오픈 소스 기반 ESB 의 패턴 적용 메시지 모니터링 시스템 구축 사례

이성재*,
*포스코 ICT 기술연구소 SW 융합기술팀
e-mail : phantom@poscoict.com

A Study on Design Pattern Applied Message Monitoring System for Open Source Software based ESB (Enterprise Service Bus)

Sung-Jae Lee
* POSCO ICT Technology Research Laboratory SW Convergence Technology Team

요 약

오픈 소스 기반 ESB 모니터링 시스템 설계에 EIP(Enterprise Integration Pattern)을 적용하여 비즈니스 요구사항을 처리하는 통합 로직과 메시지 추적 관리를 위한 관리 로직을 분리하여, 메시지 관리 로직 처리에 따른 성능 저하 및 오류에 의한 영향도를 최소화하며 메시지 데이터를 중앙 저장 장치에 저장하여 메시지 추적 관리를 용이한 모니터링 시스템 구축 사례

1. 서론

최근 기업들은 EAI 를 기반으로 한 시스템 통합의 문제를 극복하고, 보다 효과적으로 기업 환경변화에 대응하기 위해 기업 내 표준 ESB (Enterprise Service Bus)[1] 구축에 많은 노력을 기울이고 있다.

이것은 ESB 가 기존 EAI 의 중앙 집중 형 연결방식에서 벗어난 버스구조의 시스템 연동 솔루션으로, 다양한 표준 프로토콜을 기반으로 재 사용 가능한 컴포넌트들을 조립함으로써 유동적인 기업 IT 환경구축을 가능하게 하고[7-9]. 급격하게 변화하는 경영환경변화에 유기적으로 적응할 수 있도록 지원하기 때문이다.

이러한 이유로 오픈 소스 소프트웨어 진영에서도 ESB 개발이 활발하게 이루어 지고 있다. 현재 오픈 소스 소프트웨어 ESB 의 소프트웨어 성숙도는 상용제품과 비교해도 기능, 성능 및 안정성 측면에서 큰 차이가 없어 비즈니스 환경에 적용 가능한 수준이다.[10] 그러나 이러한 오프 소스 소프트웨어 기반의 ESB 를 구축할 경우 ESB 상의 모든 IT 기능들은 개별 컴포넌트로 정의되며, 채널을 통해 메시지를 교환함으로써 유기적으로 연동된다[9]. 따라서 이러한 메시지의 안정적인 송수신은 곧 ESB 의 일관성 있는 메시지 연계 보장을 위한 핵심 요건이라 할 수 있다. 하지만, 현 재까지의 오픈 소스 기반 EBS 는 시스템 또는 서비스 장애 시 발생할 수 있는 메시지의 손실 및 오류를 추적 하고 복구하기 위한 체계적인 모니터링 서비스에 대한 기능이 미흡하여, 장애 발생 시에 효과적으로 메시지 송수신을 추적하고 오류를 탐지 하기가 어렵다.

이에 본 논문에서는 이러한 오픈 소스 소프트웨어 기반의 ESB 의 메시지 송수신을 효과적으로 추적하고 오류를 탐지하기 위한 메시지 모니터링 시스템을 디자인 패턴을 적용하여 구축 하였다

디자인 패턴은 특정 영역의 설계의 문제를 해결하기 위해 고안된 형식적인 방법이다. 이 방식은 건축학 영역에서 고안된 것을 그 시초로 하며, 이후 컴퓨터 과학 및 소프트웨어 등의 분야에서 받아들여지게 되었다.[2]

따라서 즉 본 논문에서 제안하는 메시지 모니터링 서 시스템은 오픈 소스 소프트웨어 기반 ESB 상에서 다양한 서비스들이 송수신하는 메시지를 효과적으로 모니터링 및 추적하고, 신속한 오류 탐지 및 복구가 가능할 것으로 판단된다.

2. 관련연구

2.1 엔터프라이즈 서비스 버스

엔터프라이즈 서비스 버스 (ESB)는 서비스 지향 아키텍처에서 소프트웨어 응용 프로그램을 상호 작용과 커뮤니케이션을 위한 설계 및 구현에 사용되는 소프트웨어 아키텍처 모델이다. 분산 컴퓨팅을 위한 소프트웨어 아키텍처 모델로서 그것은 좀 더 일반적인 서버/클라이언트 소프트웨어 아키텍처 모델의 변형이다, 애플리케이션 간의 의사 소통 및 상호 작용에 대해 엄격한 비 동기 메시지 지향 설계를 추구하고 있다. 주요 용도는 이 기종 복잡한 기업 어플리케이션 통합에 있다[1].

본 연구에서 사용하는 JbossESB 는 오픈 소스 J2EE 기반의 서비스 지향 아키텍처 (SOA) 지원하는 ESB 이다. IT 자원, 데이터, 서비스 및 애플리케이션을

연결, 서비스를 통합하는 비즈니스 이벤트를 처리하고, 비즈니스 프로세스를 자동화를 가능하게 하며, 자바 기반이기 때문에, 크로스 플랫폼을 지원하며 자바를 지원하는 모든 운영 체제에서 사용 가능하다.

2.2 모니터링에 관한 연구

시스템을 구축함에 있어 시스템에 대한 가시성을 확보하고 오류와 문제를 탐색하여 신뢰성과 안정성, 효율성을 높이기 위한 일환으로 모니터링에 관한 연구는 끊임없이 진행되어 왔다. 특히 이러한 모니터링에 관한 연구들은 주로 시스템의 성능측정 및 부하분배라는 효율성 향상에 그 초점을 맞추고 있다. 즉, 시스템 성능에 대한 가시성확보와 성능 향상을 위한 정보수집에 초점을 맞추고 있는 연구들로써 ESB 내의 메시지의 변환 과정을 추적 관리하기 위한 모니터링 기법과는 차이가 있다. 이에 본 연구에서는 ESB 상의 효과적인 메시지 추적 관리라는 관점 아래 오픈 소스 소프트웨어인 JBossESB 를 대상으로 효과적인 모니터링 시스템을 EIP(Enterprise Integration Pattern)을 적용하여 구축한 사례에 대해 기술 한다.

2.3 기존 ESB의 메시지 모니터링 기술

ESB 는 일종의 분산 서비스 협업 모델인 SOA 에서 다양한 서비스들의 협업을 위한 인프라스트럭처로 사용되고 있다. 따라서 대부분의 연구들은 ESB 자체에 대한 모니터링 연구보다는 이를 포괄적으로 수용하는 SOA 환경에서 서비스들을 모니터링하고 관리하기 위한 기법들을 다루고 있다.

ESB 에서 모니터링에 활용되는 기술에는 크게 JMX(Java Management Extensions)과 로깅이 있다.

JMX를 이용한 모니터링은 MBean 인터페이스를 이용하여 컴포넌트 및 서비스의 목록과 상태를 관리하나 Managed Bean 인터페이스를 통해 관리 가능한 범위가 ESB 내부의 컴포넌트와 서비스의 상태정보로 제한 되어 있어 메시지를 모니터링 하고 추적하기 기능이 미흡하고 로깅(Logging)을 이용한 모니터링은 주로 로거(Logger)가 메시지 교환 시에 발생되는 정보를 텍스트파일에 기록하는 방법이다. 하지만 순수 텍스트 파일로 저장되어 있어 메시지 추적 관리 등의 목적으로 활용하기 위해서는 별도의 작업이 필요하다. 일반적인 메시지의 송수신이나 ESB 상태 정보를 기록하기 때문에 효과적인 메시지 추적을 위한 정보로 활용되기에 부족하다.

이처럼 JMX, 로깅을 이용한 기존의 모니터링 서비스 기술들은 메시지의 상태변화에서 송수신에 이르는 전 과정을 체계적으로 모니터링 및 관리하고, 이를 효과적으로 활용하는데 있어 다소 미흡한 점이 있다. 따라서 본 연구에서는 메시지 관련 정보의 수집 및 모니터링에서부터 이를 체계적으로 저장 및 관리하고, 활용하기 위해 EIP를 적용하여 효과적인 메시지 모니터링 시스템 구축한 사례를 소개 한다.

3. ESB 메시지 모니터링 시스템

3.1 시스템 요구사항 정의

ESB 에서 생성되는 메시지의 모든 상태변화와 송수신 과정을 모니터링 및 추적해야 하며, 이 기본 요구사항에서 상세 시스템 요구사항은 아래와 같다.

- 서비스 상태 정보 파악이 용이해야 한다.
- 메시지 변환과정 추적이 가능해야 한다.
- 메시지 송/수신 이력이 알 수 있어야 한다
- 메시지 추적정보의 과거 이력을 볼 수 있어야 한다.
- 추적정보 활용이 용이해야 한다.
- 시스템 확장 성을 보장해야 한다.
- 적용 편의성을 제공해야 한다.
- 실제 메시지 연계 기능의 성능저하 및 이상 발생 시의 영향도 최소화 해야 한다.

3.2 적용 패턴 및 설계

패턴은 시스템 디자인 시에 자주 발생하는 문제들에 대한 재상용 가능한 해결책으로써, 이미 증명된 과거의 경험으로부터 나온 공통적인 솔루션에 대한 표현이며 코드 재사용 문제에 대한 해결책으로 자리매김하고 있다[6].

- 와이어 탭 패턴(Wire Tap Pattern)

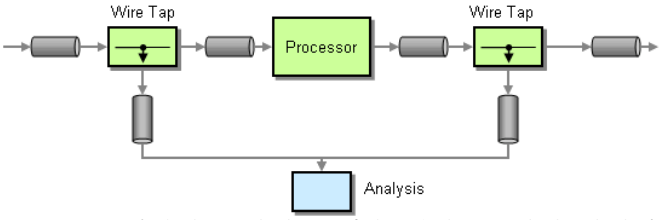
첫 번째는 포인트 투 포인트 채널에 메시지 추적 관리를 수행하는 수신 로직을 리스너로 등록하는 방법이나 메시지 수집을 위해 추가한 리스너가 메시지를 소비하여 원래 메시지를 소비해야 하는 애플리케이션이 정상적으로 비즈니스 로직을 수행 할 수 없어 적용이 불가능 하다.

두 번째는 송/수신 측에서 메시지 송수신 이벤트가 발생 할 때 마다 메시지 추적 관리를 위한 채널에 메시지를 보내는 방법은 잠재적인 많은 컴포넌트의 변경을 강요한다. 더구나 패키지 애플리케이션의 경우 애플리케이션의 수정이 불가능하여 적용이 용이하지 않다.

세 번째는 퍼블리쉬-서브스크라이브 채널로 변경하여 메시지 흐름에 방해 없이 메시지 추적 관리를 위한 메시지를 수집하는 방법이다. 그러나 퍼블리쉬-서브스크라이브 채널 문법의 변경으로 특정 메시지에 대해 하나의 비즈니스 로직이 수행되어야 하나 여러 개의 리스너가 연계되어 중복 될 수 있는 여지가 있으며 하나의 비즈니스 로직이 수행 되도록 구성되어도 퍼블리쉬-서브스크라이브 메시지 송/수신 메커니즘은 포인트 투 포인트 로직에 비해 효율성과 안정성이 떨어지는 문제가 있다[3].

이에 EIP(Enterprise Integration Pattern) 중의 하나인 와이어 탭 패턴은 포인트 투 포인트 채널을 경유하는 메시지를 디버깅과 저장을 목적으로 메시지를 추적 관리 할 수 있는 유용한 설계 방식을 적용하였다.와이어 탭은 입력 채널과 출력채널, 그리고 메

시지 추적 관리를 위한 채널로 구성되며 와이어 탭 컴포넌트는 입력 채널로 들어온 메시지를 메인 흐름인 출력 채널로 변경 없이 전달하며 수신 리스트에 설정되어 있는 메시지 추적 관리 채널에 복사본을 전달한다.



[그림-1] 메시지 런타임 분석을 위한 두 쌍의 와이어 탭 사용

・Message Store 패턴

・

비 동기 메시징 시스템은 메시지 전송을 보장한다. 그러나 언제 메시지가 전달되는 지는 보장하지 않는다. 실제 비즈니스 환경에서는 애플리케이션들의 시스템 응답 시간이 중요하다. 그리고 정해진 시간 간격 내에 얼마나 많은 메시지들이 전송되는지 아는 것은 매우 중요하다.

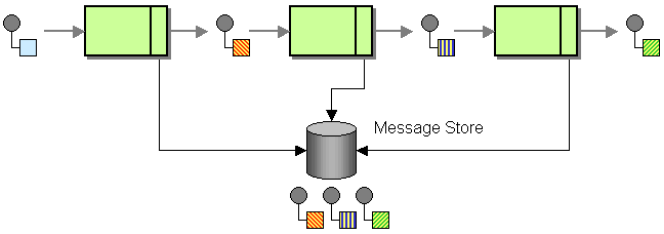
메시지 이력은 메시지 변화의 모습을 설명하는 유용한 패턴이다. 이 데이터로부터 모니터링 하고자 하는 메시지의 처리량과 실시간 통계를 집계 할 수 있다. 유일한 단점은 이러한 데이터가 각각의 개별적인 메시지에 저장되어 있다는 것이다. 이렇게 여러 메시지에 분산되어 있는 정보를 수집하여 보고하는 것은 쉬운 일이 아니다. 그리고 메시지의 생명 주기가 매우 짧아 메시지가 소비되면 더 이상 메시지 이력 정보는 사용 할 수가 없다.

이러한 이유로 의미 있는 정보의 보고를 위해서는 메시지 데이터를 중앙 장치에 저장하는 것이 필요하며, 각각의 메시지에 저장되어 있는 정보를 중앙 위치에 저장하기 위해 메시지 스토어 패턴을 사용한다. 메시지 스토어 패턴을 사용 할 때, 메시지 시스템을 인프라스트럭처로 사용하여 비 동기 방식을 사용 할 수 있다. 메시지를 채널에 보낼 때 메시지를 복사하여 메시지 스토어가 수집하는 특별한 채널에 보낸다. 이러한 작업은 컴포넌트에 의해 수행되거나 와이어 탭을 채널에 삽입하여 수행 할 수 있다.

두 번 째 메시지를 파이어 앤 포켓 모드로 보내는 것은 메인 애플리케이션 메시지 흐름에 성능 저하 등의 영향 도를 최소화 할 수 있다. 그러나 네트워크 트래픽을 야기 할 수 있으므로 모든 메시지를 저장하지 않고 메시지 아이디, 채널 및 타임스탬프 같은 분석에 필요한 최소한의 중요 키 필드만 저장하는 것을 권장한다.

중앙 저장 장치에 저장하는 방식은 중요한 고려사항이다. 많은 데이터를 저장 할수록 보고 능력은 향상되나 네트워크 트래픽과 메시지 스토어 스토리지 크기는 커지는 문제점이 있으면 심지어 모든 메시지 정보를 저장하더라도 메시지 헤더의 구조는 동일하지만 메시지 내용은 다른 구조를 가지고 있어 보고 능력에는 한계를 가지게 된다. 메시지 유형에 따라 메시지 정보 형태가 다르므로 타입에 따른 다른 저장 옵션을 고려해야 한다. 만약 각각에 메시지 타입에 따라 다른 스토리지 스키마를(테이블) 생성한다면, 인텍스를 적용하고 복잡한 검색 로직을 수행해야 한다. 그러나 이러한 방식은 각각의 메시지 타입에 따라 스토리지 구조를 가지고 있어야 함으로 유지보수 시 부하가 된다. 또한 메시지 스토어가 커지면 소거 메커니즘을 적용하여 오래된 메시지 로그를 데이터베이스에 백업하거나 삭제 할 수 있도록 해야 한다[3].

이에 와이어 탭을 통하여 ESB 를 경유하는 모든 입/출력 메시지의 내용을 메시징 시스템 통하여 전달받아 중앙 저장 장치에 저장하도록 메시지 스토어 패턴을 저장 하였다. 그리고 이 메시지 추적 관리를 위한 데이터는 웹 애플리케이션 형태의 사용자 인터페이스를 통해 추적 관리 및 통계 데이터를 모니터링 할 수 있도록 하였다.



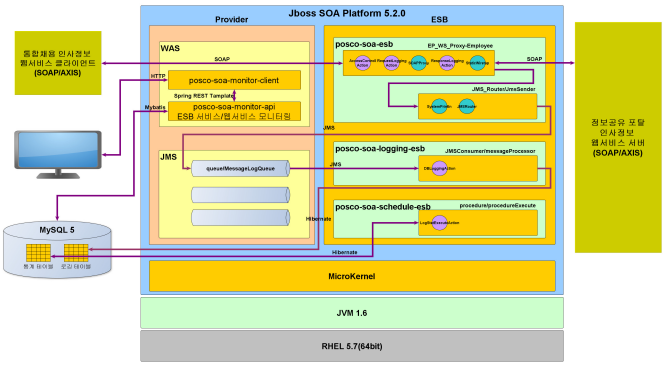
[그림-2] 메시지 추적 관리를 위한 중앙저장 장치의 이력 저장

3.3 프로토타입 구현 및 동작

SOAP (Simple Object Access Protocol) 기반의 웹 서비스 연계와 모니터링을 위하여 JbossESB 를 기반으로 메디에이션 레이어를 구축하였고 ESB 로 연계되는 메시지의 추적 관리를 위한 모니터링 시스템은 앞서 설명한 EIP 패턴 중 하나인 와이어 탭 패턴을 적용하여 메시지를 메시지 추적 관리를 위한 채널에 전송하도록 하였고, 메시지 스토어 패턴을 적용하여 메시지를 중앙 저장 장치인 데이터베이스에 저장 하도록 하였다.

와이어 탭은 JbossESB 의 수신자 리스트 기반의 StaticWireTapAction 과 JMSRouter 를 통해 메시징 시스템으로 구성된 채널에 복사본을 전송하도록 하였다. 메시징 시스템은 자바 기반의 메시지 시스템 표준 스펙인 JMS(Java Messaging System)의 구현체인 Jboss Messaging 을 사용하여 다른 자바 기반의 메시지 시스템으로 교체가 가능하도록 하였다.

메시지 스토어에 저장하는 로직은 JBossESB 상에서 JMS 리스너를 통해 메시지를 수신 받아 Hibernate 통해 메시지의 ESB 내의 처리 시간과 정상 처리 여부를 저장 하도록 하였다. 중앙 저장 장치는 MySQL 을 사용하여 메시지 추적 관리를 위한 데이터의 영속성을 보장하도록 하였다.



[그림-3]모니터링 시스템 컴포넌트 구성도 및 메시지 흐름도

메시지 추적 관리를 위한 사용자 인터페이스는 스프링 프레임웍 기반의 REST (REpresentational State Transfer) 템플릿 기반으로 다양한 애플리케이션에서 메시지 추적 관리를 위한 모니터링이 가능하도록 하였다. 그리고 웹 인터페이스는 자바 스크립트 기반의 리치 클라이언트 라이브러리인 dhtmlx 를 사용하여 구현하였다.

[그림-4] 웹 인터페이스 모니터링 화면

4. 결론 및 향후 연구

최근 기업들은 EAI 를 기반으로 한 시스템 통합의 문제를 극복하고, 보다 효과적으로 기업 환경변화에 대응하기 위해 기업 내 표준 ESB (Enterprise Service Bus)[1] 구축에 많은 노력을 기울이고 있으며 ESB 가 기존 EAI 의 중앙 집중 형 연결방식에서 벗어난 버스구조의 시스템 연동 솔루션으로, 다양한 표준 프로토콜을 기반으로 재 사용 가능한 컴포넌트들을 조립함으로써 유동적인 기업 IT 환경구축을 가능하게 하고[7-9]. 급격하게 변화하는 경영환경변화에 유기적으로 적응할 수 있도록 지원하기 때문이다.

이러한 이유로 오픈 소스 소프트웨어 진영에서도 ESB 개발이 활발하게 이루어 지고 있으며 ESB 상의 모든 IT 기능들은 개별 컴포넌트로 정의되며, 채널을 통해 메시지를 교환함으로써 유기적으로 연동된다. 따라서 이러한 메시지의 안정적인 송수신은 곧 ESB 의 일관성 있는 메시지 연계 보장을 위한 핵심 요건이다. 하지만 현재까지의 오픈 소스 기반 EBS 는 시스템 또는 서비스 장애 시 발생할 수 있는 메시지의 손실 및 오류를 추적 하고 복구하기 위한 체계적인 모니터링 서비스에 대한 기능이 미흡하여, 장애 발생 시에 효과적으로 메시지 송수신을 추적하고 오류를 탐지 하기가 어렵다.

이에 본 논문에서는 오픈 소스 소프트웨어 기반의 ESB 의 메시지 송/수신을 효과적으로 추적하고 오류를 탐지하기 위한 시스템인 메시지 추적 관리 모니터링 시스템을 EIP 인 와이어 탭 패턴과 메시지 스토어 패턴을 설계에 적용하여 JbossESB 제공하는 컴포넌트를 중심으로 구현 하였다

모니터링 시스템 설계에 패턴을 적용하여 비즈니스 요구사항을 만족시키지 위한 메인 메시지 흐름에 성능과 메시지 추적관리 로직 처리의 오류에 따른 영향도를 최소화 할 수 있었으며 메시지 스토어 패턴을 적용하여 메시지의 이력 관리를 중앙 저장 장치를 통해 웹 인터페이스 기반으로 모니터링이 용이한 시스템 구축이 가능하였다. 그러나 오픈 소스 소프트웨어인 JBossESB 를 기반으로, JBossESB 의존적인 컴포넌트를 활용하여 구현하여 다른 ESB 모니터링 시스템으로 활용에 제약이 있다. JBossESB 의존적인 컴포넌트를 보다 범용적으로 사용할 수 있는 일반기술로 개선 및 프레임웍 형태로 개발하여 다른 유사한 오픈 소스 소프트웨어 ESB 모니터링 시스템 구축 시에도 최소한의 소스레벨 개발을 통해 구현 가능 하도록 할 예정이다.

참고문헌

[1] David A., Chappell, Enterprise Service Bus, O'Reilly Media, 2004.

[2] Erich gamma, Richard Helm, Ralph Johnson, and John, Vlissides, Design Pattern,Addison-Wesley Pub.Co., 1994.

[3] Gregor Hohpe, Bobby Woolf , Enterprise Integration Patterns, Addison-Wesley Pub.Co., 2003.

[4] Thomas Earl, Service-Oriented Architecture : Concepts, Technology, and Design, Prentice Hall PTR, 2005.

[5] JBoss_Enterprise_SOA_Platform-5-ESB_Programmers_Guide-en-US For JBoss Developers by Red Hat, 2011.

[6] Robert L. Glass, Facts and Fallacies of Software Engineering, Addison-Wesley, 2003.

[7] Schmidt, M. T., B. Hutchison, P. Lambros, R. Phippen, "The Enterprise Service Bus : Making service-oriented architecture real" ,IBM Systems Journal, Vol.44, No.4(2005),pp.781-797.

[8] Papazoglou, M. P. and W-J. van den Heuvel, "Service-Oriented Architectures : Approaches, Technologies and Research Issues" ,The VLDB Journal, Vol.16, No.3(2007), pp.389-415.

[9] Naveen Erasala, David C. Yen, and T. M.Rajkumar, "Enterprise Application Integration in the electronic commerce world" ,Computer Standards and Interfaces, Vol.25 (2003), pp.69-82.

[10] TIJS RADEMAKERS JOS DIRKSEN,Open-Source Esbs in Action, 2009