컴포넌트가 제공하는 기능들을 업무 수행에 필요한 흐름과 규칙에 따라 정의하여 아키텍쳐를 구성하는 것을 의미한다. 제안된 워크플로우 기반 S/W 아키텍쳐는 업무 수행에 필요한 인터페이스 호출 흐름을 따로 관리하는 형태를 취하는 아키텍쳐 형태를 갖는다.

컴포넌트를 조립할 수 있도록 정의된 흐름과 규칙을 이용하여 소프트웨어 아키텍쳐를 만들고, 이렇게 만들어진 아키텍쳐에 컴포넌트를 쉽게 조립할 수 있는 방법으로, (그림 1)과 같은 계층 구조도를 갖는다.



(그림 1) 워크플로우 기반 아키텍쳐 구조도



(그림 2) 아키텍쳐 구성 요소 호출 흐름도

(그림 2)는 워크플로우 기반 S/W 아키텍쳐 모델의 구성 요소들과 호출 흐름을 도식화 한 것이다.

각각의 구성요소의 의미와 명세 구문을 XML로 정의하면 다음과 같다.

(1) 컴포넌트(Component)

본 논문의 컴포넌트는 분산 컴포넌트 및 지역 컴포넌트 모두를 지원하며 JavaBeans와 EJB 모델의 컴포넌트를 의미한다. (리스트 1)은 컴포넌트의 명세를 XML로 정의한 명세 구문이다.

(리스트 1) 컴포넌트 XML 명세 구문

```
<component name="컴포넌트명">
    <spcification>스펙</spcification>
    <package>패키지</package>
    <constructor name="생성자명" />
        ..... <!-- 인터페이스 XML 명세 부분 -->
    <note>추가 설명</note>
</component>
```

(2) 인터페이스(Interface)

인터페이스는 아키텍처와 컴포넌트 간의 통신 수단을 제공한다. 인터페이스 기술시 인터페이스 및 메소드의 정보를 기술하며, 기술된 메스드들은 워크플로우 명세 기술시에 사용된다. (리스트 2)는 인터페이스의 명세를 XML로 정의한 명세 구문이다.

(리스트 2) 인터페이스 XML 명세 구문

```
<interface name="인터페이스명">
 <operation name="오퍼레이션명" return_type="반환타입">
  <parameter_list>
   <parameter_item type="파라미터타입" name="파라미터명" />
       </parameter_list>
  </operation>
</interface>
```

(3) 제약조건(Constraint)

제약조건은 어떠한 작업을 수행하기 위해 필수적으로 만족해야 할 상태들을 정의하는 것으로, 아키텍처 내에선 재사용될 수 있도록 별도로 관리하며, 워크플로우에서 호출한 어댑터와 워크플로우 명세 기술시 사용하게 된다. (리스트 3)은 제약조건의 명세를 XML로 정의한 명세 구문이며, (리스트 4)는 제약조건 명세에 기술된 각각의 제약조건의 항목에 대한 명세를 XML로 정의한 명세 구문이다.

(리스트 3) 제약조건 XML 명세 구문

```
<constraints name="제약조건명" return_variable="반환할 변수명">
 <parameter_list>
  <parameter_item type="파라미터타입" name="파라미터명" />
 </parameter_list>
 <call_constraint_item name="제약조건항목명"
                          return_variable="반환값을 저장할 변수명">
  <argument_list>
    <argument_item name="파라미터명" />
   </argument_list>
 </call_constraint_item>
<call_constraints name="제약조건명"
                          return_variable="반환값을 저장할 변수명">
<argument_list>
  <argument_item name="파라미터명" />
</argument_list>
  </call_constraints>
</constraints>
```

(리스트 4) 제약조건 항목 XML 명세 구문

```
<constraint_item name="제약조건항목명">
    <parameter_list>
       <parameter_item type="파라미터타입" name="파라미터명" />
    </parameter_list>
    <logic>
        <![CDATA[      <!-- 제약조건 검사 코드 -->      ]]>
    </logic>
</constraint_item>
```

(4) 어댑터(Adapter)

어댑터는 아키텍처와 컴포넌트를 연결하여 주는 역할과 컴포넌트의 객체 생성 및 소멸을 관리하는 역할을 한다. (리스트 5)는 지역 컴포넌트(local component)를 어댑팅하기 위한 어댑터의 명세를 정의한 XML로 정의한 명세 구문이고, (리스트 6)은 원격 컴포넌트(remote component)를 어댑팅하기 위한 원격 어댑팅의 명세를 XML로 정의한 명세 구문이다. (리스트 7)은 각각의 어댑터에 공통적으로 들어갈 명세를 XML로 정의한 명세 구문이다.

(리스트 5) 지역 어댑터 XML 명세 구문

```
<adapter name="어댑터명">
    <local_adapter component_name="컴포넌트명" />
    <!-- 어댑터 공통 XML 명세 부분 -->
</adapter>
```

(리스트 6) 원격 어댑터 XML 명세 구문

```
<adapter name="어댑터명">
  <rocal_adapter component_name="컴포넌트명"
     server="서버명 또는 서버 IP" port="포트번호"
                    protocol="프로토콜" namig="검색할 이름" />

   <!-- 어댑터 공통 XML 명세 구문 -->
</adapter>
```

(리스트 7) 어댑터 공통 XML 명세 구문

```
<instance_interface invoke_interface="인터페이스명"
                        instance_name="인터페이스의 인스턴스명">
    <call_constructor name="생성자명">
       <argument_list>
            <argument_item name="파라미터명" />
          </argument_list>
       </call_constructor>
    </instance_interface>
    <adapted_operation name="오퍼레이션명">
      <parameter_list>
        <parameter_item type="파라미터타입" name="파라미터명" />
      </parameter_list>
 <instance_interface_name>
              인터페이스의 인스턴스명
       </instance_interface_name>
       <invoke_operation name="오퍼레이션명">
          <argument_list>
             <argument_item name="파라미터명" />
          </argument_list>
       </invoke_operation>
       <invoke_constraints name="제약조건명">
          <argument_list>
             <argument_item name="파라미터명" />
          </argument_list>
       </invoke_constraints>
    </adapted_operation>
```

(5) 워크플로우(Workflow)

워크플로우는 아키텍처 모델의 중요한 요소로서, 컴포넌트의 호출 흐름을 관리하는 역할을 한다. 정의되는 워크플로우들이 실제 합성된 컴포넌트를 이용할 사용자에게 제공되어지는 인터페이스가 된다.

워크플로우 수행 시에 메소드 수행 순서는 메소드에 기술된 순서대로 수행한다. (리스트 8)은 워크플로우의 명세를 XML로 정의한 명세 구문이다.

(리스트 8) 워크플로우 XML 명세 구문

```
<workflow name="워크플로우명" return_variable="반환할 변수명">
    <parameter_list>
        <parameter_item type="파라미터타입" name="파라미터명" />
    </parameter_list>
    <variable_list>
        <variable_item type="변수타입" name="변수명" />
    </variable_list>
<operation_flow>
     <call_operation adapter_name="어뎁터명"
                    operation_name="오퍼레이션명"
                     return_variable="반환값을 저장할 변수명">
          <argument_list>
              <argument_item name="파라미터명" />
          </argument_list>
        </call_operation>
        <branch_flow>
         <branch_constraints name="제약조건명"
                       return_variable="반환값을 저장할 변수명">
              <argument_list>
                  <argument_item name="파라미터명" />
              </argument_list>
          </branch_constraints>
          <master_branch>
              <!-- operation_flow XML 명세 구문 -->
          </master_branch>
              <!-- operation_flow XML 명세 구문 -->
          <alternative_branch>
          </alternative_branch>
          </branch_flow>
        <logic>
            <![CDATA[  <!-- 추가 코드 --> ]]>
        </logic>
        <loop_flow>
            <!-- operation_flow XML 명세 구문 -->
        </loop_flow>
        <break_constraints name="제약조건명"
          return_variable="반환값을 저장할 변수명">
          <argument_list>
              <argument_item name="파라미터명" />
          </argument_list>
        </break_constraints>
</operation_flow>
    <call_constraints name="제약조건명"
                      return_variable="반환값을 저장할 변수명">
        <argument_list>
            <argument_item name="파라미터명" />
        </argument_list>
    </call_constraints>
</workflow>
```

4. 결론 및 향후연구 과제

소프트웨어 시스템을 개발한다는 것은 특정 업무에 대한 흐름과 제약조건을 명시하는 일련의 작업이라 볼 수 있다. 그러나 이러한 업무 흐름과 제약조건을 각각의 컴포넌트에 분산하여 관리하게 된다면, 추후 업무 흐름변경 및 컴포넌트의 변경시에 관련된 모든 컴포넌트에 영향을 주게 되며, 전체적인 흐름을 파악하기 어렵다. 따라서 이러한 흐름 및 제약조건을 각각의 컴포넌트에 분산하여 관리하지 않고, 아키텍쳐 내에 정의하고 직접 관리하여, 소프트웨어 시스템의 전체적인 흐름 파악 및 유지보수 발생 시에 관련 컴포넌트들의 수정 없이 해당 업무 흐름 및 컴포넌트를 어뎁팅하고 있는 어뎁터만을 수정하여 손쉽게 대처할 수 있다.

이러한 업무 흐름 및 제약조건을 개별적으로 관리하기 위해서 제안된 워크플로우라는 개념을 사용한 소프트웨어 아키텍쳐 모델이다. 본 논문에서는 제안된 아키텍쳐 모델의 추가 연구를 통해서 아키텍쳐의 구성 요소들의 호출 흐름을 원활히 관리될 수 있도록 워크플로우 및 조건항목을 보완하고, 아키텍쳐의 구성 요소들의 명세를 XML로 정의하였다. XML으로 명세 구문을 사용할 경우에는 파서를 따로 개발할 필요 없이 기존에 제공되는 XML 파서를 이용하여 기술한 명세에서 원하는 정보를 쉽게 얻어낼 수 있다는 장점이 있다. 또한, 명세 구문의 확장이나 변경이 발생하더라도 파서를 새로 개발할 필요 없이 약간의 수정만으로도 쉽게 명세 구문의 확장이나 변경이 가능하다.

향후 연구과제로는 아키텍쳐의 명세 정보를 바탕으로 코드 생성을 하는 코드 생성기에 대한 구현과 함께 제약조건 항목과 오퍼레이션 플로우 중간 중간에 삽입된 특정 언어에 종속적인 코드를 일반적인 형태의 스크립트로 변환할 수 있는 방법에 대한 연구가 필요하다.

참고문헌

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

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

[3] 전주현, "아키텍쳐 기반의 컴포넌트 검색 시스템", 공학박사 학위논문, 관동대학교, 2002.

[4] 이승연 외 3명, "아키텍쳐 스타일 기반의 컴포넌트 조립 및 지원도구의 개발", 정보처리학회 제9권 제1호, 2001.

[5] 서효길 외 1명, "컴포넌트 조립을 지원하는 워크플로우 기반의 아키텍쳐 모델", 정보처리학회 제9권 제2호, 2002.

[6] 심우곤 외 3명, "CBSD 활성화를 위한 확장된 부가가치 중개 개념", 정보처리학회 제8권 제6호, 2001.