컴포넌트 통합을 위한 프로세스에 관한 연구

○
김 행 곤*, 신 호 준*, 한 은 주*
*대구효성가톨릭대학교 컴퓨터공학과
{hangkon, g98521002, ejhan}@cuth.cataegu.ac.kr

Study on the Process for Component Integration

○
Haeng-Kon Kim*, Ho-Jun Shin*, Eun-Ju Han*

*Software Engineering Lab., Dept. of Computer Engineering,
Catholic University of Taegu Hyosung

요 약

잘 정의된 인터페이스를 통해서 의미있는 서비스를 유도하거나 기술하는 소프트웨어의 식별가능한 조각인 컴포넌트를 개발하기위한 방법론은 최근에 주목받고 있는 부분이다. 컴포넌트 기반 개발은 컴포넌트를 개발하기위한 단계와 이를 바탕으로 하나의 애플리케이션이나 시스템을 구축하는 과정이 병렬적으로 진행되는 프로세스가 특징적이다. 이러한, 단위 기능의 요소로써 컴포넌트는 새로운 요구사항에 대한 대처가능성이나 시로의 계약을 통한 조합이 가능한 것은 통합(integration)에 대한 개념이 기반이 되고 있다.

본 논문에서는 상이하게 개발된 컴포넌트 통합의 전반적인 개념을 소개하고, 단일한 컴포넌트의 통합과 통합된 컴포넌트 군을 재통합하는 메커니즘을 정의한다. 또한, 이에 따르는 컴포넌트 기반 개발의 전반적인 프로세스와 통합에 관련된 로직을 정의함으로써 체계적으로 컴포넌트를 개발할 수 있도록 도움을 주고, 컴포넌트 통합에 의한 재사용성과 효율성을 높이고자 한다.

1. 서론

1990년대 후반 컴포넌트의 개념이 소개되고 확대 연구되면서 컴포넌트의 수용에 대한 관심이 학계와 업계에 빠르게 전파되고 있다. 이는 단위기능을 수행하는 컴포넌트가 가지는 잠재성이 신뢰받을 수 있는 컴포넌트의 재사용성과 통합에 의한 소프트웨어 개발시간의 단축이라는 장점이라 할 수 있다. 이러한 컴포넌트 개발을 위해 기반되는 소프트웨어 아키텍처와 명세에 대한 연구 뿐만 아니라, 컴포넌트 프로그래밍, 통합, 검증과 검사를 통한 컴포넌트 표준화 작업등에 관련하여 많은 연구가 되고 있다. 특히, 통합은 수행성과 안정성이 검증된 컴포넌트를 기반으로 안정된 시스템으로 구성할 수 있는 부분에서 중요성이 크다고 볼 수 있다.

본 논문에서는, 아직 명확하게 정의되지 않은 통합에 대한 전반적인 개념을 살펴보고, 새로운 애플리케이션을 개발하기위한 통합 메카니즘을 제시한다. 즉, 상이하게 개발된 컴포넌트를 기능이나 식별 될 수 있는 통합요소에 따라 포트(port), 속성(attribute), 아키텍처(architecture) 기반의 단일한 컴포넌트의 통합 방법과 통합 방식에 따라 다른 형태의 컴포넌트 군을 새로운 애플리케이션 컴포넌트로 구성하기위한 방법으로 점대점 연결, 어댑터를 통한 연결, 허브를 통한 연결 방법을 정의하고, 이에 따르는 컴포넌트 기반 프로세스와 통합 로직을 구성하였다.

2. 관련연구

2.1 컴포넌트 특징

컴포넌트는 단순한 관련 클래스들의 조합이 아니라 능동적으로 단위기능을 수행할 수 있는 소프트웨어로 발전되어 가고 있으며, 모델기반의 컴포넌트 개발로 파라다임이 진화되고 있다. 특히, 컴포넌트를 인터페이스 부분과 구현 부분으로 분리 설계하고 있는 것이 두드러진 특징이다. 컴포넌트가 물리적으로 또한, 논리적이거나 개념적인 형태로 존재하더라도 기본적으로 요구되는 사항과 설계가 개발과정에 포함되어 선택적으로 기대될 수 있는 특징을 가지고 있다[1].

- 식별가능
- 전체 개발 라이프사이클에 추적가능
- 같은 서비스를 제공하는 컴포넌트로 대체가능
- 인터페이스만을 통해 접근가능하며 제공되는 서비스의 불변
- 정확한 문서화 제공
- 물리적인 구현은 숨김
- 다른 컴포넌트와 독립적
- 캡슐화
- 서비스의 재사용은 물리적인 구현에 의해 제약되지 않음
- 동적인 재사용 가능
- 특정한 요구에 적용할 수 있는 일반적인 서비스 제공

잘 정의된 아키텍처상에서 어떤 기능을 수행하는 시스템의 독립적면서도 대치 가능한 부분으로 인터페이스들의 집합에 대한 물리적 구현을 제공하는 컴포넌트는 크게 네 가지 타입으로 나눌수 있다.

- Implementation Specific Components
  클래스 라이브러리처럼 추상화의 단계가 저수준인 컴포넌트를 하며, 특정 언어나 툴에 종속적이기 때문에 재사용에 제약을 가
- Encapsulated Components
  특정 영역에 초점을 두고 인터페이스를 통한 의미있는 서비스들 집합이며, 다양한 방법으로 구성될 수 있으며 인터페이스와 내부는 분리
- Component Frameworks
  애플리케이션을 유도하기에는 불완전한 형태이지만 그룹으로 재용가능한 항목의 모음
- Componentised Applications
  컴포넌트 형태로써 완전한 작업이 가능한 애플리케이션으로 정화 되지 않은 형태라 할지라도 컴포넌트의 특징을 가짐

2.2 소프트웨어 아키텍처

소프트웨어 공동체에서 연구의 방향을 객체지향 설계 방법(OODM : Object Oriented Design Methodology)에 둔 몇 년동안 소트웨어 아키텍처에 대한 관심 역시 확발하였다. 또한, 최근 실세계 스템에서 소프트웨어의 중요성이 증가하는 것은 소프트웨어 아키텍의 발전을 가져왔다. 하지만, 소프트웨어 아키텍처의 문제는 대형 시템일수록 커지고 있음에 직면하게 되었으며 또한, 소프트웨어 아키텍

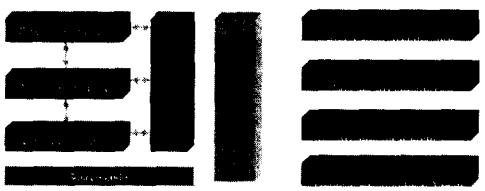
의 결정은 초기 라이프사이클에서 만들어지기 때문에 변경이 힘들다.

좋은 시스템과 시스템 아키텍처없이는 기존의 수행과 방법에 관한 요구사항은 퇴색되고 설계 변경은 거의 불가능하다. 따라서 소프트웨어 아키텍처의 재사용성을 높이기 위한 조합에 대한 연구가 요구된다.

소프트웨어 아키텍처는 시스템 컴포넌트들과 그들의 상호관계에 대한 이해를 위해 프레임워크로서 제공되며, 이러한 이해는 기존의 시스템 분석과 향후 시스템의 통합을 요구한다. 분석을 지원하기 위해 소프트웨어 아키텍처는 응용영역의 지식, 시뮬레이션과 프로토타이핑이 쉽게 획득되기를 원하며, 소프트웨어 아키텍처에 대한 정의는 다음과 같이 나타낼 수 있다.

- 컴포넌트와 그들 내부 관계성의 구조에 관한 소프트웨어 시스템의 고수준 모델[2]
- 소프트웨어 컴포넌트와 컴포넌트의 외부적으로 보이는 속성들, 그리고 그들간의 관계성을 포함하는 소프트웨어 시스템의 구조[3]

소프트웨어 아키텍처는 시스템 구조와 구조사이와 구조내에서의 관계성이 중요하며, 그러한 특징과 형태에 따라서 다음 (그림 1)과 같이 분류할 수 있다.



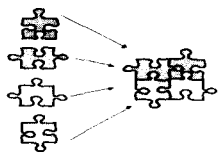
(그림 1) 관계성과 형태에 의한 소프트웨어 아키텍처의 분류

3. 컴포넌트 통합을 위한 프로세스

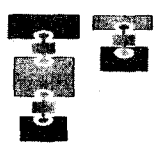
3.1 컴포넌트 통합 메커니즘

개별 기능을 지원하며, 플랫폼이나 이질적인 환경에서 상호운영성과 재사용성이 보장돼 서로 조합해서 애플리케이션을 구성할 수 있는 소프트웨어 블록으로 컴포넌트가 인식되지만, 아키텍처에 의존적이며, 실제 코딩에 의하거나 컴포넌트 개발과 통합이 함께 이루어지고 있다.

통합에 관련된 메커니즘은 직접 연결형 통합과 조합형 통합으로 크게 볼 수 있다. 직접 연결형은 제공되는 서비스와 기대되는 서비스에 대한 요구사항이 적합함으로써 어떤 중간매개체의 도움없이 상호운영성이 보장되는 형태의 통합이 가능한 경우로 다음(그림 2)과 같다.



(그림 2) 직접 연결형 통합

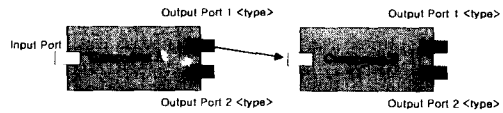


(그림 3) 조합형 통합

조합형 통합은 요구되는 서비스와 제공되는 서비스간의 불일치성이나 모호성에 근간을 두고 두 개의 컴포넌트의 인터페이스를 절충할 수 있는 어댑터 등을 통한 것과 포터나 속성에 의해 통합될 수 있는 부분을 포함하는 넓은 의미의 통합이라 할 수 있으며, 다음(그림 3)과 같다.

3.1.1 포트 기반 조합형

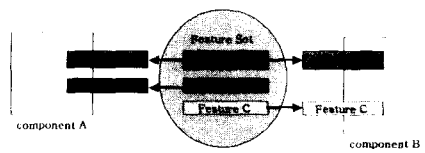
상호작용과 정보의 흐름을 사용자가 보기 쉽게 표현하기 위하여 입력 포트와 출력 포트를 가진 형태의 컴포넌트를 조합하는 형태로써, 포트에 고유의 이름과 데이터 타입을 설정하고 동일한 타입의 입력 포트와 출력 포트를 상호연결한다. 입력과 출력이 명확한 자료 처리형태에 알맞으며, 컴포넌트간의 상호작용성 표현에 유용하다.



(그림 4) 포트 기반 조립형 통합

3.1.2 속성 기반 조합형

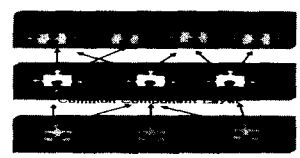
하나의 컴포넌트 속성들을 여러 구성요소로 분류하여 이들을 집합으로 구성하는 형태로 필요한 구성요소를 선택하여 조합하며, 사용자관점의 참여에 따라 사용자 인터페이스 컴포넌트 조합에 유용할 수 있다. 구성된 속성 집합은 각각의 컴포넌트의 속성에 대응하여 서비스 가능하며, 정의된 속성에 따라 유동적인 정보 흐름을 제공한다.



(그림 5) 속성 기반 조립형 통합

3.1.3 아키텍처 기반 조합형

포트 기반과 속성 기반 조합형 통합을 모두 고려하여 계층적으로 정의된 아키텍처에 속한 컴포넌트를 기능적으로 통합하는 것으로 비즈니스 컴포넌트의 통합에 이용될 수 있으며, 확장 가능하다. 통합은 특정한 도메인에서 서비스되는 컴포넌트를 선택하고 그것에 정의된 컴포넌트를 지원할 수 있는 공통적이고 기술적인 컴포넌트를 공통 컴포넌트 계층에서 선택하고 또한, 기본 컴포넌트를 지원하는 기본 컴포넌트 계층에서 선택하게 된다. 아키텍처는 다음(그림 6)과 같으며, 통합은 bottom-up 방식으로 진행되며, 요구되는 서비스에 맞는 컴포넌트는 top-down 방식으로 선택된다.



(그림 6) 아키텍처 기반 조립형 통합

3.2 컴포넌트 군 통합 메커니즘

통합된 컴포넌트는 세 가지 통합 방식에 따라 컴포넌트 군 자체로 새로운 애플리케이션으로 구성가능하며, 단순히 통합된 형태의 컴포넌트로 존재 가능하다. 구성된 컴포넌트 군을 기반으로 컴포넌트나 애플리케이션으로 구성하기 위해서는 컴포넌트 군의 특징에 따라 통합되었기 때문에 통합을 위한 연결이 다르게 적용된다. 즉, 단순히 연결의 의미만으로 통합이 가능한 반면에 어댑터를 통한 통합이나, 서비스의 입력과 출력을 분배하는 기능을 가진 개념적인 허브를 통한 통합이 필요하다. 컴포넌트 군의 통합은 상이한 컴포넌트 군사이의 간격을 줄이기 위해 어댑터에 기반한 연결이라고 할 수 있으며, 이는 각각의 변경되는 요구사항에 대해 융통성을 제공할 수 있다.

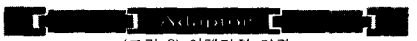
3.2.1 점대점 연결

다른 컴포넌트와 간단하게 직접적으로 연결하는 것으로 앞에서 언급한 직접 연결형 통합과 유사하다. 점대점 연결은 두 개의 컴포넌트 군을 고정하기위한 것이 필요하며, 새로운 요구사항에 대해 통합된 컴포넌트의 재통합이 요구된다.

(그림 7) 점대점 연결

3.2.2 어댑터형 연결

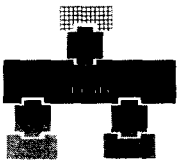
상이하게 개발된 각각의 컴포넌트에 대한 절충적인 역할을 함으로써 통합 가능한 형태를 제공하는 어댑터는 하나의 컴포넌트 군이 다른 컴포넌트 군에서 요구되는 서비스를 명세한 인터페이스를 제공할 때 서비스를 요구하는 컴포넌트와의 통신이 불명확할 경우 동일한 인터페이스를 제공하기 위해서 인터페이스나 컴포넌트 자체에 의해 정의된 프로토콜에 의해 어댑터 형태로써 서비스를 하게된다. 또한, 요구되는 서비스가 바뀔 경우에는 새로운 어댑터를 적용함으로써 통합에 대한 융통성을 제공하게 된다.



(그림 8) 어댑터형 연결

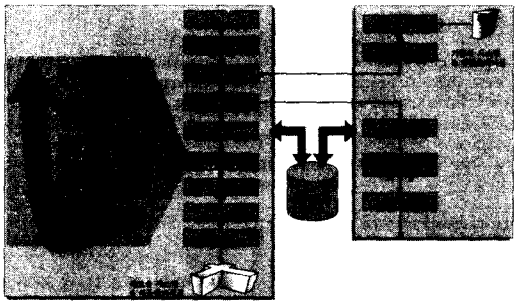
3.2.3 허브를 통한 연결

컴포넌트 군이 서비스의 입출력을 담당하는 허브를 중심으로 연결되는 형태로써, 다음(그림 9)과 같다. 허브는 컴포넌트간의 연결을 위해 타입이나, 내용의 명세, 메시지 형태와 관련하여 정의된 규칙을 포함하고 변경되는 사항에 대한 갱신을 통해 융통성을 제공할 수 있다.


(그림 9) 허브형 연결

4. 프로세스 정의

컴포넌트의 통합 메커니즘을 진재할 경우 컴포넌트의 생성이나 새로운 애플리케이션이 개발될 도메인에 대한 충분한 지식과 사전 준비과정이 수반되어야 한다. 즉, 컴포넌트를 통합할 수 있도록 최소한 통합 가능한 요소들을 아키텍처 명세와 컴포넌트 스펙에 포함하므로써, 컴포넌트를 통합하는 단계에서 좀더 체계적인 접근이 가능하다. 컴포넌트의 생성과 애플리케이션을 구성하기위한 프로세스는 병렬적으로 진행되며, 통합 로직을 포함한 전체 프로세스는 다음(그림 10)과 같이 정의 할 수 있다.


(그림 10) 컴포넌트 통합 로직을 포함한 프로세스

컴포넌트 통합에 의한 프로세스는 먼저, 새로운 사용자, 도메인, 시스템에 대한 요구사항 시작으로 도메인 분석을 하는 과정과 기존의 시스템을 컴포넌트화시키기 위한 두가지 측면을 고려할 수 있으며 그 각각에서 식별되고 추출된 아키텍처는 아키텍처의 명세과정을 거치게 된다. 식별된 아키텍처를 기반으로 필요로하는 컴포넌트를 식별하여, 기존에 이미 개발된 컴포넌트를 사용하거나 일부는 새로 개발하게된다. 선택되고 개발된 컴포넌트를 기반으로 통합 로직에 따라, 비정련화된 컴포넌트 군을 구성하게 된다.

4.1 컴포넌트 명세 분석

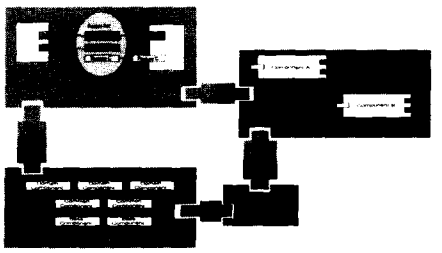
통합로직의 준비단계로써 기존의 컴포넌트나 새롭게 개발된 컴포넌트에 포함된 스펙을 바탕으로 컴포넌트를 분석하게된다. 명세의 특징상 화이트박스 형식의 분석이라기보다 컴포넌트 인터페이스와 아키텍처 구성(configuration)에 관련된 정보를 기반으로 컴포넌트의 통합가능한 형태를 파악하고 그것에 관한 정보를 통합 타입식별에 사용하게 된다.

4.2 통합 타입 식별 및 컴포넌트 군 구성

컴포넌트 명세 분석을 통한 정보에서 추출될 수 있는 컴포넌트의 통합가능한 요소에 따라 직접연결형이나 조합형 컴포넌트 통합 타입을 식별하게 된다. 조합형 컴포넌트일 경우 인터페이스를 중심으로 식별된 통합타입에 따라 포트, 속성, 아키텍처 기반 컴포넌트 군으로 각각 구성하게 된다.

4.3 통합군 식별 및 컴포넌트 통합군 재통합

통합된 컴포넌트군을 하나의 애플리케이션 컴포넌트로 구성하기위해서 컴포넌트 통합군을 재통합하게 되며, 이를 위해 컴포넌트 군을 식별하여 각각의 특징과 컴포넌트 통합 가능성에 따라 연결 타입을 정의하고, 다음(그림 11)과 같이 재통합하게 된다.


(그림 11) 컴포넌트 군의 재통합

5. 결론

소프트웨어 전반에 객체지향에서 단위기능을 수행하는 컴포넌트지향으로 패러다임이 전환되면서 컴포넌트의 수용에 대한 관심이 높아지고 있다. 이러한 컴포넌트 개발을 위해 기반되는 소프트웨어 아키텍처와 명세에 대한 연구뿐만 아니라, 컴포넌트 프로그래밍, 통합, 검증과 검사를 통한 컴포넌트 표준화 작업등에 관련하여 많은 연구가 되고 있다. 특히, 통합은 수행성과 안정성이 검증된 컴포넌트를 기반으로 안정된 시스템으로 구성할 수 있는 부분에서 중요성이 크다고 볼 수 있다.

본 논문에서는, 아직 명확하게 정의되지 않은 통합에 대한 전반적인 개념을 살펴보고, 새로운 애플리케이션을 개발하기위한 통합 메카니즘을 제시하였다. 즉, 상이하게 개발된 컴포넌트를 기능이나 식별 될 수 있는 통합요소에 따라 포트, 속성, 아키텍처 기반의 단일한 컴포넌트의 통합 방법과 통합 방식에 따라 다른 형태의 컴포넌트 군을 새로운 애플리케이션 컴포넌트로 구성하기위한 방법으로 점대점 연결, 어댑터를 통한 연결, 허브를 통한 연결 방법을 정의하고, 이에 따르는 컴포넌트 기반 프로세스와 통합 로직을 구성하였다. 이는 신뢰받을 수 있는 컴포넌트의 재사용성과 통합에 의한 소프트웨어 개발시간의 단축의 장점을 기대할 수 있다.

향후 연구로써는 컴포넌트 간의 메카니즘과 컴포넌트 군간의 통합의 사례연구를 통한 실제적인 메카니즘의 적용과 통합 가능한 요소를 아키텍처 명세언어나 컴포넌트 명세를 통해 식별하여 통합을 위한 메카니즘 적용을 자동적으로 할 수 있는 툴에 관한 연구가 수반되어야 하겠다.

【 참 고 문 헌 】

[1] CBDi Forum, "Component Development Report", Buttler Group, 1999

[2] M. Shaw and D. *Garlan, Software Architecture : Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[3] L. Bass, P. Clenents, and R. Kasman, *Software Architecture in Practice*, Addison-Wesley, 1998.

[4] Desmond F. D'Souza, Alan c. Wills, Objects, Components, and Frameworks with UML, Addison-Wesley, 1998.

[5] Ivar Jacobson, Grady Booch, James Rumbaugh, The Unified software Development Process, Addison- wesley, 1998.

[6] Clemens Szyperski, *Component Software-Beyond Object-Oriented Programming*, Addison-Wesley, 1997.