분산이슈를 고려한 컴포넌트 기반의 소프트웨어 분석 및 설계방법

정병훈[0] 염근혁
부산대학교 컴퓨터공학과
{bhchung[0], yeom}@pusan.ac.kr

Component Based Software Analysis and Design Method in Consideration of Distribution Issues

Byeong-hoon Cheong[0] Keunhyuk Yeom
Department of Computer Engineering, Pusan National University

요 약

오늘날 발달된 분산 컴퓨팅 환경 하에서는 성능, 보안, 이질성 등 시스템의 여러 분산 특성이 이슈화되며 이러한 분산 특성을 잘 고려하여 다루지 않는다면 심각한 문제를 야기시킨다. 지금까지 각 분산 이슈의 특성에 집중된 해결방법이 있었으나, 너무 구현 세부적이거나 근원적인 소프트웨어 설계방법의 관점이 아닌 경우가 많았다. 이에 반해 아키텍처 기반의 개발방법들은 소프트웨어의 여러 비기능적 특성에 근원적인 접근을 돕지만 절차가 명료하지 못한 문제를 안고 있다.

따라서 본 논문에서는 이 문제를 해결하기 위해 절차 중심의 방법을 기본으로 하여 시스템의 비기능적 특성으로의 근원적인 접근을 돕기 위해 스타일, 패턴과 같은 소프트웨어 아키텍처의 요소를 이용하는 방안을 기술한다.

1. 서 론

오늘날의 소프트웨어 시스템은 레거시 시스템과 같이 모든 구성물 또는 코드들이 똑같은 장소에서 수행되지 않으며 분산 컴퓨팅 환경 하에 여러 장소에 산재하여 서로 원거리 관계를 가지고 수행된다. 이러한 원거리 관계에서는 많은 분산 이슈가 나타나게 되며, 관련된 비기능적 특성 또한 저하된다. 대표적인 예로써 성능, 보안, 이질성 등이 있으며 원거리 관계에 의한 수행은 하나의 소프트웨어가 동일한 물리적 노드에서 수행될 때에 비해 현저한 특성 저하를 보인다.

지금까지 각각의 특성에 집중된 해결방법이 많기는 하지만 너무 구현 세부적이며 근원적인 소프트웨어 설계방법의 관점이 아닌 경우가 대부분이다. 따라서 분산 이슈에 관련된 시스템의 특성을 다루기 위해 소프트웨어 아키텍처의 측면에서 접근할 필요가 있다. 소프트웨어 아키텍처는 시스템을 구성하는 소프트웨어 컴포넌트, 그것의 외부적인 특성 그리고 그것들과의 관계에 의해 표현되는 구조이며 소프트웨어 시스템의 비기능적 특성을 근원적으로 다룰 수 있는 실마리를 제공한다.[1]

그러나 앞서 설명한 소프트웨어 아키텍처 기반의 방법들은 구현 세부적인 해결책에 비해 보다 근본적으로 소프트웨어의 비기능적 특성을 향상시키도록 도와 주지만 그 단계 또는 절차가 뚜렷하지 못한 경우가 대부분이다.

반면 Unified Software Development Process[2]와 같은 절차 중심의 방법들은 소프트웨어 개발 전체의 절차를 체계적으로 설명하고 각 단계마다 산출되어야 할 산출물을 명료하게 정의함으로써 소프트웨어 개발자들의 용이한 소프트웨어 설계를 도와주지만 소프트웨어 아키텍처를 소프트웨어의 재사용성 특성에 한정시킴으로써 설계할 소프트웨어의 여러 비기능적 특성에 제대로 접근하지 못하고 있는 것이 현실이다. 따라서 본 논문에서는 Unified Software Development Process을 보다 세부화한 "분산 컴포넌트 기반의 소프트웨어 분석 및 설계 방법"[3]을 기반으로 스타일, 패턴과 같은 소프트웨어 아키텍처의 구성요소를 적용하여 소프트웨어의 분산 이슈에 관련된 특성을 향상시키는 방안을 기술한다.

2. 관련연구

2.1 절차 중심의 소프트웨어 개발 방법

소프트웨어 개발에서 절차 및 단계를 중요하게 다루는 대표적인 개발방법으로는 Unified Software Development Process가 있다. 이것은 UML(Unified Modeling Language)을 기반으로 4개의 phase와 waterfall model의 형태를 따르는 5단계의 작업을 거치면서 각 diagram을 이용하여 5개의 모델을 형성하는 방법이다. 그러나 이 방법에서는 각 작업에 대한 구체적인 지침들이 충분히 제시되고 있지 않아서 소프트웨어 설계 시 현실적으로 도움을 받기가 쉽지 않다. 이에 "분산 컴포넌트 기반의 소프트웨어 분석 및 설계 방법"에서는 Unified Software Development Process의 각 작업에 대해 보다 세부적인 지침들을 제공함으로써 소프트웨어 설계에 도움이 되도록 하고 있으며, 또한 분산 컴포넌트 기반의 소프트웨어가 취약할 수밖에 없는 성능, 안전성,

분산 트랜잭션 관리에 대해 접근하는 방안을 제시하고 있다. 그러나 상기한 분산 특성을 설계의 마지막 단계에서 일종의 최적화 작업을 통해서 접근하는 형식을 취함으로써 설계 전반에 걸친 적용방안이 아니라는 점이 아쉬움으로 남는다.

2.2 소프트웨어 아키텍처의 이용

전술하였다시피 소프트웨어 아키텍처가 소프트웨어의 비기능적 특성을 향상시키는데 도움을 주는 것은 틀림이 없지만 그렇게 하기 위해서는 소프트웨어 아키텍처를 구성하는 요소들을 잘 이용하지 않으면 안된다. Jan Bosch는 소프트웨어 시스템의 비기능성을 향상시키기 위해 소프트웨어 아키텍처의 구성요소 중 스타일과 패턴을 이용하고 있다[4]. 그 단계는 다음과 같다.

(1) 기능성 바탕으로 아키텍처 구성
  1) System context
    · System의 interface를 정의
    · 기능적, 비기능적 요구사항을 interface에 연계
  2) Archetype
    · System을 구성하는 안정적인 기능성 unit
  3) Structure (Component, Relation)
    · System을 component로 decompose 후 관계 형성
(2) 평가 (assess)
  1) 관련 QA를 선택
  2) 각 QA에 대한 profile(set of scenario) 정의
  3) 각 QA에 대해 평가 기술 선택
(3) 변환 (Transformation)
  1) 만족되지 않는 QA 확인
  2) 각 QA에 대해 방해하는 위치 확인
  3) 가장 적절한 transformation 선택
  4) transformation 수행
※ QA(Quality Attribute:비기능적 특성)

여기서 비기능적 특성을 향상시키는 핵심은 (3) 변환(transformation)이다. 이 부분은 기능성을 바탕으로 작성된 소프트웨어의 아키텍처에 대해 평가를 마친 후 만족되지 않는 비기능적 특성을 향상시킬 것으로 판단되는 스타일이나 패턴을 이용해서 아키텍처의 변환을 시도하는 단계이다. 아키텍처를 변환함으로써 시스템의 이해도가 떨어지는 단점이 있기는 하지만 만족시키고자 하는 비기능적 특성을 향상시킬 수 있었다는 점에서 이 방법의 목적을 달성했다고 할 수 있다.

그러나 위의 방법은 커다란 형태의 절차는 제시하고 있으나 다소 세부적이지 못하고 내용의 임의적인 측면이 많아서 다른 소프트웨어 설계자가 이용하기가 쉽지 않다는 측면이 있다.

3. 소프트웨어 아키텍처 요소를 반영한 절차중심 방법

절차 중심의 방법인 Unified Software Development Process를 보다 세부화한 "분산 컴포넌트 기반의 소프트웨어 분석 및 설계 방법"의 요구사항 추출단계, 분석단계, 설계단계에서 분산 이슈에 관련된 비기능적 특성을 향상시켜줄 것으로 기대되는 소프트웨어 아키텍처의 구성요소인 스타일과 패턴의 적용방안과 더불어 부가적인 설계사안을 기술한다.

3.1 요구사항 추출단계

입력된 요구사항을 바탕으로 기능성 중심의 use-case 모델을 생성하는 단계이다. 시스템 외부의 실체를 나타내는 actor, 시스템 내부의 기능성을 나타내는 use-case, 그리고 이들간의 관계 등으로 나타내어질 수 있는 use-case 모델에서 스타일, 패턴을 적용하기 위해서는 먼저 시스템의 순수 기능성을 위주로 만들어진 use-case 모델을 필요로 한다.

순수 기능성 위주의 use-case 모델이 생성되면 만족시키고자하는 분산 이슈의 특성을 향상시킬 것으로 기대되는 스타일이나 패턴을, *기능성을 나타내는 use-case들을 재배열, 비기능성을 조절하는 기능성을 새롭게 정의 및 관계생성, 이미 패턴화되어 해결된 외부 actor를 이용* 등과 같이 적용할 수 있으며, 적용절차는 다음과 같다.

(1) 접근하고자 하는 QA 확인 및 우선순위 결정
(2) 순위 높은 QA를 만족시키는 스타일, 패턴 선정
(3) 스타일, 패턴이 해당 QA를 접근하는 방법 기술
(4) 분산에 관계하는 actor, use-case 그리고 이들간의 관계 확인
(5) 전체적인 비기능성을 조절하는 기능성 확인 및 스타일, 패턴에서의 역할 기술
(6) (5)에서 확인한 기능성을 use-case로 표현 및 관계하는 actor, use-case 재설정

예를 들어 스케줄러 패턴을 이용하면 dispatcher에 해당하는 기능성을 나타내는 use-case를 새롭게 정의하고, 스케줄러 패턴을 충족시키도록 use-case들을 배열하여 dispatch 될 수 있도록 함으로써 여러 기능성들간의 동기화 문제를 다소 해결할 수 있다. 또한 DBMS 패턴 같은 경우는 data persistence 문제에 관해 이미 해결해 놓은 DBMS를 외부 actor로 두어 쉽게 use-case 모델을 만들어 낼 수 있으며 현재 소프트웨어 설계에 대부분 암묵적으로 반영하고 있는 방식이다.

3.2 분석단계

입력된 기능성 중심의 use-case 모델을 바탕으로 분석 모델을 만들어 내는 단계이며 분석 패키지 추출, 분석 패키지간의 의존관계 형성, 분석 클래스 추출의 3가지 세부 절차를 가진다.

분석 패키지는 다루기 쉬운 조각을 의미한다. 즉 특정한 고려사항에 따라 이 분석 패키지로써 그룹화 할 수 있다는 것이다. 여기서 주의할 점은 일반적으로 분석 패키지는 구현단계의 패키지와는 달리 고려사항에 따라서 use-case 모델의 구성요소들이 여러 패키지에 걸쳐 중복된 관계를 형성할 수 있다는 것이다.

특정 분산 특성을 향상시키기 위해서 use-case를 변형했으므로 순수 기능성 위주의 use-case 모델에 비해 시스템의 이해도가 떨어지기 때문에 각 스타일이나 패턴에

따라 변형한 부분을 관리해 줄 필요가 있다. 따라서 요구사항 추출단계에서 적용했던 스타일이나 패턴을 고려사항으로 하여 분석 패키지를 추출한다. 추출방법은 스타일, 패턴의 마지막 적용단계에서 확인된 기능성 및 그와 관계하는 use-case들을 하나의 분석 패키지로 추출하면 된다. 이것은 적용했던 스타일이나 패턴이 접근하는 비기능성을 고려하고 있는 것이기도 하며 추후 향상시키고자 하는 특정 분산 특성에 대한 시스템의 요소들을 관리하기 용이하게 한다.

3.3 설계단계

입력된 분석 모델을 바탕으로 설계 모델을 만들어 내는 단계이며 분산 컴포넌트 추출, 분산 이슈를 고려한 분산 컴포넌트와 클래스 설계 단계로 나눌 수 있다.

"분산 컴포넌트 기반의 소프트웨어 분석 및 설계 방법"에서는 세 가지 분석 클래스인 boundary, control, entity 클래스를 위주로 3tier 구조의 각 layer 즉, presentation layer, business logic layer, data layer의 분산 컴포넌트로 추출하고 있다.

그러나 분산 컴포넌트의 재사용성만을 고려한 이러한 추출법은 원거리 호출을 의미하는 layer간의 호출의 빈도를 높일 가능성을 안고 있다. 따라서 분석 클래스들간의 의존도를 분석하여, 철저히 의존적이어서 분리되어서는 재사용될 수 없는 분석 클래스들은 그 layer가 다를지라도 통합하여 추출하는 것이 원거리 호출의 빈도가 높아지는 것을 막을 수 있다. 한 예로 DBMS에서는 stored procedure라는 것을 제공하고 있는데 data layer에 속할 DB에게 한정적으로 procedure을 정의할 수 있게 하고 있다. 이로써 의존적으로 자주 호출되는 procedure를 business logic layer에서 굳이 구성할 필요가 없게 하여 layer간 원거리 호출의 빈도를 줄일 수 있다.

분산 이슈를 고려한 분산 컴포넌트와 클래스 설계 시 특정 분산 이슈에 대한 특성을 향상시키기 위해서는 class의 organization을 돕는 디자인 패턴을 이용할 필요가 있다. 그러나 이 디자인 패턴을 이용할 때 어려운 점이 있는데 그것은 바로 디자인 패턴을 적용하는 시기를 결정하는 것이다.[5] 이것은 디자인 패턴의 일반성에 기인하며 이 문제를 해결하기 위해서는 자신의 관심사(분산이슈)에 대해서 보다 구체적인 부가기술을 함으로써 패턴의 사용을 도울 수밖에 없다.

디자인 패턴은 일반적으로는 분산 이슈의 특성을 기준으로 기술되어 있지는 않다. 따라서 디자인 패턴을 분산 이슈의 측면으로 바라보아 재정리할 필요가 있다. 패턴의 의도, 패턴의 참가자 등의 구성요소를 분산이슈에 관해 정리하고 분산 환경에서 적용시의 효과를 접근될 수 있는 특성별로 정리해 두어야 한다. 보통 디자인 패턴은 특정 특성을 향상시키는 반면 한편으로는 생각하지 못한 특성을 감소시키는 경향이 있기 때문이다. 이렇게 정리한 디자인 패턴의 적용효과를 보고 자신이 관심을 가지고 있는 분산 특성들과 그 외의 특성들간의 타협점을 찾은 후 디자인 패턴의 적용에 들어가야 한다. 예를 들어 Abstract factory[6] 디자인 패턴은 적용 시 특정 플랫폼이 아닌 여러 플랫폼에 대한 수용방법을 제공함으로써 heterogeneity 분산 특성을 향상시킬 수 있지만 각 플랫폼의 최적화 솔루션을 배제할 수밖에 없으므로 성능적인 측면에서 손해를 볼 수 있다는 것을 염두에 두어야 한다.

마지막으로 실행결과를 최적으로 이끌어 낼 수 있도록 수행노드 결정을 지원하는 설계에 관해 언급한다. 현재 설계는 특정한 쪽(서버 혹은 클라이언트)에서만 고정적으로 수행이 되는데 예를 들면, 구현 인프라로서 asp, jsp 같은 스크립트는 서버 쪽에서 고정적으로, 자바 애플릿은 클라이언트 쪽에서 고정적으로 수행된다. 그러나 이러한 설계는 network traffic, node load 등과 같은 환경 변화 시 최적 수행결과를 이끌어 내기 힘들다. 예를 들어 서버 쪽 load가 극심한데도 스크립트 기반의 소프트웨어가 서버 쪽에서 수행되는 것은 비효율적이며, network traffic이 아주 심각할 때 코드 다운로드 비용이 높은 자바 애플릿으로 클라이언트 쪽에서 수행하려고 하는 것 또한 역시 비효율적이기는 마찬가지이다. 따라서 분산 환경의 변수를 읽어내어 수행이 능동적으로 일어나도록 설계하는 것도 소프트웨어의 분산 특성을 향상시킬 수 있는 방법이 될 수 있다.

4. 향후 연구방향

지금까지 "분산 컴포넌트 기반의 소프트웨어 분석 및 설계 방법"을 기반으로 아키텍처의 요소를 이용하여 분산 특성을 향상시키는 것과 그 외 몇 가지에 대해 기술하였는데 소프트웨어의 분산 특성을 향상시키는 방법에 아키텍처의 요소 즉 스타일, 패턴의 이용만이 존재하는 것은 아닐 것이다. 향후 연구 방향으로서 현재 제시한 방법들의 실제 구현 사례를 만들고 스타일, 패턴 이외의 소프트웨어 아키텍처로써 소프트웨어의 분산 특성을 향상시킬 수 있는 방안을 다룰 것이다.

[참고문헌]

[1]Len Bass, Paul Clements, Rick Kazman, "Software Architecture in Practice", Addison Wesley, 1998

[2]Ivar Jacobson, Grady Booch, James Rumbaugh, "The Unified Software Development Process", Addison Wesley, 1998

[3]최유희, "분산 컴포넌트 기반의 소프트웨어 분석 및 설계 방법", 부산대학교, 2001

[4]Jan Bosch, "Design & Use of Software Architectures", Addison Wesley, 2000

[5]Alan Shalloway, James R. Trott, "Design Patterns Explained: A New Perspective on Object-Oriented Design", Addison Wesley, 2002

[6]Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison Wesley, 1995