효율적인 서비스 모니터링 프레임워크 및 전송패턴

(A Framework and Patterns for Efficient Service Monitoring)

이 현 민 † 천 두 완 † 김 수 동 ††
(Hyun Min Lee) (Du Wan Cheun) (Soo Dong Kim)

요 약 서비스 지향 컴퓨팅은 서비스를 동적으로 조합하여 사용자가 요구하는 비즈니스 프로세스를 개발하는 재사용 패러다임이다. 서비스 소비자는 서비스 제공자가 제공한 인터페이스만을 통하여 서비스의 기능성을 사용한다. 즉, 서비스의 내부 상태는 블랙박스 형태로 알 수 없기 때문에 서비스 소비자는 서비스의 품질을 알 수 없고 이는 중요한 도메인에 서비스의 도입을 어렵게 만든다. 따라서 서비스 모니터링에 대한 필요성이 증가하고 있다. 하지만 서비스 내부의 정보를 실시간으로 획득하고 품질을 측정하기 어렵기 때문에 현재는 각 벤더들이 제공하는 특정 제품을 이용하여 서비스와의 상호작용 없이 모니터링이 이루어지고 있다. 하지만 이는 데이터의 포괄성, 정확성 측면에서 한계가 있고 모니터링으로 발생하는 오버헤드를 최소화 시키기 위한 효율성을 고려하지 않았기 때문에 이를 해결하기 위한 모니터링 프레임워크가 요구된다. 본 논문에서는 효율적인 서비스 모니터링을 위한 모니터링 프레임워크를 제시한다. 먼저 모니터링 프레임워크 설계 요구사항을 정의한다. 정의된 요구사항을 기반으로 모니터링 프레임워크 아키텍처를 제시하고 모니터링 데이터를 효율적으로 전송하기 위한 패턴을 제시한다. 또한 모니터링 프레임워크의 설계 및 구현을 보여주고 실험을 통해 모니터링 프레임워크의 효율성을 보여준다.

키워드 : 모니터링 프레임워크, 서비스 모니터링 효율성, 데이터 전송패턴, 서비스 지향 컴퓨팅

Abstract Service-Oriented Computing (SOC) is a reuse paradigm for developing business processes by dynamic service composition. Service consumers subscribe services deployed by service providers only through service interfaces. Therefore, services on server-side are perceived as black box to service consumers. Due to this nature of services, service consumers have limited knowledge on the quality of services. This limits utilizing of services in critical domains hard. Therefore, there is an increasing demand for effective methods for monitoring services. Current monitoring techniques generally depend on specific vendor's middleware without direct access to services due to the technical hardship of monitoring. However, these approaches have limitations including low data comprehensibility and data accuracy. And, this results in a demand for effective service monitoring framework. In this paper, we propose a framework for efficiently monitoring services. We first define requirements for designing monitoring framework. Based on the requirements, we propose architecture for monitoring framework and define generic patterns for efficiently acquiring monitored data from services. We present the detailed design of monitoring framework and its implementation. We finally implement a prototype of the monitor, and present the functionality of the framework as well as the results of experiments to verify efficiency of patterns for transmitting monitoring data.

Key words : Service Monitoring Framework, Efficient Monitoring, Patterns for data transmission, Service Oriented Computing

· 이 논문은 2009년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2009-0076392)

† 학생회원 : 숭실대학교 컴퓨터학부
cyclro@gmail.com
raphael.cheun@gmail.com
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
sdkim777@gmail.com

논문접수 : 2010년 8월 30일
심사완료 : 2010년 9월 16일

정보과학회논문지: 소프트웨어 및 응용 제37권 제11호(2010.11)

1. 서 론

서비스 지향 컴퓨팅(Service-oriented Computing, SOC)은 재사용 가능한 서비스를 활용하여 비즈니스 프로세스나 어플리케이션을 경제적으로 개발하는 재사용 패러다임이다[1]. 서비스 제공자는 서비스를 개발하고, 개발한 서비스의 인터페이스를 WSDL(Web Service Definition Language)로 기술하여, 서비스 레지스트리 UDDI (Universal Description, Discovery and Integration)에 배포한다. 서비스 소비자는 UDDI에 등록된 서비스 중 필요한 기능성을 제공하는 서비스를 검색, 선택, 조합하여 새로운 서비스를 개발한다.

서비스 소비자는 서비스 제공자가 제공한 인터페이스만을 통하여 서비스의 기능성을 사용하며, 서비스의 내부 상태는 블랙박스 형태의 형태로 알 수 없다. 서비스 소비자는 서비스의 품질을 알 수 없고, 서비스에 문제가 발생한 경우 서비스 소비자가 사용하는 서비스도 연쇄적으로 문제가 발생하게 된다. 서비스 신뢰성의 문제로 인하여 SOC를 중요한 영역에 도입하기에는 아직 무리가 있다.

서비스 제공자 및 관리자는 서비스의 품질 관리를 위해 서비스의 상태를 알아야 하고 서비스 소비자는 자신이 운영하는 서비스를 관리하기 위해 서비스의 품질을 알아야 한다. 서비스의 품질 측정과 서비스의 관리를 통하여 서비스의 품질을 보장하여야만 서비스 소비자와의 SLA (Service Level Agreement)를 지킬 수 있다[2].

서비스 모니터링은 서비스 관리의 첫 단계로서, 서비스의 품질을 측정하고 나아가서는 서비스의 품질적인 결함 진단을 하기 위해 데이터를 수집하는 일련의 활동이다. 서비스 모니터링의 목적은 1) 서비스의 품질이 SLA에 기술되어 있는 품질 요구사항을 만족하는지 알아내고, 2) 서비스 운영 시 발생할 수 있는 품질적인 결함을 찾는 데 필요한 기반 정보를 제공하는 데에 있다. 즉, 서비스 모니터링은 모니터링 데이터를 획득하는 것뿐 아니라 획득한 데이터를 이용하여 현재 서비스 상태를 표현하는 것까지의 행위를 말한다.

서비스는 블랙박스 형태로 제공되기 때문에 서비스 내부 정보를 제공하지 않는다. 실시간으로 데이터를 수집, 조치해야 하기 때문에 모니터링 데이터는 런타임에 획득해야 한다. 그리고 모니터링에 필요한 인터페이스 및 API에 대한 표준이 없다. 이러한 이유로 인하여 지금까지 서비스 모니터링은 BPEL(Business Process Execution Language) 엔진이나 서비스 서버 혹은 미들웨어를 중심으로 연구되고 있다. 서비스의 환경에서 획득할 수 있는 모니터링 데이터는 데이터의 포괄성, 데이터의 정확성 측면에서 모두 한계가 있다.

따라서 품질 모델에서 요구하는 모니터링 데이터를 실시간으로 효율적으로 모니터링 할 수 있는 방법이 요구되고 있다. 모니터링 프레임워크는 각 서비스에 대한 품질 모델을 관리하고 품질 모델에 따라 측정해야 할 모니터링 데이터를 선정하여야 한다. 또한 각 서비스 별로 품질 모델을 이용하여 QoS를 계산, 서비스의 현재 상태를 나타내어야 한다. 또한 모니터링으로 인해 발생되는 오버헤드가 서비스 자체의 성능을 저하 시킬 수 있기 때문에 서비스 모니터링 효율성을 고려해야 한다.

본 논문에서는 앞에서 나온 요구사항을 참고하여 모니터링 프레임워크 설계 요구사항을 추출한다. 4장에서는 3장에서 도출한 요구사항을 바탕으로 모니터링 프레임워크의 아키텍처를 제시하고 이를 효율적으로 적용하기 위한 모니터링 데이터 전송 패턴을 제시한다. 5장에서는 앞장에서 제시한 아키텍처와 전송 패턴을 기반으로 모니터링 프레임워크의 설계와 구현을 보여준다. 6장의 실험에서는 구현된 모니터링 프레임워크를 이용하여 전송패턴의 효율성을 측정하는 실험을 한다.

제시된 모니터링 프레임워크는 그 동안 제공되지 않았던 서비스의 실시간 QoS 정보를 효율적으로 알 수 있게 해준다. 이를 활용하여 앞으로 서비스 모니터링 프레임워크 설계에 도움이 될 뿐 아니라 서비스 관리자들이 서비스 관리 작업을 수행하는 데 도움이 되길 기대한다.

2. 관련연구

서비스 모니터링에 관한 연구는 지금까지 서비스 관리의 다른 영역에 비해 비교적 많이 다루어지지 않았다. 서비스의 품질 측정 및 관리를 위해서는 서비스 모니터링이 필요하다는 인식은 공통적으로 가지고 있으나 모니터링 방법이 제한적이기 때문이다. 서비스의 품질 측정을 위해서는 필요한 데이터를 획득해야 한다. 소비자는 서비스 품질 정보를 이용하여 서비스를 검색하고 결합한다. 같은 기능을 가진 서비스의 선택에 있어서 서비스 품질 정보는 중요한 요소이다. 또한 모니터링은 모니터링 없는 시스템 보에 모니터링 오버헤드가 추가되므로 이에 따라 서비스 모니터링은 효율적인 방법으로 이루어 져야 한다. 이 장에서는 모니터링 할 데이터, 서비스 모니터링 기법, 서비스 모니터링 효율성의 세가지 측면에서의 관련 연구를 살펴보도록 한다.

먼저 모니터링할 데이터에 관해서 살펴보면 다음과 같은 문제점이 있다. 기존에 나온 서비스 품질 모델이 서로 다르고 아직까지 서비스 품질 모델에 대한 표준이 나와있지 않다[3-5]. 아직까지 서비스 품질 모델에 대한 정의가 되어 있지 않기 때문에 서비스 품질 측정을 위해 필요한 데이터 또한 정의되지 않았다. 지금까지 서비

스 모니터링 연구가 BPEL 엔진이나 미들웨어를 이용한 방법을 사용했다. 즉, 기존의 여러 품질 모델에 공통적으로 들어가 있는 시간, 가용성, 신뢰성, 보안성 등의 품질 매트릭을 계산하기 위하여 JMX(Java Extension Management) API를 사용하여 데이터를 수집하였다. 이는 JMX등의 외부 환경을 통해 얻어 올 수 있는 데이터로 이루어진 매트릭 만으로 품질 모델로 한정 지을 수 밖에 없었다. 즉, 외부 환경에 의존하는 모니터링으로 수집 가능한 데이터가 먼저 한정되었기 때문에 품질 모델의 범위를 제한 시켰다.

둘째로 서비스 모니터링 기법 측면에서 살펴보면 품질 모델 측정을 위한 더 많은 데이터를 얻어오기 위한 방법 보다는 모니터링 프레임워크를 활용하여 서비스 모니터링의 효율성을 높이기 위한 시도가 있었다. Baresi의 연구에서는 기존에 제시한 모니터링 기법 Dynamo와 Astro의 통합한 모니터링 프레임워크를 제안하였다[6]. Dynamo는 관점지향 프로그래밍(Aspect-Oriented Programming)을 사용하여 BPEL 프로세스가 동작하는 동안 BEPL 엔진이 수집할 수 있는 모니터링 데이터를 얻어 오는 방법이다[7]. Astro는 독립된 소프트웨어 모듈을 사용하여 RTML(run-time monitor specification Language)로 정의된 속성을 확인하는 방법이다[8]. 이 논문에서는 두 가지 방법을 통합하여 BPEL 동작 시 더욱 정교한 모니터링 결과를 얻을 수 있는 방법을 제시한다. Lin의 연구에서는 ESB를 확장시킨 미들웨어를 제안하였다[9]. 이 미들웨어는 다음과 같이 구성된다. 즉, Llama는 서비스 모니터링을 위한 컴포넌트, 서비스의 결함을 알기 위한 컴포넌트, 서비스 결함을 진단하는 컴포넌트이다. 여기서 서비스의 실행시간을 모니터링 하기 위한 방법으로 ESB에서 제공하는 모니터링 API나 추가한 컴포넌트인 Profiling Interceptors를 사용하여 모니터링 데이터를 얻어오는 방법을 제시한다.

위의 두 연구에서는 서비스와의 상호작용 없이 미들웨어에 의존하여 모니터링 데이터를 얻어 오는 방법만을 설명하고 있다. 그 이유는 다음과 같다. 서비스는 블랙박스 형태로 제공되기 때문에 서비스 내부의 정보를 수집하기 위해서는 자신의 내부 정보를 모니터에 제공해 주는 기능이 포함되어 있어야 한다. 이는 서비스가 수행되는 자원과 서비스가 관리되는 자원이 동시에 사용되어 서비스에 많은 오버헤드를 줄 수 있음을 의미한다. 또한 서비스 자체에서 발생하는 오버헤드는 모니터링 프레임워크를 통해 해결하기가 힘들기 때문에 서비스에 모니터링 기능을 추가시키는 방법은 거의 다루어 지지 않았다.

셋째로 서비스를 모니터링으로 인해 발생하는 오버헤드 문제는 기존의 여러 논문들에서 제기되어 왔다. 하지만 서비스 모니터링 시 활용되는 모니터링 기법 및 도구의 효율성을 평가 할 기준 및 매트릭이 정의되지 않아 정량적인 비교 보다는 정성적인 비교를 통한 평가만이 이루어 졌다. 또한 모니터링 시 오버헤드 문제는 인식하고 있으나 오버헤드를 해결하기 위한 상황 별 아키텍처 설계 방법에 대한 연구는 거의 다루어지지 않았다.

3. 모니터링 프레임워크 설계 요구사항

3.1 기능적 요구사항

서비스 모니터링을 하기 위한 프레임워크를 설계는 다음과 같이 크게 3가지의 기능적 요구사항을 만족시켜야 한다.

모니터링 데이터 획득: 서비스 소비자는 서비스의 인터페이스를 이용하여 서비스의 기능을 사용한다. 서비스는 블랙박스처럼 인식되므로 서비스 사용자와 관리자는 서비스에 대해 제한된 가시성과 관리성을 가지게 된다. 또한, 서비스는 서비스 변경에 대한 알림 없이 진화할 수 있고, 존재하는 서비스가 순간적으로 혹은 영속적으로 사라질 수도 있다. 이러한 일련의 정보는 서비스 내부의 정보 없이는 알 수 없다. 따라서 서비스 관리 및 품질 측정을 위해서 모니터링 프레임워크는 서비스의 모니터링 정보를 획득하여야 한다.

QoS 실시간 계산: 측정된 모니터링 데이터가 유효할 수 있도록 모니터링 프레임워크는 런타임 시에 모니터링 데이터를 획득하여 실시간으로 QoS를 계산하여야 한다. QoS 실시간 계산은 시간 유효성과 관계가 있다. 시간 유효성이란 서비스가 실행되어 나온 관리 정보가 유효하도록 모니터링 정보를 정해진 시간 안에 서비스를 모니터링 하는 컴포넌트에 전달하는 것을 의미한다. 서비스를 모니터링하는 컴포넌트가 원하는 모니터링 데이터의 전송이 일정 시간 이상 걸리면 의미없는 정보가 될 수 있기 때문에 결함 진단을 위해서는 실행된 서비스에 대한 모니터링 데이터가 유효한 시간 안에 전송되어야 한다. 유효한 모니터링 데이터를 기반으로 모니터링 프레임워크는 현재 서비스의 QoS를 실시간으로 계산하여야 한다.

QoS의 시각화: 서비스 모니터링은 모니터링 데이터를 획득하는 것뿐 아니라 획득한 데이터를 이용하여 현재 서비스 상태를 표현하는 것까지의 행위를 말한다. 서비스 관리자에게 효과적으로 서비스의 현재 QoS를 전달하기 위해서 모니터링 프레임워크는 실시간으로 계산되는 QoS를 효과적으로 시각화시켜야 한다.

3.2 비기능적인 요구사항

서비스 모니터링을 하기 위한 프레임워크를 설계할 때 효율성과 표준화를 고려한다. 일반적으로 효율성은 얼마나 많은 자원을 소비했는지를 측정하여 표현할 수

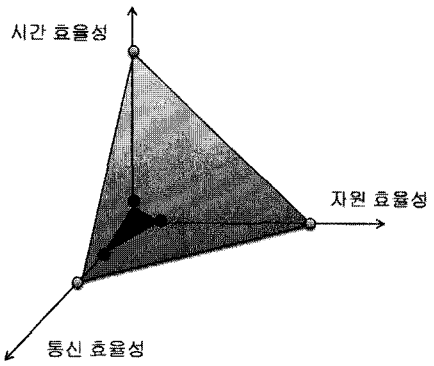

그림 1 모니터링 효율성 요구사항

있다[10]. 먼저, 서비스를 모니터링하면 서비스를 호출하는데 필요한 노력 및 자원이 추가적으로 소요된다. 이로 인하여 서비스의 품질 저하가 발생할 수 있다. 그러므로 모니터링과 관련된 기법 및 도구 설계 시 이런 서비스의 품질 저하가 최소가 될 수 있도록 모니터링 효율성을 고려해야 한다. 모니터링 효율성을 위해 고려해야 할 사항은 시간, 통신, 자원이다. 그림 1에서 보면 큰 붉은색 삼각형과 작은 푸른색 삼각형이 있고 큰 삼각형은 작은 삼각형을 완전히 포함 하고 있다. 이 의미는 큰 삼각형으로 평가된 모니터링 기법, 혹은 도구가 작은 삼각형으로 평가된 기법, 도구 보다 세 가지 기준에서, 즉 시간, 통신, 자원 측면에서 더 효율적이라는 뜻이다.

시간 효율성: 시간 효율성은 모니터링으로 인해 추가적으로 소요되는 시간이 최소화되어야 함을 의미한다. 모니터링 데이터를 가져오려면, 서비스가 모니터링 인터페이스를 구현하여야 하고, 이 인터페이스를 구현하여 사용하게 되면, 기능 호출 시 추가적으로 소요되는 시간이 발생한다. 즉, 이런 오버헤드는 모니터링 관련된 기능을 수행하기 위한 연산으로 인해 발생한다. 그러므로 오버헤드를 최소화하기 위하여 모니터링 프레임워크의 시간 효율성을 높여야 한다.

통신 효율성: 모니터 데이터를 가져오기 위해서 모니터링 프레임워크와 서비스 간의 통신 비용이 발생한다. 이런 통신 비용을 확인하는데 필요한 요소로는 모니터링 프레임워크와 서비스 간에 주고 받는 메시지 개수이다. 네트워크를 이용하여 주고 받는 메시지가 많을수록 메시지를 처리하는 시간 및 자원, 메시지를 전송하는데 걸리는 시간 및 자원 등이 추가적으로 소요된다. 그러므로 이런 부분을 최소화하기 위하여 모니터링 프레임워크의 통신 방법이 효율적으로 설계되어야 한다.

자원 효율성: 자원 효율성은 모니터링을 수행하기 위해 사용되는 자원의 효율적인 사용을 의미한다. 서비스 모니터링 데이터를 처리, 저장하는 방법에 따라 소비되는 자원이 달라질 수 있다. 이런 자원의 가용성에 따라 서비스의 가용성 및 확장성에 영향을 끼칠 수 있으므로 중요한 요소이다. 자원 효율성에서 고려할 자원은 CPU 사용량, 네트워크 사용량, 메모리 사용량 등이다.

두 번째로는 서비스 모니터링을 위한 표준이다. 서비스는 여러 서비스 제공자가 개발하기 때문에 관련 모니터링 정보에 접근할 수 있는 인터페이스를 제공할 때 표준화된 방법으로 제공하여야 한다. 즉, 서비스 인터페이스에서 사용하는 네임스페이스, 모니터링 인터페이스를 위한 스키마 등이 일관적으로 개발될 수 있도록 기반을 제공하여야 한다. 또한, 모니터링 프레임워크가 표준화된 방법을 통해서 모니터링 정보를 가져올 수 있도록 통신 매카니즘이 설계되어야 한다.

4. 모니터링 프레임워크 아키텍처와 전송패턴

본 장에서는 3장에서 도출한 서비스 모니터링 요구사항을 기반으로 데이터를 효율적으로 수집하기 위한 모니터링 프레임워크의 아키텍처를 제시한다. 또한 모니터링의 효율성을 높이기 위한 서비스 모니터링 데이터 전송 패턴을 제시한다.

4.1 모니터링 프레임워크 아키텍처

일반적으로 아키텍처는 시스템의 비기능적 요구사항을 만족시키기 위해 설계된다[11]. 그림 2는 프레임워크의 배치 뷰를 보여준다. 모니터링 프레임워크는 모니터링 에이전트, 모니터 매니저로 구성되어 있다. 모니터링 에이전트는 네트워크를 통해 서비스의 모니터링 데이터를 수집한다. 모니터링 에이전트와 모니터 가능한 서비스는 등록과정을 통해 모니터링 에이전트 - 모니터 가능한 서비스 관계가 형성되고 이는 각각 다대다의 형태로 이루어 진다. 모니터 매니저는 모니터링 에이전트가 수집한 데이터를 전송 받아 QoS를 계산하고, 서비스 관리자와의 인터페이스 역할을 한다. 여기 제시된 모니터링 프레임워크 아키텍처에서는 모니터 매니저가 모니터 가능한 서비스와 직접 메시지 및 데이터를 주고 받지 않는다. 이는 다음과 같은 이점을 제공한다. 먼저 서비

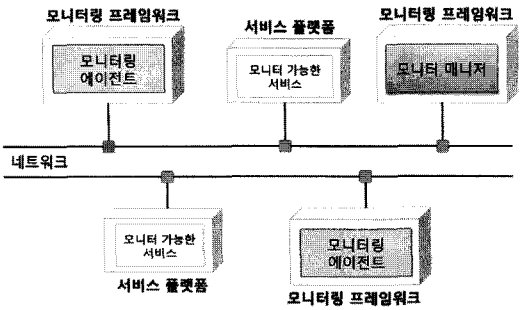

그림 2 모니터링 프레임워크 배치 뷰

스는 동적인 특성을 가지고 있다. 이러한 서비스를 주기적으로 체크하고 관리하는 작업은 엄청난 부하를 일으킨다. 서비스와의 상호작용을 모니터링 에이전트만 가능하도록 함으로써 모니터링 프레임워크는 서비스에 관계없이 구조 변경이 가능하고 또한 이를 통해 효율적으로 서비스를 관리할 수 있다. 또한 근접한 모니터링 에이전트가 서비스를 관리하게 되어 시간 효율성을 증대 시킬 수 있다.

여기서 모니터링의 대상인 모니터 가능한 서비스는 다음과 같다. 모니터링 가능한 서비스는 수동적인 모니터링이 가능한 서비스와 능동적인 모니터링이 가능한 서비스로 구성되어 있다[12]. 수동적인 모니터링은 모니터 에이전트가 모니터링의 대상인 서비스에 모니터링 데이터를 요청하여 수집하는 것이다. 모니터링 데이터를 수동적으로 수집하기 위해서 서비스는 모니터 에이전트가 접근할 수 있도록 모니터링 인터페이스를 제공해야 한다. 능동적인 모니터링은 모니터링의 대상인 서비스가 모니터 에이전트에 모니터링 데이터를 전송하는 것이다. 서비스가 능동적으로 모니터 에이전트에 모니터링 데이터를 전송하기 위해서는 서비스가 데이터를 전송할 수 있는 기능을 가지고 있어야 한다. 이를 위해 서비스는 에이전트 정보관리 모듈, 스케줄러, 데이터 전송 모듈 같은 컴포넌트로 구성되어 있다. 모니터링 가능한 서비스와 모니터링 에이전트는 표준화된 인터페이스를 통해 데이터를 전송한다. 이를 통해 모니터링 프레임워크는 다수의 서비스 제공자 혹은 다른 플랫폼에서도 사용 가능하다.

그림 3은 모니터링 프레임워크의 전체적인 아키텍처 구조를 보여준다. 모니터링 프레임워크는 모니터링 계층, 데이터 처리 계층, 표현 계층, 저장 계층, 이렇게 총 4개의 계층으로 나누어 진다. 모니터링 에이전트는 모니터 가능한 서비스와 통신이 이루어지는 모니터링 계층과 저장 계층으로 이루어져 있다. 모니터 매니저는 서비스 관리자와의 인터페이스 역할을 하는 표현 계층과 관리자 및 모니터링 계층의 제어를 담당하는 데이터 처리 계층과 저장 계층으로 이루어져 있다. 저장 계층은 모니

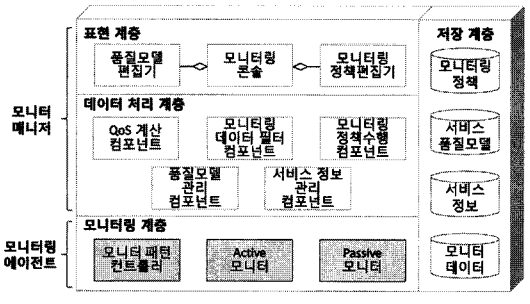

그림 3 모니터링 프레임워크 기능 뷰

터 매니저와 모니터링 에이전트에 나누어져 저장되어 있다. 이렇게 분리된 계층은 다음과 같은 이점을 제공한다. 먼저 서비스 관리자에게 관심에 따라 다양한 뷰를 제공한다. 또한 모니터링 계층은 서비스와의 통신을 한다. 모니터 마스터는 서비스를 인식하고 있을 필요가 없이 모니터링 에이전트가 제공하는 데이터를 처리하면 되기 때문에 자원소비를 줄일 수 있다.

4가지 계층을 각각 살펴보면 다음과 같다.

모니터링 계층은 모니터 패턴 컨트롤러 Active 모니터, Passive 모니터의 총 3개의 컴포넌트로 구성되어 있다. 모니터링 계층은 모니터 가능한 서비스와 직접적인 통신이 이루어 지는 계층이다. 모니터 패턴 컨트롤러는 서비스 품질 모델에서 필요한 모니터링 데이터를 수집하기 위한 패턴을 제어하는 컴포넌트이다. 이는 4.2절에서 설명할 모니터링 데이터 전송 패턴을 모두 지원하며 서비스 품질 모델 정보를 활용하여 모니터 가능한 서비스의 데이터 전송 모듈을 함께 제어한다. Active 모니터와 Passive 모니터 컴포넌트는 직접 모니터링 가능한 서비스와 모니터링 데이터를 주고 받는다. Active 모니터는 등록되어 있는 서비스의 엔드포인트(Endpoint)를 이용하여 모니터링 가능한 서비스의 모니터링 인터페이스에 접근, 모니터링 데이터를 수집한다. 동적으로 변화하는 서비스를 모니터링 하기 위해 모니터링 인터페이스에는 표준화 되어 있다. Passive 모니터는 모니터 가능한 서비스에서 보내오는 모니터링 데이터 및 이벤트를 수신하는 컴포넌트이다.

데이터 처리 계층은 QoS 계산 컴포넌트, 모니터링 데이터 필터 컴포넌트, 모니터링 정책 수행 컴포넌트, 품질 모델 관리 컴포넌트, 서비스 정보 관리 컴포넌트의 총 5개의 컴포넌트로 구성되어 있다. 데이터 처리 계층은 모니터링 계층에서 수집한 데이터와 저장 계층에 저장되어 있는 데이터를 조작하고 저장 계층에 다시 저장하는 역할을 수행하는 계층이다. QoS 계산 컴포넌트는 현재 수집하고 있는 모니터링 데이터의 품질을 계산하여 서비스의 품질모델에 정의된 품질과 비교하여 서비스 정보를 저장한다. 모니터링 데이터 필터 컴포넌트는 Active 모니터, Passive 모니터에서 수집한 데이터의 유효성을 판단하여 저장 계층의 모니터 데이터에 저장한다. 이를 통하여 모니터링 프레임워크는 자원효율성을 높일 수 있다. 또한 Passive 모니터에서 수집한 이벤트를 판단하여 Active 모니터에서 필요한 데이터를 요청하도록 한다. 모니터링 정책 수행 컴포넌트는 저장 계층의 모니터링 정책 데이터와 서비스 정보를 이용하여 Active 모니터에서 서비스에 접근할 수 있는 엔드포인트와 시간을 제공한다. 품질 모델 관리 컴포넌트와 서비스 정보 관리 컴포넌트는 저장 계층의 서비스 정보와

서비스 품질 모델 데이터를 제어하는 역할을 한다.

표현 계층은 모니터링 콘솔로 이루어져 있고, 모니터링 콘솔은 품질 모델 편집기와 모니터링 정책 편집기로 이루어져 있다. 표현 계층은 서비스 관리자와의 상호작용이 이루어지는 계층이다. 모니터링 콘솔에서는 서비스 정보를 서비스 관리자에게 다양한 시점으로 보여준다. 또한 서비스 관리자는 품질 모델 편집기와 모니터링 정책 편집기를 사용하여 서비스 관리, 품질 모델 관리, 모니터링 정책 관리를 수행한다.

저장 계층은 모니터링 정책, 서비스 품질모델, 서비스 정보, 모니터 데이터로 구성되어 있다. 저장 계층은 표현 계층, 데이터 처리 계층, 모니터링 계층에 데이터를 제공하고, 저장 계층의 데이터는 단지 데이터 처리 계층에 의해서만 변경 될 수 있다. 모니터링 정책은 모니터 매니저가 관리하며 이는 모니터링 에이전트에 전달되어 서비스 모니터링 정책을 반영한다. 서비스 품질 모델은 모니터 매니저에 저장되어 있다. 모니터링 데이터는 모니터링 에이전트는 전송된 모든 데이터를 저장하고 모니터 매니저는 이를 모니터링 데이터 필터 컴포넌트를 이용하여 선별적으로 저장한다. 서비스 정보는 모니터 매니저가 가지고 있고 모니터링 에이전트는 모니터링 하는 서비스의 리스트를 관리하고 있다.

그림 4는 모니터링 프레임워크의 동적 모델이다. 서비스 관리자가 모니터링 정책을 결정하면 정책 데이터는 모니터링 정책 저장소에 저장하고 이는 모니터링 정책 수행 컴포넌트에 전달된다. 모니터링 정책 수행 컴포넌트는 모니터링 패턴을 결정하여 이를 각 모니터링 에이전트에 알린다. 모니터링 에이전트는 등록되어 있는 모니터 가능한 서비스에 이를 전달한다. 모니터 가능한 서비스와 모니터링 에이전트는 결정된 정책대로 모니터링을 시작한다. 모니터링 에이전트가 수신한 데이터는 마스터 모니터에 전송된다. 이는 모니터링 데이터 필터 컴포넌트에 의해 필터링이 되어 모니터링 데이터에 저장된다. 마스터 모니터는 저장된 모니터링 데이터를 저장되어 있는 서비스 품질 모델로 측정하여 QoS를 계산 이를 서비스 관리자가 볼 수 있는 형태로 제공한다.

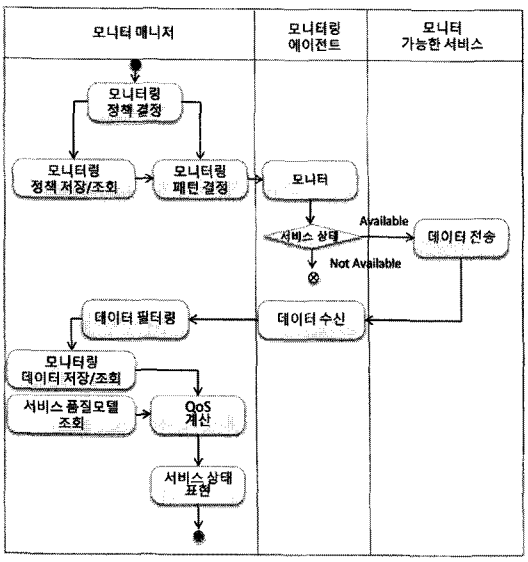

그림 4 모니터링 프레임워크 동적 모델

4.2 효율적인 모니터링 데이터 전송 패턴

이 절에서는 서비스 모니터링을 위한 전송 패턴을 정의하고 이를 적용하여 구현하기 위한 동적 모델 및 구조를 제안한다. 데이터 성격에 따라 가장 효율적인 전송패턴을 선택함으로써 제안된 모니터링 프레임워크의 효율성을 높일 수 있다. 여기서 전송이란 모니터 가능한 서비스와 모니터 에이전트 간에 데이터를 주고 받는 것을 의미하고, 전송 패턴이란 데이터 전송 시 자주 사용될 수 있는 일반화된 방법을 일컫는다. 우리는 이런 전송 패턴을 정의하기 위하여, 기존에 널리 알려진 소프트웨어 아키텍처 스타일을 참조한다[13,14]. 여기서는 풀링 패턴, 푸쉬 패턴, 블랙보드 패턴, 옵저버 패턴을 제안한다. 제안된 패턴을 적용시키기 위한 활용 방안 및 장점과 단점을 보여줌으로써 모니터링 도구 설계, 구현할 때 참조할 수 있다.

4.2.1 풀링 패턴

구조: 폴링 패턴은 데이터를 수집하는 모니터링 에이전트가 서비스나 서비스가 배치되어 있는 웹 서비스 서버, 서비스 메시지를 전송하는 ESB 등에 요청하면, 그에 따라 서비스, 웹 서비스 서버, ESB가 데이터를 전송하는 방식이다. 모니터링 에이전트와 모니터 가능한 서비스와의 제어 흐름을 살펴보면 그림 5의 (a)와 같다. 모니터링 에이전트는 모니터 가능한 서비스가 데이터를 전송할 수 있는 상태임을 확인한 뒤 모니터링 데이터를 요청하여 전송 받는다. 그림 5의 (b)와 같이 모니터링 에이전트는 모니터링 인터페이스를 통해 모니터 가능한 서비스에 모니터링 데이터를 요청하며 모니터 가능한 서비스는 가지고 있는 모니터링 데이터를 전송한다.

활용

- 모니터링 에이전트가 필요할 때만 모니터링 데이터를 전송 받기를 원하는 경우
- 모니터링 에이전트가 모니터 가능한 서비스의 누적된 기록보다는 현재 상태 데이터를 원하는 경우
- 모니터링 에이전트가 주기적인 모니터링 데이터 전송을 원할 경우

장점

- 모니터링 에이전트가 전송 받기를 원하는 시간에 데이터를 전송 받을 수 있다.
- 특정한 모니터링 데이터 즉, 가용성 같은 데이터를 요

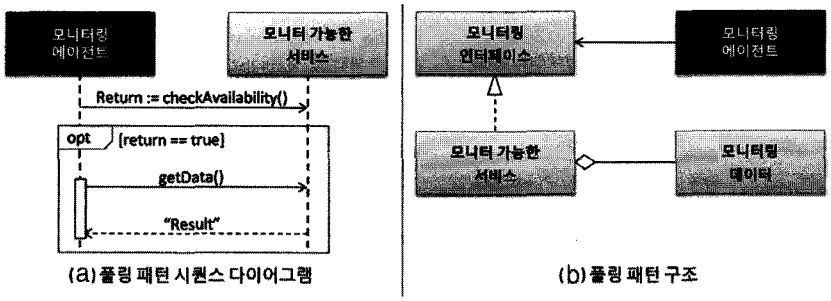

그림 5 풀링 패턴

청할 때 현재 실행되고 있는 서비스의 상태를 얻어올 수 있다.

- 이 패턴은 간단히 구현 가능하다.

단점

- 모니터링 에이전트는 모니터 가능한 서비스의 모든 상태 변화를 관찰할 수 없다. 만약 거의 모든 상태 변화를 알기 원하면 모니터 가능한 서비스에 잦은 요청을 해야 하고 이는 서비스의 성능 저하를 일으킬 수 있다.
- 같은 양의 데이터를 얻기 위해 들어가는 비용이 상대적으로 높다. 예를 들면, 서비스의 상태변화가 없다면 풀링 패턴을 사용하여 모니터링 데이터를 요청하는 경우 중복된 데이터를 전송하게 되고 이는 불필요한 비용이다.

4.2.2 푸쉬 패턴

구조: 푸쉬 패턴은 모니터링 데이터가 생성되는 서비스나 ESB가 모니터링 데이터를 수집하는 모니터링 모듈에 전달하는 방식이다. 모니터링 에이전트와 모니터 가능한 서비스와의 제어 흐름을 살펴보면 그림 6의 (a)와 같다. 모니터링 에이전트는 모니터링 가능한 서비스와 통신하기 위하여 등록한다. 서비스의 상태변화가 일어나면 모니터 가능한 서비스는 등록된 모니터링 에이전트에 상태변화가 일어난 모니터링 데이터를 전송한다. 그림 6의 (b)와 같이 모니터링 에이전트는 모니터링 인터페이스를 통해 모니터 가능한 서비스에 자신의 정보를 등록한다. 모니터링 데이터가 생성되면 모니터 가능한 서비스의 데이터 전송 컴포넌트는 모니터링 에이전트에 모니터링 데이터를 전송하며 이는 모니터링 에이전트의 Passive 모니터에 전송된다.

활용

- 모니터링 에이전트가 모니터 가능한 서비스의 모든 상태 변화를 알기 원하는 경우
- 모니터링 에이전트가 긴급한 모니터링 데이터의 빠른 전송을 원하는 경우

장점

- 모니터링 에이전트는 모니터링 가능한 서비스의 모든 상태 변화를 알 수 있다.
- 모니터링 에이전트는 항상 모니터링 가능한 서비스의 최신 상태를 가지고 있다.

단점

- 모니터링 에이전트는 모니터 가능한 서비스와의 통신을

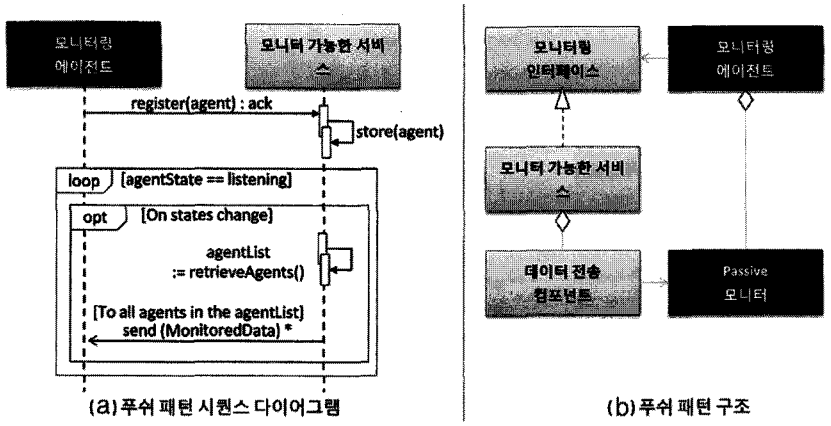

그림 6 푸쉬 패턴

위해 별로도 자원을 소비한다.

- 모니터 가능한 서비스는 모니터링 에이전트를 인식하고 모니터링 데이터를 전송하는 기능이 필요하다. 이는 모니터 가능한 서비스에 오버헤드가 된다.

모니터링 에이전트는 자신이 원하는 모니터링 데이터를 선택할 수 없고 단지 모니터링 가능한 서비스에서 전송하는 데이터를 모두 받는다.

4.2.3 블랙보드 패턴

구조: 블랙보드 패턴은 데이터 저장소를 사이에 두고, 모니터링 가능한 서비스는 모니터링 데이터를 저장소에 저장하고 모니터링 에이전트는 데이터 저장소에서 데이터를 수집하는 방식이다. 모니터링 에이전트와 모니터 가능한 서비스와의 제어 흐름을 살펴보면 그림 7의 (a)와 같다. 서비스의 상태변화가 일어나면 모니터 가능한 서비스는 데이터 저장소에 모니터링 데이터를 저장한다. 모니터링 에이전트는 필요한 모니터링 데이터를 데이터 저장소에서 가져간다. 그림 7의 (b)와 같이 모니터링 가능한 서비스는 데이터 전송 컴포넌트를 이용하여 모니터링 데이터를 데이터 저장소에 저장하다. 모니터링 에이전트는 Active 모니터를 통해 데이터 저장소에 있는 모니터링 데이터를 가져온다. 블랙보드 패턴에서 모니터링 데이터의 전송은 모니터링 가능한 서비스와 모니터링 에이전트간에 직접적인 상호작용 없이 이루어 진다.

활용

- 모니터링 에이전트가 모니터 가능한 서비스와의 직접적인 모니터링 데이터 전송을 원하는 않는 경우
- 모니터링 가능한 서비스가 모니터링 데이터를 서비스 내부가 아닌 곳에 저장하길 원하는 경우

장점

- 모니터링 에이전트와 모니터링 가능한 서비스는 느슨한 형태의 결합 관계를 가질 수 있다. 이는 동적으로 변화하는 환경에서 다양한 장점을 제공한다.
- 모니터링 에이전트와 모니터링 가능한 서비스 모두에서 모니터링 데이터 관리를 위한 비용을 줄여준다.

단점

- 모니터링 에이전트가 원하는 시간에 필요한 모니터링 데이터를 얻지 못할 수 있다.
- 모니터링 데이터 누적에 따른 성능저하가 있을 수 있다.

4.2.4 옵저버 패턴

구조: 옵저버 패턴은 모니터 가능한 서비스가 상태변화를 모니터링 에이전트에 알리면 모니터링 에이전트는 필요한 모니터링 데이터를 선택하여 수집하는 방식이다. 모니터링 에이전트와 모니터 가능한 서비스와의 제어 흐름을 살펴보면 그림 8의 (a)와 같다. 모니터링 에이전트는 모니터링 가능한 서비스와 통신하기 위하여 등록한다. 서비스의 상태변화가 일어나면 모니터 가능한 서비스는 등록된 모니터링 에이전트에 상태변화가 일어났음을 알린다. 모니터링 에이전트의 모니터링 데이터 필터 컴포넌트는 모니터링 데이터가 필요한지를 판단하여 필요한 경우 모니터 가능한 서비스에 모니터링 데이터를 요청한다. 그림 6의 (b)와 같이 모니터링 에이전트는 모니터링 인터페이스를 통해 모니터 가능한 서비스에 자신의 정보를 등록한다. 모니터링 데이터가 생성되면 모니터 가능한 서비스의 데이터 전송 컴포넌트는 모니터링 에이전트에 알림 메시지를 전송하며 이는 모니터링 에이전트의 Passive 모니터에 전송된다. 모니터링 에이전트는 필요한 모니터링 데이터인 경우 모니터링 인터페이스를 이용하여 모니터링 데이터를 전송받는다.

활용

- 모니터링 에이전트가 필요한 데이터를 선별적으로 전송받기를 원하는 경우

장점

- 모니터링 에이전트가 전송 받기를 원하는 시간에 원하는 데이터를 전송 받을 수 있다.

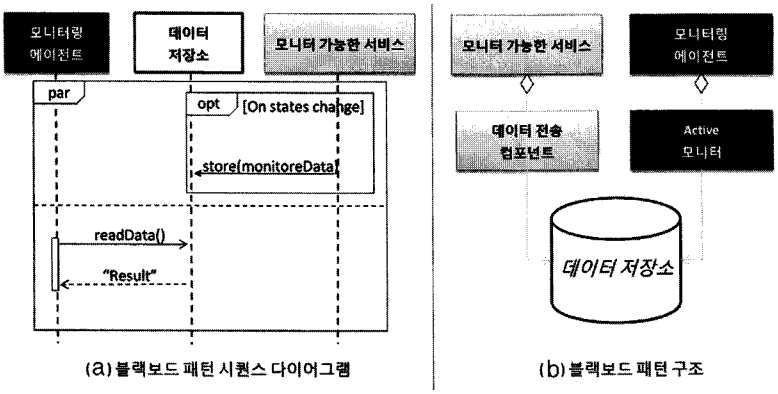

그림 7 블랙보드 패턴

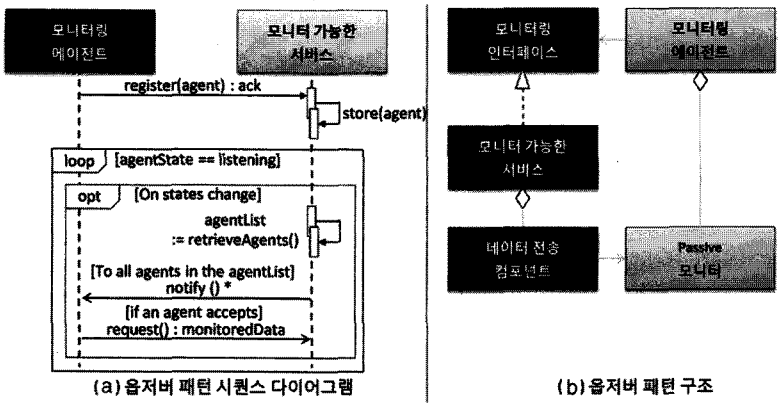

그림 8 옵저버 패턴

- 모니터링 에이전트는 중복되는 모니터링 데이터의 양을 최소화시킬 수 있다.

단점

- 모니터 가능한 서비스는 상태변화를 알린 후 일정기간 동안 모니터링 데이터를 저장해야 한다. 이는 모니터링을 위한 비용이 된다.
- 다른 패턴과 비교하여 모니터링 에이전트와 모니터 가능한 서비스간의 통신횟수가 많다. 이는 두 컴포넌트간의 강한 결합이 되어 모니터링 시 유연함이 떨어진다.

5. 모니터링 프레임워크 설계 및 구현

이 장에서는 앞에서 제시한 모니터링 프레임워크를 설계, 구현하고 일반적인 시나리오에 따라 동작하는 모습을 보여준다.

5.1 모니터링 프레임 워크 설계

3장에서 추출한 모니터링 프레임워크의 기능적 요구사항을 바탕으로 모니터링 프레임워크의 기능을 추출하면 다음과 같다. 서비스 관리자는 모니터링 프레임워크에서 총 4가지의 기능 군을 이용할 수 있다; 모니터링 가능한 서비스 관리, 품질 모델 관리, 모니터 에이전트 관리, 뷰 관리.

모니터링 가능한 서비스 관리는 모니터 에이전트가 관리하는 서비스를 관리하는 기능이다. 시스템 관리자는 모니터링을 원하는 서비스를 추가, 삭제 할 수 있다. 또한 모니터링 가능한 서비스에 대한 품질 모델 설정 및 품질 모델 측정에 필요한 모니터링 데이터에 대한 각각의 전송 패턴을 설정 한다. 품질 모델 관리는 모니터링 가능한 서비스에 적용되는 품질 모델을 관리하는 기능이다. 품질 모델에는 3장에서 설명한 것과 같이 품질 속성, 품질 부속성, 매트릭에 따른 매개 변수를 설정할 수 있다. 품질 모델에서 설정한 매개 변수가 모니터링 가능한 서비스에서 전송 받을 데이터가 된다. 모니터 에이전트 관리는 시스템 관리자가 모니터링 정책을 설정하는 기능이다. 모니터링 정책에는 모니터링 데이터를 가져오는 기간 설정, 필터링 규칙 등이 이에 해당한다. 뷰 관리는 표현 계층에서 모니터링 가능한 서비스의 상태를 나타내는 기능이다. 서비스의 상태를 표현하는 방법으로 텍스트 형식, 그래픽 형식, 타블렛 형식을 선택할 수 있다. 또한 상태를 보관하기 위한 스냅샷 기능을 지원한다.

먼저 모니터링 에이전트에 저장되는 데이터의 관계를 나타내기 위해 그림 9와 같이 데이터 구조를 설계하였다. 모니터링 에이전트는 여러 개의 모니터링 정책, 품질 모델, 서비스의 정보를 가지고 있다. 모니터링 정책에는 모니터가 운영되기 위해 참조하는 정보이다. 이에는 모니터링 데이터를 가져오는 기간, 필터링 규칙, 등이 이에 해당한다. 품질 모델은 서비스의 상태를 측정하기 위한 데이터이다. 품질 모델과 서비스는 서로 다대다의 관계이다. 모니터는 각 서비스에 적용되는 품질 모델을 선택할 수 있다. 서비스는 여러 개의 모니터링 데이터를 가지고 있다. 모니터는 품질 모델을 이용하여 서비스의 모니터링 데이터를 측정하여 보여준다.

서비스 관리자와 상호작용하는 인터페이스부터 서비스와 상호작용하는 모니터링 계층까지 모니터링프레임워크가 동작하는 것을 보여주기 위해 모니터링 에이전트가 서비스 가능한 서비스에서 수집한 모니터링 데이터를 판단하고 QoS를 계산하여 화면에 표현하는 것까지의 과정, 즉 일반적인 시나리오를 보면 그림 10과 같

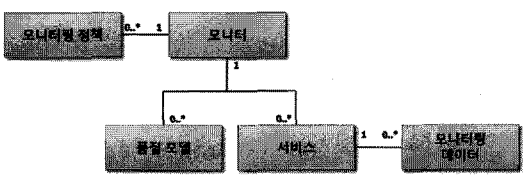

그림 9 모니터링 에이전트 데이터 구조

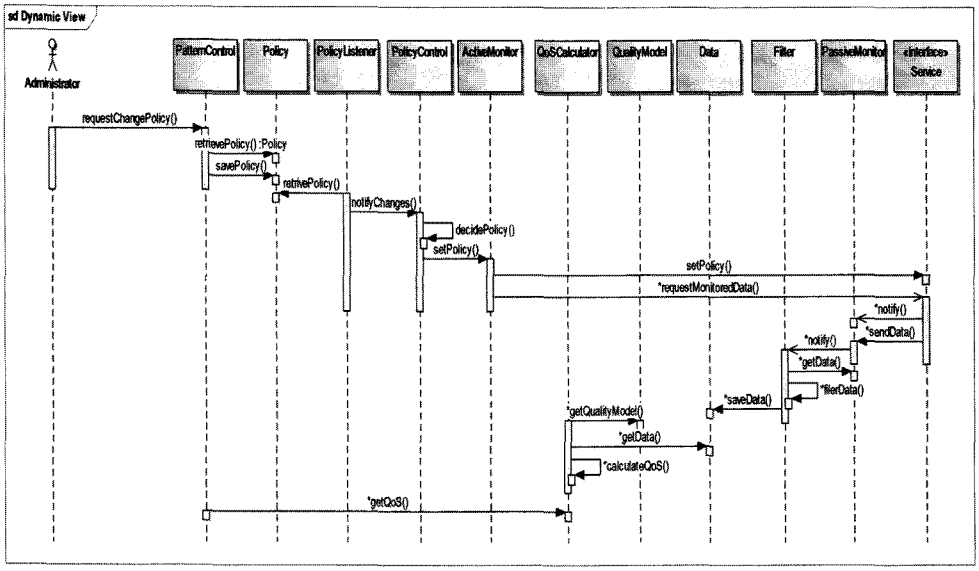

그림 10 모니터링 프레임워크 시퀀스 다이어그램

다. 먼저 서비스 관리자가 모니터링 에이전트에 모니터링 정책 변경을 요청한다. 모니터링 정책이 변경되면 모니터링 정책을 제어하는 모듈이 패턴을 및 정책 결정을 내리고 Active 모니터에 모니터링 정책 및 모니터링 패턴을 변경한다. Active 모니터는 모니터링 가능한 서비스의 인터페이스를 이용하여 결정된 모니터링 정책 및 패턴을 변경시키고, 모니터링 데이터를 요청한다. 여기 시나리오에서는 모니터링 가능한 서비스가 모니터링 데이터를 푸쉬 및 옵저버 패턴의 방식으로 전달하는 것을 가정했다. 모니터링 서비스는 Passive 모니터에 모니터링 데이터를 전달하고 이는 Filter를 통해 필요한 데이터인지 결정한 뒤 저장한다. QoS 계산기는 이렇게 수집된 모니터링 데이터를 저장되어 있는 품질 모델을 이용하여 서비스 품질을 측정한다. 이는 실시간으로 모니터링 콘솔에 전달되어 서비스 관리자가 서비스의 이상유무를 바로 확인할 수 있다.

5.2 구현

본 절에서는 구현된 모니터링 에이전트를 보여준다. 모니터링 에이전트는 Java를 이용하여 구현하였다. 개발 도구로는 NetBeans IDE V 6.7.1을 사용하였고 DB는 MySQL을 사용하였다. 그림 11은 위의 설계를 바탕으로 MSSEC 센터에서 서비스 관리 프레임워크의 일부인 모니터링 프레임워크를 구현한 것이다.

모니터링 도구를 어디서나 접근 가능하도록 하기 위하여 HTML5, 즉 웹 페이지를 이용하여 유저 인터페이스를 구현하였다. 오른쪽 화면은 모니터링 도구의 메인 화면이다. 여기서 녹색은 현재 이상 없는 상태임을 나타내고 노란색은 임계치에 거의 다다르고 있는 상태임을 나

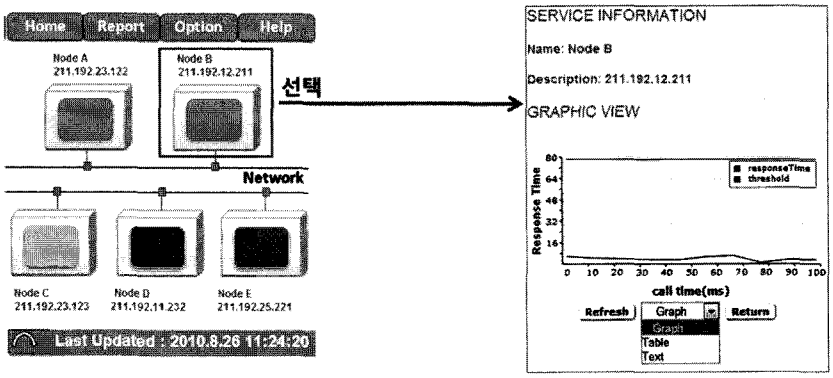

그림 11 모니터링 프레임워크 유저 인터페이스

타낸다. 붉은색은 현재 서비스의 상태가 임계치를 넘어갔음을 나타낸다. 상단 메뉴를 살펴보면 다음과 같다. Home은 모니터링 도구가 가지고 있는 모니터링 가능한 서비스 및 품질 모델을 관리하는 메뉴이다. Report는 현재 모니터링 되고 있는 모든 서비스의 상태정보를 보기 위한 메뉴이다. 이는 왼쪽 화면으로 나타내어진다. 붉은색 선은 서비스에 요구되는 응답시간의 임계치이고 파란색 선은 현재 모니터링 되고 있는 데이터를 나타낸다. 이는 임계치보다 현저히 낮으므로 서비스 품질이 높게 유지되고 있음을 알 수 있다. Graph, Table, Text의 총 3가지 형태의 화면을 볼 수 있다. 왼쪽에 있는 화면은 특정한 서비스의 Graph 화면을 보여준 것이다. Option은 서비스 모니터링 도구의 환경 설정을 할 수 있는 메뉴이다. Help는 서비스 모니터링 도구를 사용하기 위한 튜토리얼과 기능 정보들이 있는 메뉴 아이템이다.

6. 실 험

본 장에서는 전송 패턴에 따른 모니터링의 효율성을 보여주기 위하여 4.2절에서 제시된 모니터링 데이터 전송 패턴을 이용하여 각각 모니터링 하는 실험을 하였다. 본 실험을 위하여 시뮬레이션 웹 서비스를 구현하였고 서비스를 호출시키기 위하여 시뮬레이션 방법을 이용하였다.

이 실험의 실험 시나리오는 다음과 같다. 실험 환경으로는 시뮬레이션 웹 서비스를 5개를 Glassfish V2 서버에 배치하였다. 이는 각각 풀링 패턴, 푸쉬 패턴, 블랙보드 패턴, 옵저버 패턴 Glassfish V2 서버 컴퓨터의 사양은 CPU 인텔 펜티엄 3.0GHz, 메모리 2GB이고, 운영체제로는 Windows Vista를 사용한다.

실험은 다음과 같은 네 가지 과정으로 수행된다.

1. 구현한 시뮬레이터는 각각의 서비스를 같은 횟수로 호출한다.
2. 호출된 시뮬레이션 웹 서비스에는 모니터링 데이터가 생성된다.
3. 모니터링 에이전트는 시뮬레이션 웹서비스에서 수집한 모든 데이터를 저장소에 저장한다.

시뮬레이션 웹서비스에 모니터링 데이터를 얻어오기 위해 모니터링 인터페이스를 다음과 같이 구현하였다.

- 먼저 웹서비스의 실행 시간을 구하기 위해 *getTimeBeginExecutingServiceOperation()*과 *getTimeEndExecutingServiceOperation()*을 모니터 인터페이스에 추가 후 서비스 안에서 구현하여 값을 가져갈 수 있게 하였다. 이 데이터는 호출 될 때마다 생성되는 데이터로 일정시간 이상 저장해야 하는 데이터이다.
- 서비스 호출 시 발생하는 데이터 베이스 예외상황 발생 등을 알려주는 데이터를 수집하기 위해 *getExceptionsServiceOpertation()*을 모니터링 인터페이스에 추가하였다. 이 데이터는 예외 발생에 따라 값을 바꾸기 때문에 상태 값이며 상태가 변할 때마다 값이 변한다.
- *getNumberofExceptions()*을 모니터링 인터페이스에 추가하였고, 이는 일정한 시간 마다 발생한 예외의 개수를 알려주는 메소드이다. 모니터링 인터페이스를 이용해 구현한 메소드를 호출하면 모니터링 데이터를 수집할 수 있다.

모니터링 데이터 전송 패턴의 효율성을 평가하기 위해 우리는 앞에서 제시한 모니터링 효율성 평가 기준의 평균값을 계산하였다. 여기서 고려한 상황은 모니터링 에이전트가 필요한 경우에만 데이터를 전송하는 경우, 모든 상태 변화가 있을 때마다 데이터를 전송하는 경우, 일정한 간격을 두고 데이터를 전송하는 경우 이렇게 총 3가지 이다. 이는 기반으로 상황에 따라 중요한 효율성 기준을 두고 패턴을 선택하여 사용할 수 있다.

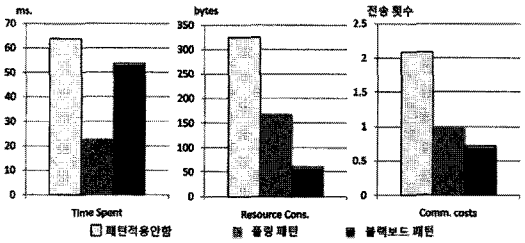

그림 12 필요한 경우에만 데이터 전송 시 결과

그림 12는 필요한 경우에만 데이터를 전송 시의 효율성을 측정한 결과이다. 먼저 필요한 경우에만 데이터를 전송하는 경우에는 4가지 패턴 중 풀링 패턴과 블랙보드 패턴이 사용 가능하다. 먼저 패턴을 적용하지 않은 경우에는 3가지 기준 모두에서 가장 낮은 효율성을 보여준다. 시간 효율성에서는 풀링 패턴이 블랙보드 패턴보다 효율적이고 자원 효율성에서는 블랙보드 패턴이 풀링 패턴보다 효율적이다. 데이터 최소성에서는 두 패턴이 거의 차이가 나지 않는다.

그림 13은 모든 상태 변화 때마다 데이터 전송 시의 효율성 측정 결과이다. 두 번째로 모든 상태 변화 때마다 데이터 전송에는 블랙보드 패턴, 푸쉬 패턴, 옵저버 패턴이 사용 가능하다. 패턴을 적용 하지 않은 경우에는 3가지 기준 모두에서 가장 낮은 효율성을 보여준다. 시간 효율성에서는 푸쉬 패턴이 가장 좋다. 이는 상태 변화가 발생하는 즉시 데이터를 전송하기 때문이다. 자원 효율성에서는 블랙보드 패턴이 가장 효율적이다. 이는 공통적인 저장소에서 데이터를 저장하기 때문이다. 데이터 최소성에서는 옵저버 패턴이 비효율적이다. 이는 모든 상태변화 때마다 이를 알려야 하기 때문이다.

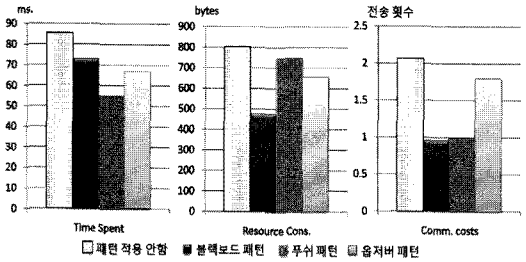

그림 13 모든 상태 변화 때마다 데이터 전송 시 결과

그림 14는 일정한 시간 간격마다 데이터 전송 시 효율성 측정 결과이다. 세 번째로 일정한 시간 간격마다 데이터 전송에는 풀링 패턴과 블랙보드 패턴이 사용 가능하다. 패턴을 적용하지 않은 경우에는 3가지 기준 모두에서 가장 낮은 효율성을 보여준다. 시간 효율성의 경우 두 가지 패턴의 효율성이 거의 차이가 없다. 이는 일정한 간격마다 데이터를 전송하기 때문에 풀링 패턴과 블랙보드 패턴이 사용하는 전송 매카니즘이 같이 때문이다. 자원 사용량에서는 블랙보드 패턴을 사용하는 경우에 가장 효율성이 좋다. 이는 공통적인 저장소에서 데이터를 저장하기 때문이다.

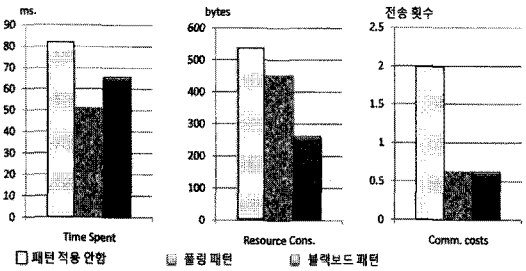

그림 14 일정한 시간 간격마다 데이터 전송 시 결과

이 실험을 통해 나타난 모니터링 프레임워크의 효율성은 다음과 같이 평가 할 수 있다. 먼저, 실험에서 패턴을 적용하여 모니터링 하는 경우에는 패턴을 적용하지 않고 모니터링 할 때보다 효율적임이 실험 결과를 통해 확인되었다. 또한 실험에서 제시된 세가지 상황에 따른 효율적인 모니터링 패턴이 존재하고 이는 효율성을 측정하는 세가지 기준에 따라 가장 효율적인 데이터 전송패턴의 선택을 가능하게 한다. 이와 같은 실험 결과를 바탕으로 모니터는 상황에 맞는 패턴의 선택에 따라 효율적인 방향으로 모니터링 할 수 있다. 예를 들면, 데이터가 필요할 때에만 데이터를 수집하는 모니터링 방법을 선택한 경우, 가장 중요한 효율성의 기준을 시간 효율성이라고 결정하면, 풀링 패턴을 적용하여 모니터를 가장 효율적으로 할 수 있다.

7. 기술적 레슨

본 장에서는 모니터링 프레임워크를 설계 및 구현하면서 발생했던 기술적인 이슈와 해결에 관하여 교훈을 정리한다. 주요 기술적 이슈로는 서비스 동적 관리, 서비스 품질 모델 동적 관리, 모니터링 효율성 관리가 있다.

- 서비스 동적 관리: 구현 시 고려해야 할 점으로는 모니터링 에이전트의 모니터 계층이다. 모니터 계층은 모니터 가능한 서비스와 통신을 하는 계층이다. 이는 동적인 특성을 가진 모니터 가능한 서비스를 다루어야 하기 때문에 다음과 같은 점을 고려해야 한다. 먼저 모니터 가능한 서비스에 메시지를 전달하거나 메소드를 호출하기 위해서는 WSDL에 표기된 형식대로 메시지가 작성되어야 한다. 또한 Java로 만들어진 어플리케이션에서 웹서비스에 접근하기 위해서는 Stub과 Proxy 메소드를 활용해야 한다. 이는 SEI (Service Endpoint Interface)을 이용하는 방식이다. SEI는 WSDL 포트 타입 엘리먼트에 기술된 웹 서비스 오퍼레이션의 자바 표현이다. 즉, 웹 서비스와 상호작용하기 위한 자바 클라이언트에 의해 사용된 메소드를 정의한 자바 인터페이스이다. 이는 WSDL을 Apache Axis의 Java2WSDL 이나 IBM WSDK의 WSDL2Client 같은 도구를 사용하여 만들 수 있다. 그러나 모니터링 에이전트에서는 어플리케이션의 운영 시간에 동적으로 서비스의 SEI를 생성해야 한다. 이를 해결하기 위해 우리는 모니터링 인터페이스를 호출하는 공통 부분을 하나의 클래스로 추출하였으며, 등록된 서비스의 엔드포인트를 실시간에 가져와서 바인딩 하는 방법으로 해결하였다. 이 방법은 WSDL마다 WSDL2Java를 이용하여 자바 클래스를 만드는 기존 방법보다 자원 활용도 측면에서 효율적이다.
- 서비스 품질 모델 동적 관리: 모니터링 프레임워크의 범용성을 높이기 위해서는 서비스 품질 모델, 그 중에서도 실제 품질을 구하는 메트릭 부분의 범용성이 허용되어야 한다. 즉, 서비스 품질 모델과 서비스 모니터링 기능의 결합도를 낮출 수 있도록 설계하여야 한다. 본 모니터링 프레임워크에서는 이 부분을 해결하기 위하여, 가져올 수 있는 모든 모니터링 데이터의 리스트를 보여주고, 원하는 데이터와 산술 연산자를 이용해서 수식을 실시간에 수정할 수 있는 기능을 추가하였다. 이를 이용하여 서비스 품질 모델과 모니터링 간의 결합도를 낮추고, 모니터링 프레임워크의 범용성을 높일 수 있었다.
- 모니터링 효율성 관리: 서비스 모니터링 시 발생하는 성능 문제는 여러 연구에서 자주 언급되어 왔다. 이를 위해서는 서비스 모니터링에 사용되는 데이터 전송,

즉 통신 방법과 관련된 것이 요구된다. 본 모니터링 프레임워크에서는 모니터링 효율성을 향상시키기 위하여 4가지 데이터 전송 패턴을 정의하였다. 정의된 패턴을 상황 별로 적용하여 효율성을 평가하였으며, 패턴을 사용하지 않은 모니터링 전송 방법보다 세 가지 측면의 효율성이 모두 높았다. 즉, 정의된 데이터 전송 패턴을 사용하여 모니터링 프레임워크의 효율성을 높일 수 있었다.

8. 결 론

서비스 지향 컴퓨팅에서 서비스는 블랙박스 형태로 제공되어 서비스의 품질을 알기 어렵다. 서비스 제공자 및 관리자는 서비스의 품질 관리를 위해 서비스의 상태를 알아야 하고 서비스 소비자는 자신이 운영하는 서비스를 관리하기 위해 서비스의 품질을 알아야 한다. 지금까지 나온 모니터링 방법은 미들웨어에 의존하기 때문에 데이터의 포괄성, 정확성에서의 한계가 있다.

본 논문에서는 서비스를 모니터링 하기 위한 모니터링 프레임워크의 요구사항을 기능, 비기능으로 나누어 정의하였다. 도출한 요구사항을 바탕으로 모니터링 프레임워크의 아키텍처를 제시하고 서비스와의 통신을 효율적으로 하기 위해 모니터링 데이터 전송패턴을 제시하였다. 또한 제시된 아키텍처 및 패턴을 기반으로 모니터링 프레임워크의 설계와 구현을 보여주었다. 실험을 통하여 상황 별 패턴의 효율성을 증명하였다.

제시된 모니터링 프레임워크는 그 동안 제공되지 않았던 서비스의 실시간 QoS 정보를 효율적으로 알 수 있게 해준다. 이는 미들웨어 수준에서 얻을 수 있는 데이터와는 포괄성, 정확성에서 큰 차이가 있다. 또한 모니터링시의 효율성을 고려하여 이를 실제로 사용하는 데에 큰 어려움이 없다. 이를 기반으로 앞으로의 서비스 관리 연구, 즉, 결함 관리나 서비스 품질 관리에 큰 도움을 줄 것이다.

참 고 문 헌

[1] Erl, T., SOA Principles of Service Design, *Prentice Hall*, July, 2007.

[2] Cardellini, V., Casalicchio, E., Grassi, V., and Presti, F., "Efficiet provisioning of service level agreements for service oriented applications," *In Proceedings of 2nd International Workshop on Service Oriented Software Engineering (IW-SOSWE'07)*, pp.29-35, September, 2007.

[3] Zeng L., Benatallah, B., Ngu A., Dumas, M., Kalagnanam J., and Chang H., "QoS-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering*, vol.30, no.5, pp.311-327, May 2004.

[4] Cardoso, J., Sheth, A., and Miller, J., "Workflow Quality of Service," *In Proceedings of International Conference on Enterprise Integration and Modeling Technology and International Enterprise Modeling Conference(ICEIMT/IEMC'02)*, pp.303-311, April, 2002.

[5] Ran, S., "A Model for Web Services Discovery with QoS," *ACM SIGEcom Exchanges*, vol.4, no.1, pp.1-10, Spring, 2003.

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

[7] Baresi, L. and Guinea, S., "Towards dynamic monitoring of WS-BPEL processes," *In proceedings of International Conference on Service-Oriented Computing (ICSOC 2005)*, pp.269-282, 2005.

[8] Barbon, F., Traverso, P., Pistore, M., and Trainotti, M., "Run-Time Monitoring of the Execution of Plans for Web Service Composition," *In Proceedings of Automated Planning and Scheduling (ICAPS 2006)*, pp.346-349, June, 2006.

[9] Lin, K., Panahi, N., Zhang, Y., and Chang, S., "Building Accountability Middleware to Support Dependable SOA," *IEEE Internet Computing*, vol.13, no.12, pp.16-25, 2009.

[10] Wilson, P., "Communication Efficiency in Research and Development," *Journal of the American Society for Information Science*, vol.44, no.7, pp.376-382, Jan 1999.

[11] Taylor, R. N., Medvidovic, N., and Dashofy, E.M., Software Architecture: Foundations, Theory, and Practice. Wiley, 2009.

[12] Lee, H.M. and Kim, S.D., "A Method to Designing Managed Service for Efficient Service Monitoring," *In Proceedings of the Korea Computer Congress (KCC2010)*, pp.84-89, June, 2010. (in Korean)

[13] 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.

[14] Gamm, R., Johnson, R., and Booch, G., Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, 1994.

이 현 민

2009년 숭실대학교 컴퓨터학부 공학사
2009년~현재 숭실대학교 컴퓨터학과 석사과정. 관심분야는 서비스 지향 아키텍처(SOA), 서비스 관리(Service Management)

천 두 완

2005년 숭실대학교 컴퓨터학부 공학사
2006년 숭실대학교 컴퓨터학과 공학석사
2006년~현재 숭실대학교 컴퓨터학과 박사수료. 관심분야는 서비스 지향 아키텍처(SOA), 서비스 관리(Service Management)

김 수 동

정보과학회논문지 : 소프트웨어 및 응용
제 37 권 제 7 호 참조