효율적인 서비스 모니터링을 위한 이벤트 주도 동적 모니터

(An Event-Driven Dynamic Monitor for Efficient Service Monitoring)

금 득 규 [†]　　김 수 동 [††]
(Deuk Kyu Kum)　(Soo Dong Kim)

요 약 서비스 지향 아키텍처(Service Oriented Architecture, SOA)에서의 서비스는 서비스 소비자에게 대부분 블랙 박스 형태로 인식되고, 동적으로 실시간에 진화될 수 있으며, 다수의 인지되지 않은 이질적인 환경에서 실행된다. 이러한 SOA의 특성으로 인해 동적으로 서비스의 다양한 측면을 효과적, 효율적으로 모니터링하는 것은 필수 핵심 기능이다. 하지만, 이와 관련된 현재까지의 연구나 솔루션들은 실질적으로 서비스 자체에 초점을 맞추어 비즈니스 프로세스상의 영향 요인은 간과되는 측면이 있으며, BPEL엔진이나 미들웨어의 API에 의존한 외부 모니터링 데이터만의 획득으로 비즈니스 수준의 유용한 정보를 제공하는 데 부족한 면이 있다. 또한, 서비스 품질을 저하시킬 수 있는 모니터링으로 인한 과부하를 줄일 수 있는 효율적인 방법에 대한 연구 역시 부족하다. 이벤트 주도 아키텍처(Event Driven Architecture, EDA)는 발생하는 이벤트들을 효율적으로 수집하고 분석하기 위해 SOA를 보완하는 역할을 할 수 있다. 본 논문에서는 모니터링 측면에서의 EDA 장점들을 도출하고, 모니터링 대상을 분류하여 각 대상에 적합한 효율적인 모니터링 기법을 제시한다. 또한, 그것을 더 적용성 있도록 하기 위하여 이벤트 메타 모델을 정의하고, 이를 기반한 이벤트 처리 모델과 아키텍처를 제안한다. 제안하는 아키텍처와 기법을 사용하여 실행시간에 외부 모니터링 데이터뿐만 아니라 내부 모니터링 데이터를 효율적으로 수집 및 처리할 수 있는 이벤트 주도 동적 모니터링 프레임워크의 프로토타입을 구현하고, 사례연구를 통하여 본 연구의 실효성과 적용 가능성을 보여준다.

키워드 : 서비스 지향 아키텍처, 이벤트 주도 아키텍처, 이벤트 모델, 서비스 모니터링

Abstract Services in SOA are typically perceived as black-box to service consumers, and can be dynamically evolved at runtime, and run on a number of unknown and heterogeneous environments. Because of these characteristics of the services, effective and efficient monitoring of various aspects on services is an essential functionality for autonomous management of service. But the problem with or limitation in conventional or existing approaches is, that they focus on services themselves, ignoring the effects by business processes. Consequently, there is a room for service monitoring which provides more useful information of business level by acquisition of only external monitoring data that depend on specific BPEL engine and middleware. Moreover, there is a strong demand to present effective methods to reduce monitoring overhead which can degrade quality of services. EDA can cope with such limitations in SOA by collecting and analyzing events efficiently. In this paper, we first describe EDA benefits in service monitoring, and classify monitorring target, and present an appropriate monitoring method for each monitoring target. Also to provide the applicability of our approach, an

· 본 논문은 정보통신산업진흥원의 SW공학 요소기술 연구개발사업의 결과물임 (2010년도)

† 종신회원 : 숭실대학교 컴퓨터학과
dkkum73@gmail.com
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim777@gmail.com

논문접수 : 2010년 8월 18일
심사완료 : 2010년 10월 6일

Copyright©2010 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.

정보과학회논문지: 소프트웨어 및 응용 제37권 제12호(2010.12)

event meta-model is defined, and event processing model and architecture based on the meta-model are proposed. And, with the proposed architecture and method, we implement a prototype of an event-driven dynamic monitoring framework which can collect and process internal and external data at runtime. Finally, we present the result of a case study to demonstrate the effectiveness and applicability of the proposed approach.

Key words : Service-oriented Architecture, Event Driven Architecture, Event Model, Service Monitoring

1. 서 론

최근의 경영 환경에서 기업들은 각종 위험과 불확실성 등 상시적 변화에 항상 노출되어 있다. 경영환경의 변화는 기업의 유연하고 민첩한 대응을 요구하게 되었고, 이러한 상황에서 SOA와 EDA가 이러한 요구 사항을 충족시키기 위한 소프트웨어 아키텍처로 주목을 받고 있다[1].

SOA에서의 서비스는 서비스 소비자에게 대부분 블랙박스 형태로 인식되며, 많은 수의 대체 가능하고 실행시에 발견할 수 있는 컴포넌트들의 조합으로 이루어진다. 또한, 동적으로 실시간에 진화될 수 있으며, 다수의 인지되지 않은 이질적인 하드웨어, 소프트웨어 플랫폼상에서 실행된다. 이러한 SOA의 특성으로 인해 동적으로 서비스의 다양한 측면을 효과적, 효율적으로 모니터링하는 것은 필수 핵심 기능이다. 그리고 이런 서비스를 모니터링할 때, 여러 서비스뿐만 아니라 SOA의 다양한 애플리케이션들과 컴포넌트들로부터 발생하는 이벤트들을 효율적으로 수집하기 위해 EDA는 SOA를 보완하는 역할을 할 수 있다. 또한, EDA는 기업의 내, 외부 이벤트들을 모니터링 하고 응답하는 전역 상황을 다루는데 필요하다[1]. 하지만, 효과적인 서비스 모니터링을 위해 EDA를 적용하거나 특화시킨 연구는 많이 진행되지 않았다. 이와 관련된 현재까지의 연구나 솔루션들은 실질적으로 서비스 자체에 초점을 맞추어 비즈니스 프로세스상의 영향 요인은 간과되는 측면이 있으며[2], BPEL 엔진이나 미들웨어의 API에 의존한 외부 모니터링 데이터만의 획득으로 비즈니스 수준의 유용한 정보를 제공하는 데 부족한 면이 있다. 또한, 서비스 품질을 저하시킬 수 있는 모니터링으로 인한 과부하를 줄일 수 있는 효율적인 방법에 대한 연구 역시 부족하다.

본 논문에서는 모니터링 측면에서의 SOA와 EDA의 장, 단점들을 도출하여 효율적인 모니터링을 위한 근거를 제시하고, 모니터링 대상을 분류하여 각 대상에 적합한 효율적인 상호작용 모델과 모니터링 기법을 제안한다. 또한, 그것을 더 적용성있도록 하기 위하여 이벤트 메타 모델을 정의하고, 이를 기반한 이벤트 처리 모델과 아키텍처를 제안한다. 제안한 이벤트 메타모델은 OASIS Web Services Distributed Management Event Format (WEF)[3] 표준을 기반으로 정의되었으며, 쉽게 확장할 수 있는 매커니즘을 포함하여 타 모니터링 프레임워크에서 쉽게 확장 및 사용이 가능하다. 제안하는 아키텍처와 기법을 사용하여 실행시간에 외부 모니터링 데이터뿐만 아니라 내부 모니터링 데이터를 효과적으로 수집 및 처리할 수 있는 이벤트 주도 동적 모니터링 프레임워크를 설계하고 제안하며, 효율성을 측정할 수 있는 메트릭을 정의하고, 실험을 통하여 효율성을 입증한다. 또한, 포로토타입 구현을 통하여 사례연구를 수행한다.

2장에서는 관련연구를 서술하며, 3장에서는 모니터링 대상을 분류하고, EDA의 장점들을 채용한 이벤트 주도 서비스 모니터링의 구성요소와 기술, 폴링 및 푸쉬 상호작용 모델의 적용 기법, 그리고 이벤트 메타모델을 설명하고, 4장에서는 모니터링 프레임워크의 설계 기준과 이를 준수하는 아키텍처를 설계하고 이벤트 처리모델과 함께 자세히 설명한다. 5장에서는 이벤트 주도 서비스 모니터링의 효율성을 평가할 메트릭을 정의하고, 6장은 사례연구를 기술하고, 마지막 7장은 평가 및 결론을 제시한다.

2. 관련연구

McGregor의 연구에서는 비지니스 프로세스 성능 측정을 위한 솔루션 매니저 서비스(Solution Manager Service, SMS) 아키텍처를 제안하였다[4]. 제안된 아키텍처는 웹서비스 로그를 지원하며 이벤트 프로세싱 컨테이너(Event Processing Container)를 통하여 실시간에 많은 수의 프로세스 이벤트들을 처리할 수 있도록 하였다. 제안된 아키텍처의 장점은 수행된 비즈니스 프로세스로부터 감사 추적 데이터(audit trail data)를 실시간에 근접한 주기로 통합하여 잠재적인 품질속성(Key Performance Indicator)를 식별하고, 의사 결정의 지연을 방지할 수 있는 포괄적인 정보를 공급자와 수요자 모두에게 제공한다는 것이다. 하지만 제안된 아키텍처와 프로세스가 추상적이고, 모니터링 대상과 그에 따른 데이터들의 수집 방법은 언급되지 않았으며, 상세화된 오퍼레이션 처리, 컴포넌트, 인스턴스(단일서비스, 서비스 엔진, 바인딩 컴포넌트 등)를 고려 할 필요가 있다.

Baresi의 연구에서는 기존에 제시한 모니터링 기법

Dynamo와 Astro의 통합한 모니터링 프레임워크를 제안하였다[5]. Dynamo는 관점지향 프로그래밍(Aspect-Orieted Programming, AOP)을 사용하여 BPEL 프로세스가 동작하는 동안 BEPL 엔진이 수집할 수 있는 모니터링 데이터를 얻어 오는 방법이다. Astro는 독립된 소프트웨어 모듈을 사용하여 RTML(Run-Time Monitor specification Language)로 정의된 속성을 확인하는 방법이다. 이 논문에서는 두 가지 방법을 통합하여 BPEL 동작 시 더욱 정교한 모니터링 결과를 얻을 수 있는 방법을 제시한다. 하지만 수집되는 이벤트의 메타모델이나 구체적인 형태는 언급하지 않았으며, BPEL에서 주고받는 외부 모니터링 데이터를 획득하는 방법을 설명하면서 내부 모니터링 데이터와 서비스 모니터링 데이터는 언급하지 않았다. 또한, 모니터링 과부하 부분은 BPEL 엔진이 서비스를 호출하는데 영향을 미치지 않는다고 하여 직접적인 언급을 하지 않았다.

Lin의 연구에서는 Enterprise Service Bus(ESB)를 확장시킨 Llama라는 미들웨어를 제안하였다[6]. 이 미들웨어는 서비스 모니터링을 위한 컴포넌트, 서비스의 결함을 알기 위한 컴포넌트, 서비스 결함을 진단하는 컴포넌트로 구성된다. Llama에서 서비스의 실행시간을 모니터링 하기 위한 방법으로 ESB에서 제공하는 모니터링 API나 추가한 컴포넌트인 Profiling Interceptors를 사용하여 모니터링 데이터를 얻어오는 방법을 제시한다. 하지만 주고 받는 메시지를 획득하는 방법만을 제안했기 때문에 내부 모니터링 데이터는 획득 할 수 없으며, 구체적이고 상세화된 오퍼레이션 처리, 컴포넌트, 인스턴스를 고려할 필요가 있다.

3. 이벤트 주도 서비스 모니터링

3.1 서비스 지향 아키텍처 vs 이벤트 주도 아키텍처

가트너 그룹은 이벤트들이 느슨하게 연결된 이벤트 소비자들 사이에서 라우팅되는 시스템, 서비스 그리고 애플리케이션의 설계 및 구축을 위한 아키텍처인 EDA를 소개했다[7]. 이벤트는 서비스, 비즈니스, 컴포넌트나 시스템에서 어떠한 변화가 생기더라도 일어날 수 있으며, 이벤트 드리븐은 이러한 변화에 대한 시기적절한 대응 측면에서 전통적인 매커니즘 보다 강력하다.

SOA는 애플리케이션 독립적인 서비스들의 재 사용을 극대화함으로써 IT 적응성과 효율성을 증가시키지만, 전통적인 요청/응답(request/reply), 원 투 원 의사소통(one-to-one communications) 매커니즘에 기반하고 있다. 이에 반해 EDA는 완전히 독립적인 이벤트간의 상호작용, 매니 투 매니 의사소통(many-to-many communications)을 사용하며, 애플리케이션 혹은 컴포넌트들의 설계와 확장에 있어서 SOA를 보완하는 역할을 한다[1]. 특히, 기업의 내/ 외부에서 발생하는 이벤트들을 효과적으로 수집하고, 유용한 비즈니스 정보의 제공 및 응대(response), 조치(action)를 취하는 데 있어 EDA는 여러 장점을 제공한다[1].

표 1은 이 두가지 기술의 장, 단점을 8가지 항목으로 요약한 것이다. 상호작용 모델은 풀링 모델(Pulling Model)과 푸쉬 모델(Pushing Model)로 구분되는데[8], 풀링 모델에서는 이벤트 수요자가 채널을 통해 이벤트를 요청하면 이벤트 공급자는 채널을 통해 요청된 이벤트를 기다리다가 요청이 도착했을 때 해당 이벤트 데이터를 생성하고 채널을 통해 리턴하고 수요자는 결과값을 받는다. 푸쉬 모델에서는 이벤트 공급자가 능동적으로 이벤트를 생성하고 이벤트 채널에 보내면 이벤트 수요자는 채널로부터 생성된 이벤트를 비동기적으로 실시간에 받는다.

상호작용 스타일은 SOA가 ESB의 중앙 집중적인 관리하에 하나의 명령을 계층적 구조의 기능적 분할 요소간 수행하는 커맨드 앤 컨트롤(Command-and-Control) 스타일 인 것에 반해, EDA는 수평적 구조의 연합된 자율 프로세싱(Federated autonomous processing) 스타일이다[2].

조정 능력(Controllability)은 두 요소간의 상호작용을 얼마나 잘 관리하는 가의 척도이고, 소요시간(Trip Time)은 두 요소간의 상호작용에 소요되는 시간이다.

표 1 서비스 지향 아키텍처 vs 이벤트 주도 아키텍처

평가 항목	서비스 지향 아키텍처	이벤트 주도 아키텍처
의사소통 패턴 (Communication Pattern)	request-and-reply	publish-and-subscribe
의사소통 패러다임 (Communication Paradigm)	Synchronous	Asynchronous
상호작용 모델 (Interaction Model)	Pulling Model	Pushing Model
상호작용 스타일 (Interaction Style)	Vertical command-and-control style	Horizontal federated processing style
조정능력 (Controllability)	High	Low
소요시간 (Trip Time)	Long	Short
처리량 (Throughput)	Low	High
의사소통 과부하 (Communication Overhead)	High	Low

표 1에서 살펴본 바와 같이 SOA는 요청 및 응답, 동기화(Synchronous) 방식의 커맨드 앤 컨트롤 타입의 의사소통 방식, 풀링 방식의 상호작용 모델로 의사소통 과부하가 많이 발생하며, 소요 시간이 오래 걸려 서비스에서 일어난 이벤트나 익셉션 발생 등 빠른 처리를 요하는 모니터링에는 적합하지 않다. 이에 반해 이벤트 주도 아키텍처는 발행 및 구독(Publish-and-Subscribe), 비동기화(Asynchronous) 방식, 푸쉬 방식의 상호작용 모델 등 느슨한 결합에 유용한 상호작용 패러다임을 가지고 있으며, 짧은 소요시간과 많은 처리량 그리고 적은 의사소통 과부하 등 서비스 모니터링에 있어 주요한 장점들을 가지고 있다.

3.2 이벤트 주도 서비스 모니터링의 구성 요소 및 기술

본 절에서는 3.1절에서 살펴 본 8가지 평가 항목에서 효율적인 서비스 모니터링을 위한 장점들을 고려하여 모니터링 프레임워크의 구조를 제안하고 적용 기술을 알아본다.

이벤트 주도 모니터링 프레임워크의 논리적 구조는 그림 1과 같이 정보시스템 또는 업무 프로세스에서 발생하는 이벤트를 수집(Collect)하고, 정의된 규칙과 업무 요건에 따라서 발생한 이벤트를 분석(Analyze)하며 적절한 시점에 해당 담당자에게 보고(Action)하는 기능 대응의 3 요소로 정의할 수 있다.

그림 1에서 모니터링 인터페이스는 외부와의 인터페이스를 담당하는 부분으로 기본적으로 웹서비스인터페이스를 채택하며 모니터링 프레임워크에게 외부의 이벤트를 전달하는 역할을 한다.

EDA에서 채택하고 있는 푸쉬 방식의 상호작용 모델은 필요로 하지 않는 여분의 모니터링 데이터가 전송될 수 있기 때문에 그로 인한 모니터링 프레임워크의 과부하가 발생하게 된다. 이벤트 필터는 필요로 하지 않는 여분의 모니터링 데이터가 전송될 경우 발생하는 과부하를 줄이기 위해 이벤트 중에서 처리할 이벤트 만을 획득하도록 이벤트를 필터링하는 역할을 한다. 이벤트 프로세서는 이벤트에서 데이터를 추출 및 가공, 정제하는 부분으로 이벤트의 데이터를 모니터링 액션, 디스플레이 요소들이 사용 가능한 데이터로 가공하는 역할을 한다. 각 구성요소 별 자세한 내용은 4장에서 기술한다.

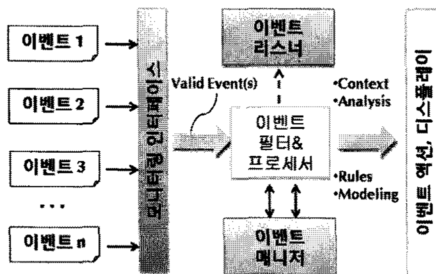

그림 1 이벤트 주도 모니터링 프레임워크의 논리적 구조

3.3 이벤트 주도 서비스 모니터링의 대상

본 절에서는 목표 시스템이 모니터링하는 서비스의 대상을 정의한다. 서비스 모니터링의 대상은 그림 2와 같이 의미있는 3가지 데이터 타입으로 분류할 수 있다.

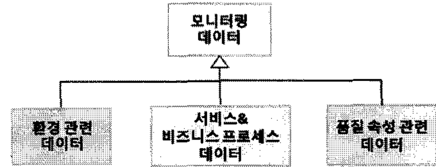

그림 2 모니터링 데이터 분류

환경 관련 데이터는 가상 머신의 상태, CPU와 메모리의 효용성(Utility), ESB상에서 서비스 배치 토폴로지(Topology) 등 하드웨어적 특성을 반영한 서비스 실행 환경에 대한 정보 또는 통계 자료이다.

서비스&비즈니스프로세스 데이터는 서비스나 비즈니스 프로세스의 상태와 진행 상황, 액티비티 별 처리 상황 및 관련 기록 조회 등을 포함한다. 예를 들어 서비스가 실행 중 혹은 초기화 상태인지, 업데이트가 되었는지, 그리고 발생한 이벤트 정보 등이다.

품질 속성 관련 데이터는 서비스나 비즈니스 프로세스의 품질 속성을 측정한 데이터로, 예를 들어 현재의 '응답시간은 120초이며 가용성은 95%이다'와 같은 서비스 품질 속성과 '고객 불만 접수 건수가 10건이며, 처리 건수가 5건이다'와 같은 비즈니스 프로세스 진행 상의 품질속성 값을 반영한다.

본 논문에서는 모니터링의 효율성을 극대화 하기 위해 각 모니터링 대상 별로 적합한 상호작용 모델을 적용한다. 상호 작용 모델이란 데이터를 주고받기 위해 자주 사용될 수 있는 일반화된 방법으로 풀링 모델과 푸쉬 모델로 구분되며[8], 모니터링 시스템에 적용할 수 있는 특징 및 적용 가능한 경우를 정리하면 표 2와 같다[9].

일반적으로 비즈니스와 관련된 서비스와 비즈니스 프로세스는 순간적으로 빠르게 일어나기보다는 여러 조건과 상황을 종합하여 처리되므로 어느 정도의 시간이 소요된다[10]. 따라서 요청을 보낸 후 응답을 대기하기만 하는 폴링 모델은 비효율적이며 의사소통 과부하가 발생한다. 또한, 서비스&비즈니스 프로세스 데이터는 비정기적으로 발생하고, 긴급성이 요구되는 정보들로 푸쉬 모델을 적용하는 것이 바람직하다. 하지만 푸쉬 방식의 상호작용 모델은 필요로 하지 않는 여분의 모니터링 데이터가 전송될 수 있기 때문에 그로 인한 과부하가 발생하게 되는데, 이는 이벤트 필터를 통해 해결한다.

표 2 모니터링 상호작용 모델과 적용 대상

상호작용 모델	특 징	적용가능한 경우/ 모니터링 대상
풀링 (Pulling)	특정 컴포넌트가 인터페이스를 통하여 필요한 데이터를 요청하면, 인터페이스를 구현한 컴포넌트가 응답한다.	데이터의 유효성이 중요한 경우, 데이터의 보안성이 요구되는 경우/ 환경 관련 데이터
푸쉬 (Pushing)	풀링과의 차이점은 인터페이스를 구현한 컴포넌트가 특정 컴포넌트에 전송하여 알린 후 데이터에 접근 하도록 한다는 것이다. 즉, 공급자가 능동적으로 데이터를 생성하고 전송하면 수요자는 생성된 데이터를 비동기적으로 실시간에 받는다.	데이터 전송의 긴급성이 요구되는 경우, 데이터의 수집 빈도가 일정하지 않은 경우/ 서비스&비즈니스프로세스 데이터, 품질 속성 관련 데이터

품질 속성 관련 데이터는 서비스 및 비즈니스 프로세스의 품질과 직결되며 서비스 모니터링에서 중요한 정보이기 때문에 관리자는 품질 속성의 변화가 있을 때, 즉시 이를 알고자 할 것이다. 또한, SOA 환경에서 품질 속성 관련 데이터는 빈번하게 변화될 수 있으며, 품질 속성 지표 역시 수정이 필요시 즉시 반영하여야 하므로 이벤트를 정기적으로 수집하는 것보다 즉시 수집하는 것이 효율적이다. 따라서, 이 경우 푸쉬 모델을 적용한다.

서비스가 존재하는 서버 노드의 하드웨어적 특성을 반영하는 환경 관련 데이터의 경우 비교적 변화가 자주 발생하지 않으며, 따라서 관리자는 필요 시 혹은 일정한 주기별로 확인하고자 할 것이다. 이 경우 발행 및 구독의 푸쉬 모델을 구현하기보다는 SOA 상호작용의 일반적인 모델인 풀링 모델을 적용한다.

3.4 풀링 및 푸쉬 모델의 적용 기법

본 절에서는 3.3절에서 살펴본 서비스 모니터링의 대상 별 적합한 상호작용 모델을 적용하기 위한 기법을 제시한다. 풀링 모델에서는 모니터링 데이터를 수집하는 모니터링 프레임워크가 필요한 모니터링 데이터를 서비스나 서비스가 배치되어 있는 웹 서비스 서버, 서비스 메시지를 전송하는 ESB 등에 요청하면, 그에 따라 서비스, 웹 서비스 서버, ESB 등이 응답하는 방식이다[9]. 그림 3은 풀링 모델을 적용한 서비스 모니터링 구조를 메시지 처리 흐름과 함께 표현한 것이다. (A)는 이벤트 수요자 즉 모니터링 프레임워크가 ESB를 통해 웹 서비스 서버에 요청 메시지를 보낸다. (B)는 Binding Component가 메세지를 받고, (C)는 Binding Component가 Message Router를 통해 Service Container로 메시지를 전송한다. (D)는 Service Container가 Message Router를 통해 Binding Component에 응답을 보낸다. (E)는 ESB를 통해 응답을 받고, (F)는 모니터링 프레임워크가 메시지를 받고 응답이 콘솔에 출력된다.

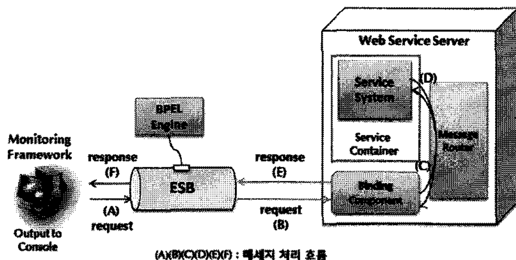

그림 3 풀링 모델을 적용한 서비스 모니터링

푸쉬 모델은 모니터링 데이터가 생성되는 이벤트 공급자가 직접 모니터링 데이터를 수집하는 모니터링 프레임워크에 전달하는 방식이다[9]. 그림 4는 서비스 시스템에 메시지를 직접 보내지 않고 JMS를 활용, 토픽을 공유함으로써 발행 및 구독의 푸쉬 모델을 구현한 서비스 모니터링 구조를 메시지 처리 흐름과 함께 표현한 것이다. 물론 JMS를 활용하지 않더라도 푸쉬 모델을 구현할 수 있다. Event-Notification 기술[11]을 적용하여 이벤트를 발생 시키면 알림(Notification)을 받은 모니터링 프레임워크는 모니터링 인터페이스를 통하여 모니터링 데이터를 수집한다. 모니터링 가능한 서비스의 내부 모니터링 데이터를 푸쉬 모델을 이용하여 가져오는 것이다. 본 논문에서는 두 가지 기술을 모두 사용한다. 비즈니스프로세스 데이터는 실행모델인 BPEL 내부에 이벤트 포맷을 포함시킴으로써 내부 데이터를 획득한다. 이렇게함으로써 내부 모니터링 데이터를 획득 및 분석할 수 있다.

그림 4에서 (A)는 JMS Client가 ESB를 통해 Topic에 접속하여 메시지를 보내고, (B)는 Topic의 가입자인 Binding Component가 메세지를 받고, (C)는 Binding Component가 Message Router를 통해 Service container로 메시지를 전송한다. (D)는 Service container가 Message Router를 통해 Binding Component에 응답을 보낸다. Binding Component는 JMS Template을 사용하여 메시지를 발행한다. (E)는 JMS Template은 JMS Factory를 사용하여 토픽과 연결된 포트의 접속을 얻고, 토픽에 메시지를 발행한다. (F)는 JMS Client는 메시지를 받고 응답이 콘솔에 출력된다.

3.5 이벤트 메타모델

이벤트 메타모델은 수집된 이벤트 데이터를 잘 분석하여 비즈니스 수준의 유용한 정보를 제공하기위해 모니터링 프레임워크 서버가 서비스, BPEL, 액티비티, 데이터 베이스 등의 이벤트 소스(source)가 되는 곳으로부터 이벤트를 받을 때 필요한 이벤트의 형식을 정의한 모델이다[3]. 이벤트는 서로 독립적인 것이 아니라 본질

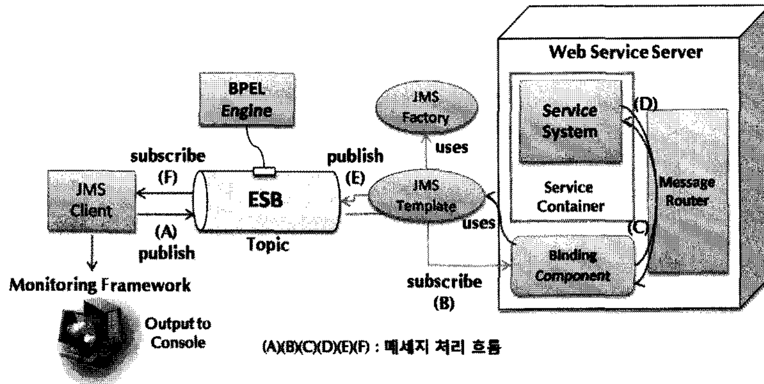

그림 4 푸쉬 모델을 적용한 서비스 모니터링

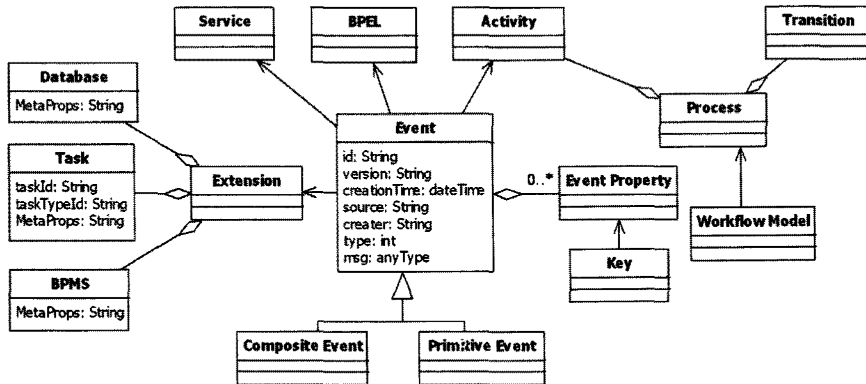

그림 5 이벤트 메타모델

적으로 다른 요소들과 서로 밀접한 관계를 가지고 있으며, 이를 정형화된 메타모델로 표현하면 그림 5와 같다.

본 논문에서 제안한 이벤트 메타모델은 OASIS WEF[3] 표준을 기반으로 정의되었으며, 쉽게 확장할 수 있는 매커니즘이 포함되어 설계 되었으므로 타 모니터링 프레임워크에 이식 및 확장이 가능하다.

이벤트는 일반적으로 원시 이벤트와 복합 이벤트로 분류될 수 있으며, 둘 모두 동일한 이벤트 속성을 가진다. 이벤트들은 서비스, BPEL, 액티비티 그리고 그 외 여러 이벤트 소스로부터 수집될 수 있다. 본 논문의 메타모델에서는 'Extension'요소를 통해서 확장이 가능하다.

표 3은 이벤트 메타모델의 구성 요소를 설명한 것이다. 이들 구성 요소는 이벤트 처리 모델 설계 시 사용되고, 나아가 모니터링 프레임워크를 설계하는 기초가 된다.

4. 모니터링 프레임워크의 설계

4.1 모니터링 프레임워크의 설계 기준

SOA에서 서비스들은 분산환경에서 동작하며, 수많은 서비스 혹은 이벤트 공급자와 수요자가 존재한다. 또한, 다수의 이질적인 하드웨어, 소프트웨어 플랫폼 상에서 실행된다. 이러한 특성을 고려하여 3.1절에서 살펴본 모니터링 측면에서의 EDA의 장점을 도입한 이벤트 주도 동적 모니터링 프레임워크를 제안하기 위해 아래와 같은 모니터의 설계기준을 제시한다.

- 표준의 준수: 본 논문에서 제안하는 모니터링 프레임워크에서 준수하는 주요 표준은 먼저 WSDL(Web Service Definition Language)[12], SOAP과 같은 웹 서비스 기본 표준과 비즈니스 프로세스 실행과 관련한 표준인 WS-BPEL(Web Services Business Process Execution Language)[13]이 있으며, 분산된 서비스 관리와 이벤트 포맷을 정의하기 위한 WSDM, WEF[3]등의 표준을 준수함으로써 모니터링 프레임워크의 확장성과 이식성, 호환성 등을 보장할 수 있다.
- 구현 플랫폼의 비 종속성: 서비스는 다양한 SOA 플랫폼에서 개발될 수 있다. 따라서 하위 레이어의 플랫폼 종속성은 피할 수 없지만 이들 다양한 서비스들을

표 3 이벤트 메타모델의 구성 요소

이 름	설 명
Event	이벤트 소스로부터 발생하는 이벤트 정보를 담는 부분으로서 아래와 같은 구성 요소로 이루어져 있다.
id: String	이벤트의 처리모델을 식별해 주는 아이디로서 이벤트 처리모델의 식별은 아이디와 버전을 통해서 이루어진다.
version: String	이벤트의 처리모델을 식별해 주는 버전이다.
creationTime:dateTime	이벤트가 발생한 시간
source : String	이벤트가 발생한 위치를 의미한다. 단순한 문자열로 표시하나, type 속성과 연계하여 이벤트 소스를 명시적으로 표현할 수 있다.
creater : String	이벤트를 발생시킨 사람의 아이디
type : integer	이벤트가 발생한 위치 즉, 이벤트 소스를 표시한다. 'source'속성과 같은 의미이나 보다 명시적으로 표현하기 위해 숫자를 사용한다. 0은 일반 이벤트 타입, 1~3은 기본 이벤트 타입, 4~6은 추가 이벤트 타입을 나타냄. ex) 0 : 일반 이벤트 타입 - 식별되지 않은 이벤트, 1 : Service 이벤트 타입, 2 : BPEL 이벤트 타입, 3 : Activity 이벤트 타입, 4 : Database 이벤트 타입, 5 : Task(작업) 이벤트 타입, 6 : BPMS 이벤트 타입
msg : anyType	XML 형식의 복합구조 데이터로 이벤트 데이터가 전달되는 부분
Event Property	프로퍼티 형식(key-value)으로 이벤트의 데이터를 전달하는 부분. Property 속성안에 value를 표시
Key	Property의 value에 해당하는 key
Extension	이벤트 소스 중 추가되는 이벤트 소스 정보를 담는 부분으로 Type이 0(일반 이벤트 타입)과 1~3(기본 이벤트 타입)이 아닌 경우에 각 경우에 맞는 형식의 정보를 표시
database : tDatabaseExt	Database 시스템의 확장정보
task : tTaskInstance	작업관리자 Task 인스턴스의 확장정보
bpms : tBPMSExt	BPMS와 관련된 확장정보

모니터링하고, 그 품질을 측정해야 하는 모니터링 프레임워크는 폭넓은 적응성과 응용성을 위해 플랫폼 독립적으로 구현되어야 한다.

• 효율성: 모니터링 대상의 특성에 따라 적합한 상호작용 모델을 적용함으로써, 상호작용 시 발생하는 과부하를 최소화 할 수 있도록 설계되어야 하며, 모니터링 프레임워크의 성능 측면에서도 과부하로 인해 발생될 수 있는 문제를 줄일 수 있도록 설계되어야 한다.

• 확장성: 좋은 소프트웨어 툴은 이클립스 플랫폼처럼 항상 새로운 기능의 추가나 변화 시 기존 소프트웨어의 변화를 최소화할 수 있는 매커니즘이 포함되어 설계 되어야 한다[14]. SOA에서 서비스는 새로운 요구사항을 충족시키기 위해 언제나 진화할 수 있으므로 향후 확장성을 고려한 설계는 효율적인 서비스 모니터링을 위하여 필수적으로 포함되어야 한다.

• 이해성: 모니터링 프레임워크는 기능적으로 다수의 서비스들과 비즈니스 프로세스를 관찰하고 품질을 측정하기 때문에 수집한 이벤트의 분석 정보를 한 눈에 쉽게 이해할 수 있어야 한다. 따라서, 유용한 정보를 쉽게, 효율적으로, 효과적으로, 그리고 직관적으로 서비스 관리자에게 보여줄 수 있어야 한다.

4.2 전체 아키텍처

본 절에서는 설계 기준을 준수하는 이벤트 기반의 동적 모니터링 프레임워크 아키텍처를 제안한다.

그림 6과 같이 모니터링 프레임워크를 설계하기 위해서 데이터 계층, 처리 계층, 표현 계층의 계층 구조 아키텍처를 적용하였다. 이 아키텍처 스타일은 각 계층을 독립적으로 관리함으로서 각 계층의 모니터링의 역할을 분리한다.

• 데이터 계층: 데이터 계층은 서비스, 비즈니스 프로세스, 그리고 환경 관련 데이터 등 모니터링 프레임워크가 모니터링해야 하는 이벤트 소스들을 포함한다. 정의된 규칙과 요건에 따라 이벤트를 분석하기 위해 처리 계층에서 필요한 이벤트 데이터들이 발생하고 생

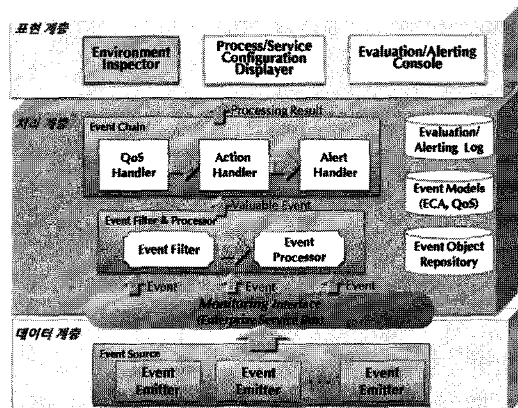

그림 6 모니터링 프레임워크 아키텍처

성되는 계층이다.

- **처리 계층**: 처리 계층은 크게 3가지의 주요 컴포넌트들로 구성된다. 모니터링 인터페이스는 외부와의 인터페이스를 담당하는 부분으로 모니터링 프레임워크에게 데이터 계층의 이벤트를 전달하는 역할을 한다. 본 논문에서는 SOA에서 가장 많이 사용되고 있으며, 복합 서비스의 구성과 서비스 관리 기술로 조정능력이 뛰어난[15] ESB를 모니터링 인터페이스로 채택한다. 이벤트 필터와 프로세서는 처리할 이벤트만을 획득하기 위하여 이벤트를 필터링하고, 이벤트 데이터를 이벤트 체인(Event Chain)에서 사용할 수 있도록 추출 및 가공, 정제하는 역할을 한다. 모니터링 프레임워크가 목표로 하는 정보를 얻어내거나 액션을 취할 실질적인 작업은 이벤트 체인에서 담당하며, 품질속성 핸들러(QoS Handler), 액션 핸들러(Action Handler), 알림 핸들러(Alert Handler)로 구성되어 있다. 처리계층에서는 또한 모니터링을 위한 지속적인 데이터를 저장하는 3개의 공유 저장소를 관리하는데, 가공되지 않은 이벤트 로우 데이터를 저장하는 Event Object Repository, 수집된 데이터를 측정하기 위한 기준이 되는 모델을 저장하는 Event Models, 측정 결과 및 알림의 히스토리 데이터를 저장하는 Evaluation/Alerting Log Repository가 있다.
- **표현 계층**: 표현 계층은 처리 계층에서 처리된 결과를 다양한 그래픽 디스플레이로 제공받을 수 있는 환경을 지원한다. 환경 관련 데이터들의 상태와 변화에 대한 정보를 확인하여 디스플레이하는 환경 검사기(Environment Inspector), 비즈니스 프로세스 및 서비스의 설정 및 상태, 변화에 대한 정보를 디스플레이하는 프로세스/서비스 구성 디스플레이어(Process/Service Configuration Displayer), 이벤트 처리 결과를 평가하여 발생된 문제점을 사용자에게 즉각적으로 전달하고 인지할 수 있도록 하는 평가/알림 콘솔(Evaluation/Alerting Console)로 구성되어 있다.

4.3 주요 컴포넌트 설계

본 절에서는 4.2절에서 제시된 전체 아키텍처에서 핵심 기능을 하는 주요 컴포넌트들을 상세하게 설계하고, 세부 기능을 정의한다.

이벤트 필터(Event Filter): SOA의 복잡도로 인해, 모니터링 프레임워크는 많은 수의 이벤트들을 생성하며 송, 수신한다. 또한, 필요하지 않은 여분의 이벤트 데이터가 쌓여 모니터링 프레임워크에 과부하가 발생할 수 있다. 이벤트 필터는 모니터링 프레임워크에서 처리하지 않는 이벤트를 필터링하는 역할을 함으로써 이러한 문제를 해결한다. 그림 7에서 볼 수 있는 것과 같이 이벤트 필터는 전역 필터 체인(Global Filter Chain)과 지역 필터 체인(Local Filter Chain)의 이중 필터 구조로 설계한다.

전역 필터에서는 이벤트 메타데이터(타입, 아이디, 버전, 발신위치, 시간, 중요도 등)에 근거하여 이벤트를 필터링하는 역할을 한다. 전역 필터 체인은 모든 이벤트에 대하여 동일하게 적용되도록 메타데이터에 기반하고, 각 이벤트 데이터 형식 구분을 하여 처리하지 않기 때문에 높은 처리능력을 유지한다.

지역 필터에서는 이벤트가 가지고 있는 실제 데이터 즉, 이벤트의 내용(Context)을 근거로 하여 이벤트를 필터링하는 역할을 한다. 예를 들어 '생산량이 100이하인 생산결과 이벤트를 필터링'하는 경우와 같이 이벤트의 내용을 근거로 필터링을 하는 기능을 가지고 있기 때문에 각 이벤트 타입마다 정의되어 있는 이벤트의 데이터

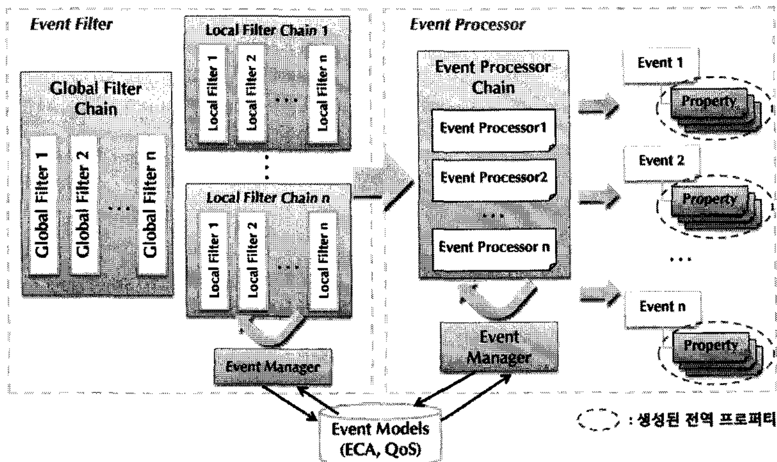

그림 7 이벤트 필터와 프로세서

정의에 기반하여 구성된다. 이처럼 전체 이벤트에 대하여 정의할 전역 필터와 세부 이벤트에서 정의할 지역 필터의 내용을 구분하여 관리와 처리에 편의성을 제공한다.

이벤트 프로세서(Event Processor): 그림 7에서 구성 요소별로 살펴보면 이벤트 프로세서는 이벤트 프로세서 체인과 이벤트 매니저로 구성되어 있다. 각각의 이벤트 프로세서는 해당하는 이벤트의 처리모델을 이벤트 매니저를 통해 찾아서 이벤트에서 데이터를 추출 및 처리하여 전역 프로퍼티(Global Properties)를 생성하여 이벤트에 추가해서 다음 단계인 이벤트 체인에서 데이터로서 사용할 수 있도록 가공 및 정제하는 역할을 한다. 전역 프로퍼티는 실제 구현시 이벤트 객체뿐만 아니라 모니터링 프레임워크의 다른 객체들과 공유하는 프로퍼티로 각 객체는 독립적으로 정의되기 때문에 다른 영역으로 데이터를 전달하기 위해서는 반드시 전역 프로퍼티를 선언하고 할당 식을 통해 원하는 형식으로 데이터를 설정해야 한다. 이에 대한 자세한 내용은 4.4 이벤트 처리모델에서 기술한다. 이벤트 매니저는 이벤트 필터와 이벤트 프로세서에게 해당되는 이벤트 필터 및 처리 정의 모델을 전달하고, 실행시키는 역할을 한다.

품질속성 핸들러(QoS Handler): 품질속성 핸들러는 전달받은 이벤트의 내용과 품질속성 모델(QoS Model)을 기준으로 품질속성 값을 계산하여 데이터베이스에 저장하는 역할을 한다. 품질속성 핸들러는 품질속성 데이터 핸들러(QoS Data Handler)와 품질속성 정보 핸들러(QoS Information Handler)로 나누어진다. 품질속성 데이터 핸들러는 전달받은 이벤트에서 품질속성 데이터만을 추출하여 품질속성 데이터 저장소(QoS Data Repository)에 저장하고, 품질속성 정보 핸들러는 품질속성 데이터로부터 추가적인 계산을 통해 분석된 데이터를 품질속성 정보 저장소(QoS Information Repository)에 저장한다. 품질속성 매니저는 품질속성 데이터를 분석하기 위한 기준이 되는 품질속성 모델 정의 목록을 전달하고, 품질속성 핸들러를 실행시키는 역할을 한다. 그 결과 이벤트에 품질속성 타입의 프로퍼티값이 생성되어 다음 단계인 액션 핸들러로 전송된다. 이를 표현하면 그림 8과 같다.

액션 핸들러(Action Handler): 액션 핸들러는 전달된 이벤트 내용을 이용하여 액션 모델에서 정의한 기준에 따라 대응 웹서비스 호출, 프로세스 호출, 사용자 정의 함수 호출 등과 같은 액션을 수행하는 역할을 한다. 액션에서 중요한 사항은 언제 어떤 액션을 취할 것인가이다. 따라서, 액션 핸들러는 그림 9에서 볼 수 있는 것과 같이 품질속성 핸들러의 처리 결과를 입력값으로 받아 배치된 액션 모델중 지정한 액션 실행 조건이 만족할 경우에만 해당 액션 모델을 실행한다. 액션 매니저는 액션 핸들러에게 액션을 실행하기 위한 조건과 처리하는 방법, 관리를 위한 정보를 전달하고, 액션 핸들러를 실행시키는 역할을 한다. 그 결과 이벤트에 액션 타입의 프로퍼티값이 생성되어 다음 단계인 알림 핸들러로 전송된다.

알림 핸들러(Alert Handler): 알림 핸들러는 전달된 이벤트 내용과 알림 모델을 기준으로 모니터링프레임워크에서 제공하는 알림 테이블에 알림 메시지를 전달하는 역할을 한다. 알림에서 중요한 사항은 언제 누가 누구에게 어떤 메시지를 전달하는가이다. 따라서, 알림 핸들러는 그림 10에서 볼 수 있는 것과 같이 액션 핸들러의 처리 결과를 입력값으로 받아 배치된 알림 모델중 지정한 알림 실행 조건이 만족할 경우 해당 알림 모델을 실행한다. 그리고 이 조건이 만족할 경우 보내는 주체와 받는 주체, 전달할 메시지를 설정하여 전달한다. 알림 매니저는 알림 핸들러에게 알림을 생성하는 방법과 관리를 위한 정보를 전달하고, 알림 핸들러를 실행시키는 역할을 한다. 그 결과 알림 핸들러는 조건에 해당하는 알림 모델의 알림을 생성하여 평가/알림 로그 저장소(Evaluation/Alerting Log Repository)에 저장한다.

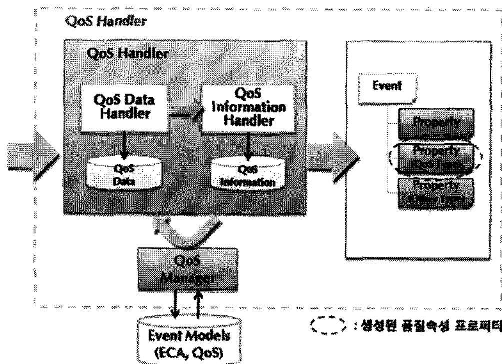

그림 8 품질속성 핸들러

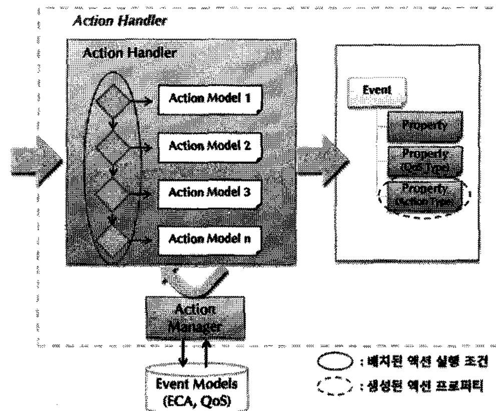

그림 9 액션 핸들러

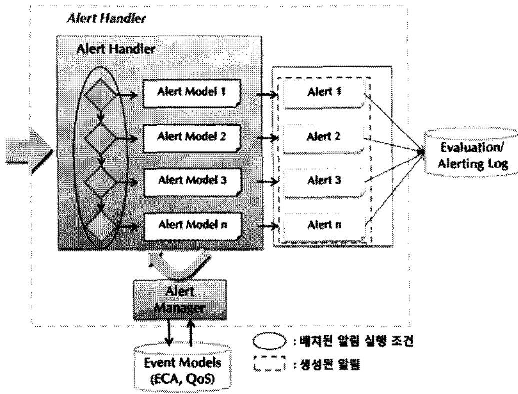

그림 10 알림 핸들러

4.4 이벤트 처리 모델

4.3절에서는 이벤트가 발생했을 때 모니터링 프레임워크가 이에 대해 어떻게 처리하는지에 대한 절차와 각 모듈의 역할을 살펴보았다. 하지만 모니터링 프레임워크가 그렇게 처리하기 위해서는 기준이 되는 정보가 필요하다. 이벤트 처리 모델은 바로 이벤트에 대한 처리 절차 기준 모델이 된다. 본 절은 이러한 이벤트 처리 메타 모델과 구현된 소스코드 예를 살펴본다.

그림 11에서 보는 바와 같이 모니터링 프레임워크의 확장성과 유지보수성을 향상시키기 위하여 모니터링 프레임워크의 주요 객체는 모두 Monitor Object라는 객체에서 파생되도록 하였다. 즉, Monitor Object의 속성은 이벤트 처리모델의 객체들이 공통으로 갖는 속성이다. Event는 이벤트의 유효성을 확인할 수 있는 기준 및 이벤트를 처리할 절차를 제시하는 요소이다. Event의 속성 중 type, message는 이벤트의 유효성 기준의 역할을 하고, assignments는 이벤트가 어떻게 동작할 지를 나타내며, chain은 다음 절차에 대해 알려준다. Event Property는 모니터링 프레임워크의 데이터 상태를 필요로 하는 여러 곳에서 사용된다. 예를 들어 Event 및 Flow Object의 전역 프로퍼티와 지역 프로퍼티를 비롯하여, 이벤트의 메시지 데이터 및 외부 서비스와의 매개변수 등에 사용된다. Event Chain은 품질속성, 액션, 알림 객체와 이들의 진행 순서를 포함하고 있다. Event Chain은 내부 요소를 순서대로 처리하는 간단한 구조라는 장점이 있는 반면, 어떠한 방식으로 흐름을 제어할지에 대한 정보는 없기 때문에, 실행 여부의 조건은 각 Flow Object에서 정보를 제공한다. 즉, Flow Object는 이벤트 체인의 내부 처리 흐름을 어떠한 방식으로 제어할 지에 대한 실행 여부의 조건을 제공함으로써 실질적인 이벤트에 대한 처리를 담당한다.

이벤트 처리에서 가장 중요한 요소는 이벤트 자신과 이벤트 처리 절차에 해당하는 이벤트 체인의 각 요소 그리고 이벤트 처리 절차의 프로퍼티를 할당하는 기능(데이터 처리 기능)이다. 그림 12는 앞에서 설계한 이벤트 처리 모델에 따라 이벤트 클래스와 이벤트 체인의 각 요소에 해당하는 프로퍼티를 할당하는 할당(monitor Assignment) 클래스를 구현한 소스코드이다. 이벤트 클

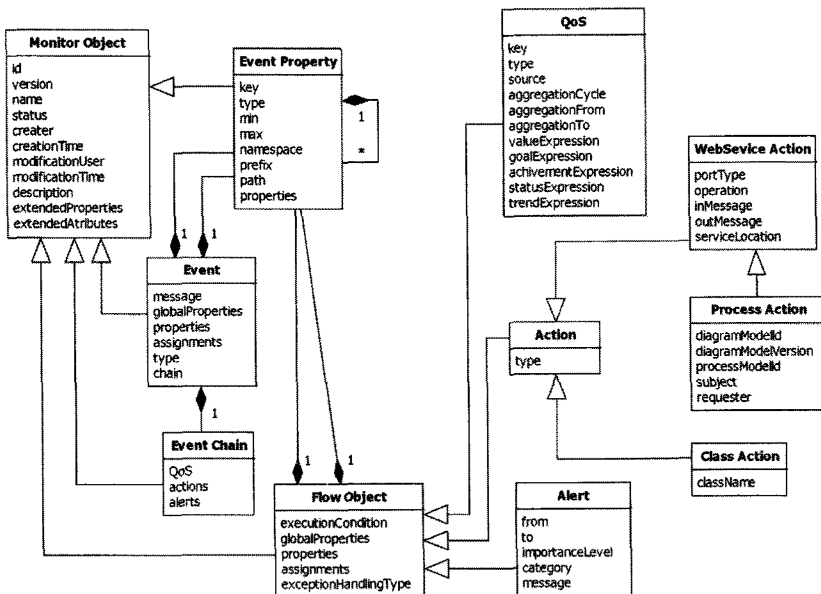

그림 11 이벤트 처리 메타모델

```
...
public class monitorEvent extends monitorObject {
  ...
  private static String toStubString(String className, monitorEvent event, ElemInfo
    if (className == null)
      className = "Event";
    ...
    String id = event.getId(); //이벤트 아이디와 버전, 메시지 정보 획득
    String ver = event.getVersion();
    monitorMessage msg = event.getMessage();
    monitorElement[] elems = null;
    if (msg != null)
      elems = msg.getElements();
    List elemNameList = new ArrayList();
    ...
  }
  public void addAssignment(monitorAssignment obj)
    this.assignments = monitorAssignment.add(this.as
  }
  public monitorAssignment[] getAssignments() {
    return assignments;
  }
  public monitorExpression getEventCondition() {
    return eventCondition;
  }
  ...
  public String getEventType() {
    return eventType;
  }
  public monitorMessage getMessage() {
    return message;
  }
  public String getCycle() {
    return cycle;
  }
```

이벤트 처리 절차의 프로퍼티를 할당하기 위한 공간 확보

```
public class monitorAssignment extends ClassObject {
  ...
  private static final String A_FROM = "from";
  private static final String A_TO = "to";
  ...
  public static monitorAssignment[] add(monitorAssignment[] objs,
    if (obj == null)
      return objs;
    int size = 0;
    if (objs != null)
      size = objs.length;
    monitorAssignment[] newObjs = new monitorAssignment[size+1];
    int i;
    for (i=0; i<size; i++)
      newObjs[i] = objs[i];
    newObjs[i] = obj;
    return newObjs;
  }
  ...
  public monitorExpression getFrom() {
    return from;
  }
  public void setFrom(monitorExpression from) {
    this.from = from;
  }
  public monitorElementRef getTo() {
    return to;
  }
  public void setTo(monitorElementRef to) {
    this.to = to;
```

이벤트 체인의 입/ 출력 요소 지정

그림 12 이벤트와 프로퍼티 할당 클래스 소스코드

래스에서는 먼저 이벤트 아이디, 버전, 메시지 등의 정보를 획득하고, 프로퍼티를 할당하기 위한 공간을 확보하는 할당 클래스의 *add* 함수를 호출한다.

그림 13은 이벤트 체인 요소 중 하나인 품질속성 클래스와 품질속성 값을 추가/ 삭제하고 목표치, 달성도, 상태 등을 획득 및 설정하는 품질속성값(QoSValue) 클래스를 구현한 소스코드이다. 품질속성 클래스에서는 품질속성 데이터의 집계주기, 시작/종료 시간 등의 정보를 획득하고, 품질속성 값을 추가하는 품질속성값 클래스의 *add* 함수를 호출한다. 체인의 다른 요소인 액션과 알림

```
...
public class QoS extends monitorActivity {
  ...
  private monitoringDataType[] type; // 비즈니스 프로세스,서비스 구분
  private String cycle = CYCLE_ALL;  // 집계 주기
  private Date from = null;          // 집계 시작
  private Date to = null;            // 집계 종료
  private monitorElementCondition[] classifications; // 품질 속성 구분
  private QoSValue[] values;
  ...
  }
  ...
  public void addValue(QoSValue obj) {
    this.values = QoSValue.add(this.values, obj);
  }
  public void removeValue(QoSValue obj) {
    this.values = QoSValue.remove(this.values, obj)
  }
  public String getCycle() {
    return cycle;
  }
  public Date getFrom() {
    return from;
  }
  public Date getTo() {
    return to;
  }
  public QoSValue[] getValues() {
    return values;
  }
  public void setValues(QoSValue[] values) {
    this.values = values;
  }
```

품질속성 값 추가

```
public class QoSValue extends ClassObject {
  ...
  private MonitorExpression value;        // 품질 속성 값
  private MonitorExpression goal;         // 목표치
  private MonitorExpression achievement;  // 달성도
  private MonitorExpression status;       // 상태
  private MonitorExpression trend;        // 변화 추세
  }
  ...
  public static QoSValue[] add(QoSValue[] objs, QoSValue obj) {
    if (obj == null)
      return objs;
    int size = 0;
    if (objs != null)
      size = objs.length;
    QoSValue[] newObjs = new QoSValue[size+1];
    int i;
    for (i=0; i<size; i++)
      newObjs[i] = objs[i];
    newObjs[i] = obj;
    return newObjs;
  }
  public MonitorExpression getAchievement() {
    return achievement;
  }
  public MonitorExpression getGoal() {
    return goal;
  }
  public MonitorExpression getStatus() {
    return status;
  }
  ...
```

목표치, 달성도, 상태 값 획득

그림 13 품질속성 관리 클래스 소스코드

역시 같은 방법으로 관리된다.

5. 서비스 모니터링의 효율성 평가 메트릭

본 장에서는 이벤트 주도 서비스 모니터링의 효율성을 평가하기 위한 메트릭을 정의하여, 이를 실험을 통하여 평가할 수 있는 근거를 제시한다. 서비스 모니터링의 성능 품질 속성은 소요시간(Trip Time), 그리고 처리량(Throughput) 두 가지의 메트릭을 기반으로 정의 및 측정되며[6], 이것은 3.1절 표 1의 평가 항목에도 포함되어 있다. 서비스 모니터링은 모니터링 데이터의 전송을 얼마나 효율적, 효과적으로 하느냐에 달려 있으므로[6], 서비스 모니터링의 효율성은 모니터링 대상과 모니터링 프레임워크간의 상호작용 모델에 크게 의존한다. 따라서, 기존의 폴링 모델과 이벤트 주도 서비스 모니터링의 푸쉬 모델을 비교함으로써 이벤트 주도 서비스 모니터링의 효율성을 평가할 수 있다.

5.1 소요시간(Trip Time)

소요시간은 모니터링 프레임워크와 모니터링 대상과의 상호 작용을 초기화하는 것에서부터 끝내는 것까지에 대한 시간이다. 즉, 소요시간은 얼마나 빨리 모니터링 프레임워크가 이벤트를을 받을 수 있는지를 의미한다. 상호작용의 완료 시간을 TT, 시작 시간을 ST라 하면, 소요시간 RT는 다음과 같다.

$$RT = TT - ST$$

폴링 모델에서의 소요시간은 요청을 보내는 시점으로부터 응답을 받기까지의 시간이다[8]. 요청을 보내는 시간을 T_{Req}, 요청에 대하여 실행되는 동안의 시간을 $T_{Process}$, 완료되어 응답받는 시간을 T_{Res}라 하면, 폴링 모델에서의 소요시간 T_{Pull}은 $T_{Pull} = TT - ST = T_{Req} + T_{Process} + T_{Res}$가 된다.

푸쉬 모델에서의 소요시간은 모니터링 대상으로부터 통지(Notification)되는 시간이다[8]. 통지되는 시간을 T_{Notify}라 하면 푸쉬 모델에서의 소요시간 T_{Push}는 $T_{Push} = TT - ST = T_{Notify}$가 된다.

그림 14는 서비스 모니터링의 소요시간 메트릭을 폴링과 푸쉬 상호작용 모델로 표현한 것이다. 만약 네트워크 대역폭의 성능 영향이 무시할 수 있을 정도라고 가정한다면 $T_{Req} \approx T_{Res} \approx T_{Notify}$이므로 $T_{Push} < T_{Pull}$이다.

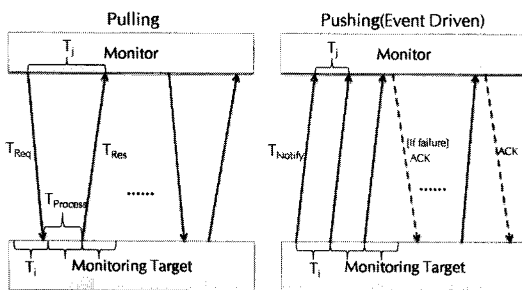

그림 14 폴링과 이벤트 주도의 푸쉬 상호작용 모델

5.2 처리량(Throughput)

처리량은 정해진 단위 시간 동안 모니터링 프레임워크가 얼마나 많은 이벤트를 처리하였는지의 정도, 즉 처리된 이벤트의 수를 측정하는 메트릭이다. 단위 시간당 처리할 수 있는 이벤트의 수가 많음은 소요시간이 짧아지는데 많은 영향을 미친다. 따라서, 처리량과 소요시간은 밀접한 관계에 있다고 할 수 있다. 처리량을 측정하는 전통적인 방법은 단위 시간 당 전송 및 처리된 이벤트의 수를 측정하는 것이다[6]. 모니터링 프레임워크가 전달받아 처리한 이벤트의 수를 N, 모니터링 프레임워크에 전달된 각 이벤트의 처리 시간을 T_i라 하면, 처리량 TP는 다음과 같다.

$$TP = N / \sum_i^n T_i$$

폴링 모델에서의 처리량 TP_{pull}은 $TP_{pull} = N / \sum_i^n (TT_i - ST_i) = N / \sum_i^n (T_{Req} + T_{Process} + T_{Res})$이 된다. T_{Req}, $T_{Process}$, T_{Res}에 대한 정의는 5.1절에서의 정의와 같다.

푸쉬 모델에서의 처리량 TP_{push}는 $TP_{push} = N / N * T_{Interval} = 1/T_{Interval}$이 된다. $T_{Interval}$은 모니터링 대상으로부터 순차적으로 발생한 두 이벤트 사이의 시간 간격을 의미한다.

만약 $\sum_i^n (T_{Req} + T_{Process} + T_{Res}) > N * T_{Interval}$이면, $TP_{pull} < TP_{push}$이고, 만약 $\sum_i^n (T_{Req} + T_{Process} + T_{Res}) <= N * T_{Interval}$이면, $TP_{pull} >= TP_{push}$이다.

6. 사례연구

본 장에서는 이벤트 주도 서비스 모니터링의 효율성 및 실용성을 보여주기 위해서, 모니터링 프레임워크의 프로토타입과 호텔 예약 서비스 시스템(Hotel Reservation Service System, HRSS)을 구현하고, 앞장에서 제안된 메트릭을 적용한 사례연구를 수행한다. 본 사례연구에서의 개발 및 운영환경은 다음과 같다.

개발 환경 명세: 본 논문에서 사례 연구를 위한 개발환경은 모두 오픈 소스로 구현함으로써 연구 결과의 적응성과 확장성, 플랫폼 비 종속성을 추구하였다. 비즈니스 프로세스 모델링 및 실행을 위한 BPEL Engine은 BPEL 2.0 표준을 준수하는 Apache ODE[16], ESB는 벤더와 플랫폼에 비종속적인 통합을 가능하게 하는 Java Business Integration(JBI) 기반의 Apache ServiceMix [17], 개발 플랫폼은 Eclipse를 사용하였으며, DBMS는 PostgreSQL[18], 웹서버는 Apache Tomcat을 사용하였다.

운영 환경 명세: HRSS는 서비스 소비자 계층(Service Consumer Layer), 서비스 계층(Service Layer)과 같이 2개의 계층으로 구성되어 있다. 서비스 소비자 계층에는 서비스 소비자의 서비스 어플리케이션이 배치되어있고, 서비스 계층에는 호텔 예약 서비스가 배치되어있다. HRSS의 각 계층 구현과 실행을 위한 하드웨어 환경은 다음과 같다. CPU(인텔 펜티엄 3.0GHz), Memory(2GB RAM), OS(MS Windows XP)으로 구성되어 있다.

6.1 서비스 시나리오

본 절에서는 사례연구에서 적용될 HRSS의 시나리오에 대하여 기술하며, 시나리오의 구성은 표 4와 같다. HRSS 시나리오에서 기술된 (1)에서 (6)의 과정은 비즈니스 프로세스 형태로 모델링되어, BPEL Engine에서 실행된다.

그림 15는 HRSS의 시나리오를 적용하여 비즈니스 프로세스로 모델링한 것이다. (1)~(6)은 HRSS의 시나리오의 순서가 비즈니스 프로세스 모델에서 어느 부분(액티비티)에 매핑되는지를 표현한 것이다. (1)은 클라이언트 어플리케이션이 ID/PW를 입력받아 호텔 예약 비지니스 프로세스를 실행시킨다. (2)는 입력된 아이디를 기반으로 사용자의 프로파일을 분석한다. (3)은 호텔 검색 활동(Find Hotel Activity)를 수행함으로써 호텔 검색 서비스가 호출된다. (4)는 검색된 후보 호텔 리스트를 받아서 예약 가능한 후보 호텔을 소비자에게 보여줌으로서 소비자의 입력을 받는다. (5)는 예약할 호텔에 대한 정보를 소비자에게 입력 받아, 예약 호텔 활동(Reserve Hotel Activity)를 수행함으로써, 호텔예약을 수행한다. (6)은 예약된 정보를 서비스 소비자에게 전달하여 결과를 확인할 수 있도록 한다.

표 4 호텔 예약 서비스 시스템의 시나리오

(1) 서비스 소비자는 여행지의 호텔을 예약하기 위해 ID/PW를 입력하여 HRSS에 접속하고, 호텔의 등급, 방 종류, 여행지역, 여행 기간 등 서비스 소비자가 원하는 호텔을 예약하는데 필요한 정보를 입력한다. (2) HRSS는 서비스 소비자의 ID를 기반으로 여행자의 기존의 호텔예약 히스토리와 선호 호텔을 분석한다. (3) HRSS는 서비스 소비자로부터 입력된 정보를 기반으로 호텔을 검색하고 입력된 정보와 일치하는 호텔이 없을 경우, 서비스 소비자의 ID를 기반으로 여행자의 예약 히스토리를 분석하여 선호하는 호텔 리스트를 보여준다. (4) HRSS는 후보 호텔리스트를 보여 주며, 호텔 리스트 중에서 현재 예약 가능한 호텔을 검사하여 호텔 세부 정보를 소비자에게 보여준다. (5)소비자에게 예약할 호텔에 대하여 소비자의 선택을 입력 받아 소비자가 선택한 호텔을 예약한다. (6)예약된 호텔 정보를 호텔 예약 소비자에게 출력한다.

6.2 실험 시나리오 및 서비스 명세

본 절에서는 HRSS 서비스 실행 환경에서 서비스 호출과 관련된 일반적인 경우를 고려하여 실험 시나리오를 작성하고 모니터링 대상이 되는 서비스의 기능성, 배치 등을 명세한다. 실험은 다음과 같은 다섯 가지 과정으로 수행된다.

1. 서비스 사용자가 간단한 입력값과 함께 시뮬레이션 가능한 웹 서비스를 호출한다. 서비스가 실행됨과 동시에 비지니스 프로세스 역시 실행된다. 여기서 서비스는 여러 번 호출할 수 있다.
2. 호출된 시뮬레이션 웹 서비스는 입력값 및 호출된 오퍼레이션 정보 및 호출된 시간을 반환한다. 이 때 모니터링 대상의 내부 및 외부에 모니터링 데이터가 생성된다.
3. 여기서 생성된 모니터링 데이터 중 서비스&비즈니스 프로세스의 내부 모니터링 데이터는 각각 폴링 모델과 푸쉬 모델의 두 가지 상호작용 모델로 모니터링 프레임워크에 전달된다. 단, 환경 관련 데이터와 같은 모니터링 외부 데이터의 경우, 폴링 모델을 이용하여 수집한다.
4. 이벤트를 발생시키는 모니터링 대상은 이벤트 발생의 다양성을 주기 위해 이벤트 발생 수와 발생 간격을

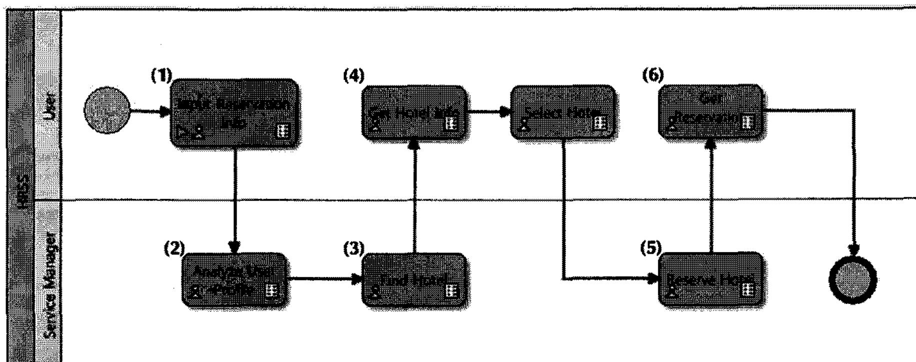

그림 15 HRSS의 비즈니스 프로세스 모델

상이하게 설계한다.

5. 모니터링된 데이터 및 모니터와 관련된 시스템 레벨 데이터를 정보화하여 서비스 모니터링의 상호작용 모델에 따른 효율성을 확인한다.

HRSS 서비스 구현 시 서비스 복잡도에 따른 상호작용 모델과의 연관 관계를 알아보기 위해 다음과 같이 서비스 복잡도를 다르게주어 시뮬레이션 서비스를 구현하였다. 예를 들어, 표 5에서처럼 Service A는 서비스의 복잡도를 위해 Sleep() 매소드를 사용하여 80ms의 시간이 더 소요되도록 서비스를 설계하였다.

표 5 서비스 복잡도

서비스	서비스 복잡도
Hotel Find Service A	Sleep시간: 80ms
Hotel Find Service B	Sleep시간: 40ms
Hotel Find Service C	Sleep시간: 80ms
Hotel Find Service D	Sleep시간: 100ms

HRSS 서비스는 웹서비스이며 WSDM [3] 표준에 따라 모니터링 인터페이스가 구현된 모니터링 가능한 서비스(Monitored Service)로 모니터링 데이터를 얻어오기 위해 *notifyServiceStartupTime(), notifyServiceRunningStatus(String statusValue), notifyServiceTerminateTime()* 등과 같은 Event-Notification 매소드를 모니터링 인터 페이스에 추가 후 그림 16과 같이 서비스안에서 구현하여 서비스 내부 모니터링 데이터의 획득이 가능하게 하였다. 이 데이터는 호출 될 때마다 생성되는 데이터로 기준이 되는 품질속성 모델(QoS Model)과 비교하여 서비스 품질을 평가한다. 이처럼 모니터링 인터페이스를 이용해 구현한 매소드를 호출하면 내부 모니터링 데이터의 수집이 가능하다.

비즈니스 프로세스를 모니터링 할 때 이벤트 메타모델의 요소인 프로세스, 액티비티에 관련된 정보를 필요로 하게 된다. 비즈니스 프로세스 다이어그램은 BPMN과 같은 프로세스 정의 언어로 기술되어 있고, BPMN 프로세스 다이어그램의 저장 및 배치 방식은 WS-BPEL[13] 표준을 따르게 되어 있다. 따라서 WS-BPEL로 기술된 프로세스 실행모델을 모니터링하게 된다. 그림 17은 HRSS 비즈니스 프로세스 다이어그램의 배치 및 실행모델인 BPEL 문서이다. 그림에서 볼 수 있는 것과 같이 BPEL 문서 내부에 앞에서 정의한 이벤트 메타모델 필드를 포함시켜 이벤트의 발생 및 처리를 통해 비즈니스 프로세스 내부 모니터링 데이터의 획득이 가능하게 하였다.

6.3 실험 내용 및 평가

본 절에서는 앞에서 정의한 HRSS 시나리오와 실험 시나리오에 따른 결과를 통해 5장에서 정의한 메트릭을 기준으로 이벤트 주도 서비스 모니터링의 효율성을 평가한다.

1) 소요시간 (Trip Time): 앞절에서 명세한 4개의 서비스(*Hotel Find Service A~D*)에 대하여 각각 10개의 이벤트를 발생시켜 모니터링 프레임워크에서 전달받았다. 이벤트의 발생 간격은 1ms로 고정하였다. 폴링과 푸쉬 상호작용 모델의 소요시간 측정 결과는 그림 18과 같다. 그림에서 볼 수 있는 것과 같이 이벤트 주도의 푸쉬 모델 소요시간이 폴링 모델 소요시간의 50% 미만인 것으로 측정되었다. 따라서, 폴링 모델은 푸쉬 모델에

```
@WebService
public class HotelFindServiceA implements ServiceMonitoringInterface
  ...
  public HotelFindServiceA() {
    notifyServiceRunningStatus("Running");
    ...
  }
  @WebMethod
  public PremiseInfo[] searchPremises(String keyword) {
    attributeList.put("endTimeOfStartupTime", System.currentTimeMillis
    notifyServiceStartupTime();

    PremiseInfo[]primiseList = null;
    ...
    try {
        primiseList = googleMap.getPre
        attributeList.put("startTimeOfSt
      } catch (Exception e) {};

      }

      if (primiseList != null) {
        ...
        attributeList.put("endTimeOfTerm
        notifyServiceTerminateTime();
      }
      ...
      notifyServiceStableStatus("Stable")
      ...
```

```
public void notifyServiceStartupTime() {
  try {
     long startTime=getStartTime("startTimeOfStartupTime");
     long endTime=getEndTime("endTimeOfStartupTime");
     ...
  } catch (RemoteException e) {
     e.printStackTrace();
  }
}
...
public long getStartTime(String startTimeName) {
    if (startTimeName != "") {
      return (Long)attributeList.get(startTimeName);
    }
    else return 0;
...
}
```

모니터링 인터페이스

이벤트 발생

그림 16 모니터링 인터페이스를 구현한 HRSS 서비스

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- BPEL Process Definition -->
<bpws: queryLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116" enableInstanceCompensation="no" abstr
  xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
  <bpws:diagram>
    <bpws:id>ecbc6f98-cd2a-4bc1-80e8-970805b8f685</bpws:id>
    <bpws:version>1.0</bpws:version>
    <bpws:processId>1</bpws:processId>
    <bpws:processName>HRSS</bpws:processName>    ←——— 비즈니스 프로세스 명
  </bpws:diagram>
  ...
    <bpws:variables>
      <bpws:variable name="ID가_1이고_이름이_HRSS_프로세스의_데이타" messageType="ProcessDataMessage_1"/>
    </bpws:variables>
    <bpws:sequence name="HRSS_1_프로세스">          이벤트 아이디와 버전, 메시지 등 정보
      <bpws:empty name="start"/>
      <bpws:scope name="시작_1_시작 이벤트 영역" bpmnId="3">
        <bpws:event>
            <bpws:variables>
              <bpws:variable name="id" messageType="eventField" value = "1" />
              <bpws:variable name="version" messageType="eventField" value = "1.0"/>
              <bpws:variable name="type" messageType="eventField" value = "1" />
              <bpws:variable name="msg" messageType="eventField" value = "initiateMsg"/>
              ...
          <bpws:sequence name="시작_1_시퀀스">
            <bpws:sequence name="시작_1_메세지 시작 이벤트 시퀀스">
              <bpws:assign name="시작_1_인스턴스속성 지정">
              <bpws:copy>
              <bpws:from expression="bpws:getVariableData('ID가_1인_액티비티의_메세지_이름이_initiateMsg
              <bpws:to part="taskId" variable="RegistInsPropVar3"/>
              </bpws:copy>
            </bpws:assign>
              <bpws:invoke operation="registProperties" outputVariable="RegistInsPropReVar3" inputVariabl
```

그림 17 HRSS 비즈니스 프로세스의 BPEL 문서

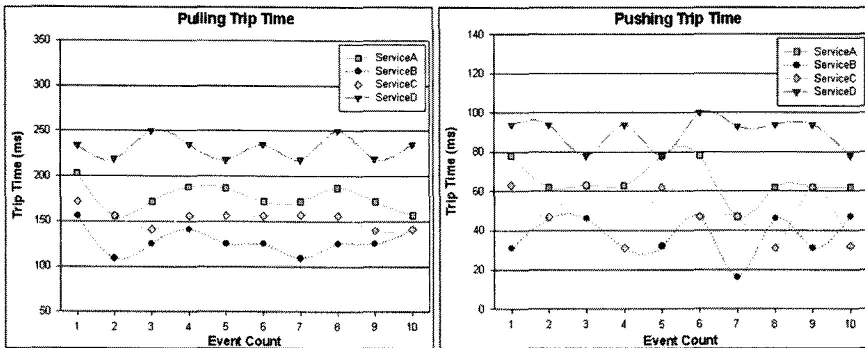

그림 18 풀링과 푸쉬 상호작용 모델의 소요시간

비해 많은 시간을 낭비하며, 의사소통의 과부하가 발생함을 알 수 있다. 결국, 푸쉬 모델을 적용한 이벤트 주도 서비스 모니터링이 더 효율적임을 알 수 있다.

2) 처리량 (Throughput): 처리량은 앞절에서 명세한 *Hotel Find Service*에 대하여 100개의 이벤트를 서로 다른 발생 간격(10ms에서 210ms까지)으로 발생시켜 모니터링 프레임워크가 정해진 시간 동안 얼마나 많은 수의 이벤트를 처리하는 지 측정한다. 측정 결과는 그림 19와 같다. 그림에서 볼 수 있는 것과 같이 푸쉬 모델의 경우 10ms에서 210ms까지의 이벤트 발생 간격을 가진 서비스에 대하여 모니터링 프레임워크가 처리한 이벤트의 수는 100개에서 4.17개이다. 푸쉬 모델에서는 처리량이 이벤트 발생 간격에 크게 의존함을 볼 수 있다. 이에 비해 풀링 모델에서는 5.6개 사이의 이벤트 처리 개수로 비교적 안정적인 모양이다. 따라서, 자주 발생하는 모니터링 데이터의 경우 푸쉬 모델을 적용한 이벤트 주도 서비스 모니터링이 훨씬 더 효율적임을 알 수 있다.

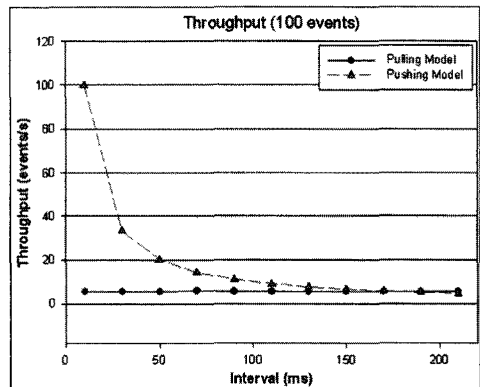

그림 19 풀링과 푸쉬 상호작용 모델의 처리량

7. 평가 및 결론

기존 연구와 비교해 보면, McGregor[4]는 비지니스 프로세스 성능 측정을 위한 솔루션 매니저 서비스 아키텍처를 제안하였다. 제안된 아키텍처는 기밀성, 허가, 데이터 무결성 등 보안 메커니즘을 내장하고 있으며, 이벤트 프로세싱 컨테이너를 통하여 실시간에 많은 수의 프로세스 이벤트들을 처리할 수 있도록 하였다. 하지만 수집되는 이벤트의 구체적인 형태나 수집 방법은 언급하지 않았으며, 모니터링 과부하는 고려되지 않았다. Baresi[5]는 BPEL엔진에 AOP를 사용하여 BPEL 프로세스가 수행되는 동안 주고 받는 데이터를 수집하는 방법과 수집된 이벤트로부터 RTML로 정의된 속성을 확인하는 방법을 제안하였다. 하지만 수집되는 이벤트의 메타모델이나 구체적인 형태는 언급하지 않았으며, BPEL에서 주고 받는 외부 모니터링 데이터를 획득하는 방법을 설명하면서 내부 모니터링 데이터는 언급하지 않았다. 또한, 모니터링 과부하 부분은 BPEL 엔진이 서비스를 호출하는데 영향을 미치지 않는다고 하여 직접적인 언급을 하지 않았다. Lin[6]은 서비스 관리 아키텍처를 제안하고 서비스 버스에 배치된 서비스는 버스에서 제공하는 API를 사용하여 주고 받는 메시지를 획득하는 방법을 제안하였다. 하지만 주고 받는 메시지를 획득하는 방법만을 제안했기 때문에 서비스 내부 모니터링 데이터를 획득하는 방법에 대한 언급은 없으며, 모니터링 시 발생하는 과부하는 일부만 언급하였다.

본 논문에서 제안한 기법을 평가하기 위하여 다음과 같은 평가항목을 선정하였다. 현재 연구 결과에 의하면 다음과 같은 항목들이 서비스 모니터링 기능으로 제공될 필요가 있다[5,6].

- 모니터링 대상 별 특성 및 적용 기법 제공: 모니터링 대상을 분류하고 그 특성에 적합한 상호작용 모델 등 모니터링 기법을 제공하는지에 대한 평가
- 이벤트 및 이벤트 처리 메타모델 제안: 이벤트 주도 모니터링의 핵심인 이벤트와 이벤트 처리에 대한 모델을 제안하는지에 대한 평가
- 모니터링 과부하 고려: 의사소통 과부하 및 모니터링 프레임워크의 과부하를 고려하는지에 대한 평가
- SOA 표준 준수: WS-BPEL, WSDM, WSDL 등 SOA 표준을 준수하는지에 대한 평가
- 내부 모니터링 데이터 획득: 미들웨어의 API를 이용한 제한된 모니터링 데이터를 해결하기 위해 서비스 내부 모니터링 데이터를 획득할 수 있는 기법이 필요
- 구현 및 상세지침 제공: 제안된 기법을 수행하기 위한 세부적인 기준과 지침, 구현 프로토타입이 제공되는지를 평가

이상의 평가 항목으로 표 6에서와 같이 기존 연구와 본 기법을 비교해 보았다.

SOA에서의 서비스는 블랙 박스 형태로 제공되며, 실시간에 진화될 수 있으므로 동적으로 서비스의 다양한 측면을 효과적, 효율적으로 모니터링하는 기능의 필요성이 증대되고 있다. 특히, 서비스 품질 관리를 위하여 서비스 내부 모니터링 데이터가 필요하지만 현재 모니터링 기법들은 미들웨어나 BPEL엔진의 API에 의존한 제한된 모니터링 데이터만의 획득으로 한계가 있으며, SOA의 전통적인 상호작용 모델인 폴링 모델과 요청/응답의 의사소통 패턴 등으로 모니터링 과부하가 발생하는 등의 문제가 있다.

본 논문은 SOA환경에서 서비스와 비즈니스 프로세스 그리고 환경 관련 데이터를 효율적으로 모니터링하기 위한 방안에 대한 연구로써 EDA의 비동기식(Asynchronous), 발행 및 구독(Publish-and-Subscribe)등의 상호작용 방식을 모니터링 대상에 맞게 적용하여 모니터링의 과부하를 줄이고, 소요시간을 절약하는 등의 효과를 추구하였다. 또한, OASIS WSDM Event Format(WEF) 기반의 이벤트 메타모델과 처리 모델을 제시함으로써 이식성과 확장성을 추구하였고, 미들웨어나 BPEL 엔진의 API를 통한 외부 모니터링 데이터만의 획득이 아닌 내부 모니터링 데이터의 획득을 위한 기법을 제안하였다. 그리고, 획득한 데이터의 효과적, 효율적인 분석을 위한 이벤트 필터와 이벤트 체인, 이벤트 매니저 등의 주요 컴포넌트를 상세히 설계하고, 이를 프로토타입으로 구현하였다. 그리고, 서비스 모니터링의 효율성을 평가하기 위한 메트릭을 소요시간과 처리량 측면에서 정의하였고,

표 6 기존 연구와의 비교 평가

○ : 지원, △ : 부분 지원

평가 항목 \ 대상	McGregor [4]	Baresi [5]	Lin [6]	본 논문의 기법
모니터링 대상 별 특성 및 적용 기법 제공				○
이벤트 및 이벤트 처리 메타모델 제안	△	△		○
모니터링 과부하 고려			△	○
SOA 표준 준수		○	○	○
내부 모니터링 데이터 획득				○
구현 및 상세지침 제공		△		○

이를 적용하기 위하여, 시뮬레이션할 수 있는 HRSS 서비스를 구현하였다. 그런 다음 구현한 서비스를 제안한 시나리오에 따라 비즈니스 프로세스로 모델링하고, 실험 시나리오에 따른 실험을 통하여 실험 결과를 얻었다. 앞에서 정의한 메트릭을 기준으로 이 실험 결과를 분석하였으며, 그에 따라 이벤트 주도 서비스 모니터링의 효율성을 확인할 수 있었다. 이처럼 본 논문에서 제안한 아키텍처와 기법 및 지침은 효율적인 서비스 모니터링을 위한 실용적인 접근이 될 수 있을 것이다.

참 고 문 헌

[1] K. M. Chandy, "Event-Driven Applications: Costs, Benefits and Design Approaches," *California Institute of Technology*, 2006.

[2] K. M. Chandy, S. Ramo and W. R. Schulte, "What is Event Driven Architecture (EDA) and Why Does it Matter?," *Gartner Inc.*, 2007.

[3] OASIS, Web Services Distributed Management: Management Using Web Services (MUWS 1.0) Part 1 and Web Services Distributed Management: Management Using Web Services (MUWS 1.0) Part 2, 9 March 2005.

[4] McGregor, C., and Schiefer, J., "A web-Service based framework for analyzing and measuring business performance," *Information Systems and e-Business Management*, vol.2, no.1, pp.89-110, Springer, 2004.

[5] Baresi, L., Guinea, S., Pistore, M., Trainotti, M., "Dynamo + Astro: An Integrated Approach for BPEL Monitoring," *IEEE International Conference on Web Services* (ICWS 2009), pp.230-237, 2009.

[6] Lin, K., Panahi, N., Zhang, Y., Chang, S., "Building Accountability Middleware to Support Dependable SOA," *IEEE Internet Computing*, vol.13, no.12, 2009.

[7] Gartner Inc., "Event-Driven Architecture Complements SOA," http://www.gartner.com/Display Document?doc_cd=116081.

[8] J.P. Martin-Flatin, "Pushing vs. Pulling in Web-Based Network Management," *6th IFIP/IEEE International Symposium on Integrated Network Management (IM '99)*, Boston, USA, May 1999.

[9] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R, Nord, R., and Stafford, J., Documenting Software Architectures: Views and Beyond, Addison Wesley, 2002.

[10] Erl, T., Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall, 2005.

[11] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, Design, *Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994.

[12] Booth, D. and Kevin, C. eds., *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*, W3C Recommendation, W3C, 26 June, 2007, http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/ (accessed September 26, 2007).

[13] OASIS, Web Services Business Process Execution Language Version 2.0, Public Review Draft, 23rd August, 2006.

[14] K. Cwalina and B. Abrams, *Framework Design Guidelines: Conventions, Idioms, and Patterns for Reuseable.NET Libraries*, Addison Wesley Professional, 2008.

[15] Chappell, D.A., Enterprise Service Bus, O'Reilly, 2004.

[16] Apache ODE, http://ode.apache.org/index.html

[17] Apache ServiceMix, http://servicemix.apache.org/home.html

[18] PostgreSQL, http://www.postgresql.org/

금 득 규

1995년 강남대학교 산업공학과 공학사 2005년 숭실대학교 정보통신학과 공학석사. 2006년~현재 숭실대학교 컴퓨터학과 박사과정. 현재 동서울대학 겸임교수 관심분야는 서비스지향 아키텍처(SOA), 클라우드 컴퓨팅(Cloud Computing), 모바일 서비스(Mobile Service)

김 수 동

정보과학회논문지 : 소프트웨어 및 응용 제 37 권 제 7 호 참조