DOI: 10.3745/KIPSTD.2008.15-D.2.235

SOA 서비스 성능 측정을 위한 실용적 품질모델

오 상 헌[†] · 최 시 원[††] · 김 수 동[†††]

요 약

서비스 지향 아키텍처(Service-Oriented Architecture, SOA)는 재사용 가능한 서비스들을 동적으로 발견하고 조립하여 완성된 어플리케이션을 만드는 효과적인 접근 방법으로 주목 받고 있다. 일반적으로 알려진 SOA의 장점으로는 개발 비용 절감, 기민성, 확장 용이성, 비즈니스 수준 재사용 등이 있다. 그러나, SOA를 널리 적용하는데 대표적인 문제점으로 성능 문제가 있으며, 이는 SOA의 특징인 분산환경에서의 배치 및 실행, 서비스 플랫폼의 이질성, 표준 메시지 포맷 사용 등에 기인한다. 따라서, SOA를 효과적으로 적용하기 위해서는 성능 문제가 개선되어야 하며, 성능 개선을 위해서는 서비스의 성능을 상세히 측정하여 문제가 어디서 발생했고 왜 발생했는지 분석할 수 있어야 한다. 이를 위해서는 우선적으로 서비스 성능을 효과적으로 측정하기 위한 품질모델이 정의되어야 한다. 그러나, 현재까지 SOA의 실행 환경 및 특징을 잘 반영한 실용적이고 상세한 성능 측정 품질모델에 대한 정의가 부족하다. 따라서 본 논문에서는 서비스 성능을 측정하기 위한 실용적인 메트릭의 집합을 가진 품질모델과 제안된 메트릭을 효과적으로 측정하기 위한 기법을 정의한다. 또한, 제안된 메트릭의 실용성과 유용성을 보여주기 위해 호텔 예약 서비스 시스템에 메트릭을 적용한다.

키워드 : 서비스, 성능측정, 품질모델, 메트릭

Practical Quality Model for Measuring Service Performance in SOA

Sang Hun Oh[†] · Si Won Choi[††] · Soo Dong Kim[†††]

ABSTRACT

Service-Oriented Architecture (SOA) is emerging as an effective approach for developing applications by dynamically discovering and composing reusable services. Generally, the benefits of SOA are known as low-development cost, high agility, high scalability, business level reuse, etc. However, a representative problem for widely applying SOA is the performance problem. This is caused by the nature of SOA such as service deployment and execution in distributed environment, heterogeneity of service platforms, use of a standard message format, etc. Therefore, performance problem has to be overcome to effectively apply SOA, and service performance has to be measured precisely to analyze where and why the problem has occurred. Prerequisite for this is a definition of a quality model to effectively measure service performance. However, current works on service performance lacks in defining a practical and precise quality model for measuring performance which adequately addresses the execution environment and features of SOA. Hence, in this paper, we define a quality model which includes a set of practical metrics for measuring service performance and an effective technique to measure the value of the proposed metrics. In addition, we apply the metrics for Hotel Reservation Service System (HRSS) to show the practicability and usefulness of the proposed metrics.

Key Words : Service, Performance Measuring, Quality Model, Metric

1. 서 론

서비스 지향 아키텍처에서 서비스는 분산환경에 배치되고 실행되는 것이 일반적인 특징이다. 서비스 등록자는 서비스 레지스트리에 서비스를 등록하고, 서비스 클라이언트는 서비스를 발견하고 구성하여 서비스 요구사항을 실행한다[1, 2, 3]. 이러한 서비스의 특징 때문에, 서비스 지향 아키텍처에서는 원격지에 분산되어있는 서비스를 실행시키는데 있어서 성능 측면에서 많은 문제를 발생시키고 있다. 서비스의 성능은 서비스의 품질 속성 중 하나로써, 전체 서비스의 품질에 많은 영향을 준다. 따라서 서비스의 성능 문제점이 잘 해결되고 향상 되어야 전체적인 서비스의 품질이 향상될 수 있다. 이러한 성능 문제를 해결하기 위해서는 먼저 성능저하가 발생하는 부분이 어디인지 알 수 있어야 하고 또한 얼마나 성능저하가 발생하는지를 알 수 있어야 한다. 이러한 성능 저하를 정확하게 측정할 수 있기 위해서는 서비스 성능 측정

※ 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음.
† 준 회 원 : 숭실대학교 컴퓨터학과 박사과정
†† 준 회 원 : 숭실대학교 컴퓨터학과 박사과정
††† 종신회원 : 숭실대학교 컴퓨터학부 교수
논문접수 : 2007년 11월 15일, 심사완료 : 2008년 1월 11일

메트릭이 잘 정의 되어야 한다. 특히 SOA의 동적인 특성(Dynamism)과 서비스 구현 및 실행 관련 특징을 고려하여 메트릭이 정의 되어야 한다. 서비스는 두 개의 타입을 가지며 그것은 단일(Atomic Service)서비스와 복합(Composite) 서비스이다[4, 5]. 서비스들은 디자인 시점이나 실행 시점에 발견되고 서로 구성될 수 있다[1, 2, 6, 7]. 서비스들은 유상태(Stateful) 혹은 무상태(Stateless)일 수 있다[1, 2, 7, 8]. 마지막으로 서비스들은 특화 가능하거나(Adaptable) 특화 가능하지 않을(Non-Adaptable) 수 있다[1, 2, 6, 7]. 따라서, 본 논문에서는 기존연구에서 세부적으로 다뤄지지 않은 서비스 성능 측정 메트릭에 대해 SOA의 특징을 반영하여 세부적인 성능을 측정할 수 있는 메트릭을 정의한다. 또한 제안된 메트릭 별로 실용적인 성능측정 기법을 정의한다. 본 논문의 구성은 다음과 같다. 2장에서는 현재의 서비스 성능에 대한 관련연구를 기술하고, 3장에서는 SOA에서의 서비스 컴퓨팅 모델을 제안하고, 서비스 어플리케이션과 서비스 서버측면에서 각 수행되는 태스크와 상호작용에 대해서 기술을 하고, 4장에서는 서비스의 특징을 고려한 서비스 성능 메트릭을 정의하고, 5장에서는 제안된 성능 메트릭을 측정할 수 있는 기법을 제안한다. 6장에서는 사례연구를 통하여 제안 성능 메트릭과 측정기법을 실용성을 보여주고, 7장에서는 평가 및 결론을 제시한다.

2. 관련연구

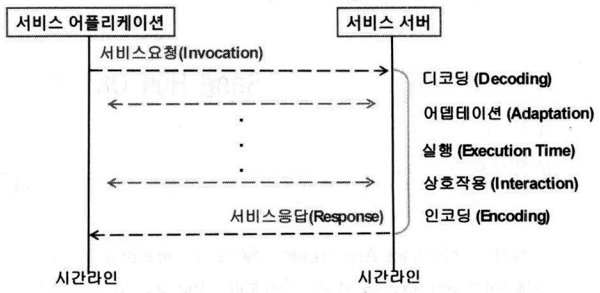
Zhang의 연구에서는 서비스 지향 컴퓨팅(Service- Oriented Computing, SOC)에서 서비스의 신뢰성과 배치(deployments) 관리 등에 대한 서비스 프로세스와 이 프로세스를 기반으로 하여 프레임워크를 제안하였다[9]. 또한 이 연구에서는 제안된 프레임워크의 시스템 아키텍처를 제안하고, 서비스 프로세스의 4단계에 대한 논리흐름을 정의하였다. 서비스 프로세스의 4단계중에 책임모델(accountability model)은 3단계에 걸쳐 서비스 프로세스 실행(Service Process Execution)이 적용된다. 서비스 프로세스 실행 단계에서는 SLA(Service Level Agreement)기반과 기능장애 발생에 대한 근본적인 원인을 규명하는 Bayesian network을 이 연구에서는 적용하였다.

Song의 연구에서는 웹 서비스를 위한 성능분석 방법론을 제안하였다[10]. 이 방법론은 시간과 비용절감에 대해 시뮬레이션을 기반으로 하여 성능분석 방법론과 서비스 결합을 위한 5가지의 테스트 활동 그리고 성능 메트릭을 정의하였다. 하지만, 이 방법론에는 서비스의 조립(Composition), 적용(Adaptation)에 대한 동적 모니터링을 측정할 수 있는 메트릭의 확장이 필요하다.

Kim의 연구에서는 웹 서비스 성능 향상을 방법과 웹 서비스을 식별하기 위한 모델을 제시하였다[12]. 이 연구에서는 SLA의 접근성과 웹 서비스 식별을 위한 신뢰성등에 대한 응답시간과 만족도 그리고 처리량에서 나오는 메시지 분류에 대한 가치를 가장 우선적으로 도출하였다. 따라서 소비자의 요구사항은 우선순위에 의하여 제공되어야 하며, 가장 높은 우선순위 요구사항으로부터 성능을 향상시키기 위해 필요한 방법을 이 연구에서 제시하였다.

3. SOA의 서비스 컴퓨팅 모델

본 장에서는 서비스 어플리케이션과 서비스가 배치되어 있는 서비스 서버간의 서비스 요청에서부터 요청된 서비스가 다시 응답되는 과정을 설명한다. (그림 1)에서는 서비스 어플리케이션측에서 서비스 서버측으로 서비스 요청을 하면 서비스 요청을 받은 서버측에서는 서비스에서 처리할 입력정보를 디코딩을 한다. 그런다음, 서비스 소비자의 요구사항에 따라 특화 될 수 있는 어뎁테이션이 발생한 후에 요청된 서비스가 수행이 된다. 서비스가 수행되는 과정에서 서비스 어플리케이션 또는 서비스 소비자와의 상호작용이 발생될 수 있으며, 이러한 과정이 끝난 후에 서비스 어플리케이션측으로 정보를 보내기 위해 다시 인코딩 과정을 거쳐 최종적으로 서비스 어플리케이션측으로 요청한 서비스에 대해 응답을 한다.



(그림 1) SOA의 서비스 컴퓨팅 모델

3.1 서비스 어플리케이션(Service Application)

서비스 어플리케이션은 소비자측면의 시스템으로서 소비자가 요청하는 서비스에 대해서 서비스 서버측으로 요청하는 역할을 한다. 또한 서비스 어플리케이션측에서는 서비스 실행과정 중 서비스 서버와의 상호작용도 발생될 수 있다. 서비스 서버측에서 요청된 서비스에 대해서 응답이 오면 서비스 어플리케이션측은 다시 서비스 소비자에게 전달되고, 서비스 어플리케이션의 역할은 끝이 난다.

3.2 서비스 서버(Service Server)

서비스 서버는 서비스들을 처리하기 위한 서버로서, 서비스 어플리케이션으로부터 받은 서비스 요청에 대해 서비스에서 처리할 입력정보를 디코딩을 하고 선택적으로 서비스 소비자의 요구사항에 따라서 특화 할 수 있는 어뎁테이션이 수행며, 이러한 과정이 끝난 후에는 서비스 태스크 과정이 수행된다. 이 과정에서 서비스 서버측에서는 서비스 소비자와 상호작용이 발생할 수도 있다. 서비스의 모든 과정이 수행되고 나면 다시 서비스 어플리케이션측으로 요청된 서비스를 전송하기 위해 서버측에서는 인코딩과정을 거쳐 최종적으로

서비스 어플리케이션측에서 요청한 서비스가 전송된다.

3.3 서비스 요청과 응답(Service Invocation and Response)

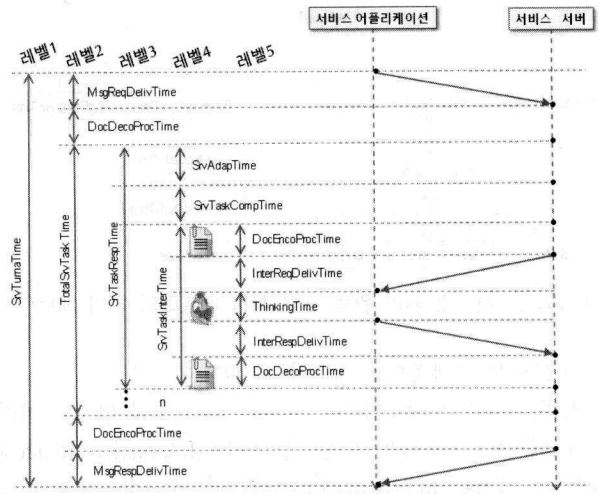
서비스 요청은 서비스 어플리케이션과 서비스 서버 사이에서 가장 최초로 발생되는 단계로서 서비스 요청 후에 서비스 서버측에서는 문서변환이나 메시지패싱이 일어나며 또한 서비스 서버측에서 원거리 접속을 통하여 서비스들을 요청한다. 그리고 이렇게 요청된 서비스들을 서버측에서 수행하고 나면 다시 서비스 어플리케이션측으로 요청된 결과가 다시 돌아온다. 즉, 서비스 요청에 대해서는 서비스 어플리케이션측에서 담당을 하고 요청된 서비스를 수행하여 다시 서비스 어플리케이션측으로 응답을 보내는 것은 서비스 서버측에서 담당 한다.

3.4 서버측 태스크(Server Side Task)

서비스 어플리케이션측에서 요청이 들어오면 서비스 서버측에서는 XML기반의 서비스 명세로 인한 문서를 처리하기 위해서 XML형태의 구조로 되어 있는 웹 서비스들에 대해서 파싱이나 문서 변환을 하기 위한 디코딩 과정이 수행 된다. 또한 각 서비스들간의 메시지들은 SOAP(Simple Object Access Protocol)프로토콜을 이용하여 교환된다. SOAP을 이용하는 이유는 여러 서비스들간의 메시지의 일률화가 가능해야 하기 때문이다. 서비스 서버 실행 중에 서비스 어플리케이션으로부터 요청된 서비스가 처음 요구사항과 달리 소비자의 입력이나 요구사항의 변경 등이 발생 할 경우 서버측은 요청된 서비스에 맞게 어뎁테이션을 한다. 이 과정에서 서비스 서버는 두 단계에 걸쳐 어뎁테이션을 수행한다. 첫 번째는 요청에 대한 구조적 시스템과 서비스에 대한 구조적 서비스 사이에 문법적 비교를 하고, 두 번째는 요구사항에 대한 제약 시스템과 서비스에 대한 제약 시스템을 비교한다. 서비스 서버 실행 도중 소비자의 추가 입력을 위한 상호작용이 발생될 수 있다. 이러한 상황은 서비스 서버측에서 서비스 어플리케이션측으로 요청된 서비스를 보낸 후에 서비스 어플리케이션측으로부터 다시 추가 입력을 받아야 할 상황이 생길 수 있기 때문이다. 서비스 어플리케이션측에서 요청한 서비스를 서비스 서버측에서 완료하고 난 후에 다시 서비스 어플리케이션측으로 요청한 서비스를 전송하기 위해 인코딩 과정을 수행하고 최종적으로 요청한 서비스를 서비스 어플리케이션측으로 전송한다.

4. 서비스 성능 메트릭

본 장에서는 3장의 SOA 서비스 컴퓨팅 모델을 기반으로 서비스 성능측정 메트릭을 정의한다. 서비스의 성능 품질 속성은 서비스 처리 완료 소요시간(Service Turnaround Time, SrvTurnaTime), 그리고 서비스 처리량(Service Throughput) 두 가지의 메트릭을 기반으로 측정된다. (그림 2)는 서비스를 사용하는 서비스 응용프로그램과 서비스를 제공하는 서버의 상호작용을 표현한 것으로, 각 상호작용에



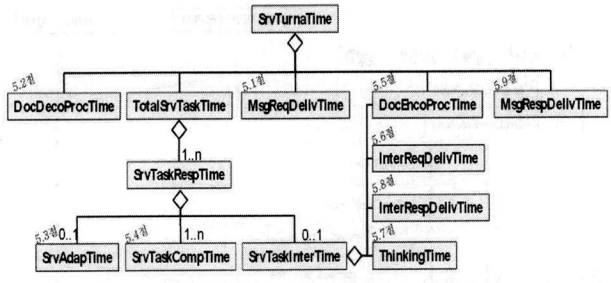
(그림 2) 서비스 성능 측정 메트릭

따라 서비스의 성능 측정에 필요한 메트릭을 레벨별로 보여준다. (그림 2)에서 서비스 어플리케이션과 서비스서버에서 시작되는 수직선 점선은 시간라인(Time Line)을 의미한다. 시간의 흐름에 따라서 서비스 어플리케이션과 서비스를 위한 서버 사이의 메시지의 흐름을 보여준다.

4.1 서비스 처리 완료 소요 시간(Service Turnaround Time, SrvTurnaTime)

서비스 처리 완료 소요 시간은 서비스 어플리케이션이 서비스를 사용하기 위하여 서비스로 요청을 보낸 시점과 서비스 어플리케이션에 응답이 도착한 시점까지의 시간이다. (그림 3)은 서비스 처리 완료 소요 시간 메트릭과 하위 메트릭의 포함관계를 정의한다. 서비스 처리 완료 소요 시간 메트릭은 메시지 요청 전달 시간(Message Request Delivery Time, MsgReqDelivTime), 문서 디코딩 처리 시간(Document Decoding Processing Time, DocDecoProcTime), 전체 서비스 태스크 실행 시간(Total Service Task Time, TotalSrvTaskTime), 문서 인코딩 처리 시간(Document Encoding Processing Time, DocEncoProcTime), 그리고 메시지 응답 전달 시간(Message Response Delivery Time, MsgRespDelivTime)의 5개의 하위 메트릭을 포함하고 있다. (그림 3)과 같이 서비스 처리 완료 소요시간을 측정하기 위해 필요한 리프(Leaf) 메트릭의 실용적인 측정기법은 5장에서 제시한다.

전체 서비스 태스크 실행 시간 메트릭은 하나 이상의 서비스 태스크 응답 시간을 포함한다. 서비스 태스크 실행 시간 메트릭은 서비스 어뎁테이션시간(Service Adaptation Time, SrvAdapTime) 메트릭, 서비스 태스크 수행 시간(Service Task Computing Time, SrvTaskCompTime) 메트릭, 서비스 태스크 상호작용 시간(Service Task Interaction Time, SrvTaskInterTime) 메트릭을 포함하고 있다. 서비스에 따라서 어뎁테이션을 지원하지 않는 경우와 서비스 어플리케이션과 상호작용이 발생하지 않을 수 있기 때문에, 서비스 어뎁테이션 시간 메트릭과 서비스 태스크 상호작용 시



(그림 3) 서비스 처리 완료 소요 시간 메트릭의 하위 메트릭

간 메트릭은 선택적으로 적용될 수 있다.

서비스 태스크 상호작용 시간 메트릭은 문서 인코딩 처리 시간 메트릭, 상호작용 요청 전달 시간(Interaction Request Delivery Time, InterReqDelivTime) 메트릭, 상호작용 응답 전달 시간(Interaction Response Delivery Time, InterRespDelivTime) 메트릭, 생각 시간(Thinking Time) 메트릭, 그리고 문서 디코딩 처리 시간 메트릭을 포함하고 있다. 서비스 태스크 상호작용 시간 메트릭의 하위 메트릭들은 서비스 태스크 상호작용 시간이 존재 해야만 측정될 수 있는 메트릭 들이며, 문서 디코딩 처리 시간 메트릭과 문서 인코딩 처리 시간 메트릭은 앞서 정의된 서비스 처리 완료 소요 시간 메트릭의 하위 메트릭과 동일하게 사용된다.

서비스 처리 완료 소요 시간 메트릭은 5개의 하위 메트릭으로 구성되며, 하위 메트릭들은 9개의 리프 노드 메트릭(Leaf Node Metric)으로부터 측정된다. 서비스 처리 완료 소요 시간 메트릭은 (메트릭 1)과 같이 측정된다.

$$SrvTurnaTime = MsgReqDelivTime + DocDecoProcTime + TotalSrvTaskTime + DocEncoProcTime + MsgResDelivTime$$

(메트릭 1) 서비스 처리완료 소요시간 메트릭

(메트릭 1)에서 서비스 처리 요청 완료 소요 시간은 다섯 개의 하위 레벨 메트릭들로 구성된다. SrvTurnaTime 값의 범위는 0 보다 큰 실수 시간 값을 가지며, SrvTurnaTime 값이 낮은 값을 가질 수록 높은 성능을 내는 것을 의미한다. MsgReqDelivTime은 메시지 요청 전달 시간 (Message Request Delivery Time)으로 서비스 어플리케이션으로부터 서버에 있는 서비스에게 요청 (Request)을 보낸 시점과 서버에 있는 서비스에 요청이 도착한 시점 사이의 시간이다. 메시지 요청 전달 시간에 대한 메트릭은 (메트릭 2)와 같다.

$$MsgReqDelivTime = MsgArrivalTimetoSrv - MsgSentTimefromSrvApp$$

(메트릭 2) 메시지 요청 전달시간 메트릭

(메트릭 2)에서 MsgArrivalTimetoSrv는 서비스 어플리케이션에서 요청한 메시지가 서비스에 도착한 시점(Message Arrival Time to Service)이다. MsgSentTimefromSrvApp는 서비스 어플리케이션에서 요청한 메시지를 서비스에 보낸 시점(Message Sent Time from Service Application)이다. 따라서 메시지 요청 전달 시간은 위의 두 시점의 시간차로 측정된다. MsgArrivalTimetoSrv 값의 범위는 0 보다 큰 실수 시간 값을 가지며, MsgArrivalTimetoSrv 값이 낮은 값을 가질 수록 높은 성능을 내는 것을 의미한다.

DocDecoProcTime는 문서 디코딩 처리 시간 (Document Decoding Processing Time)으로 서비스 어플리케이션으로부터 요청이 들어온 후 서비스에서 처리할 입력정보를 디코딩하는데 소요되는 시간이다. 서비스에서 필요한 입력정보는 일반적으로 XML 형태로 표현이 되며, 서비스 어플리케이션에서 서비스로 전송될 때 보안 문제와 해당 전송 프로토콜에 적합하게 인코딩된다. 문서 디코딩 처리 시간에 대한 메트릭은 (메트릭 3)과 같다.

$$DocDecoProcTime=DocDecoProcEndTime - DocDecoProcStartTime$$

(메트릭 3) 문서 디코딩 처리시간 메트릭

(메트릭 3)에서 DocDecoProcEndTime은 서버에 있는 서비스가 서비스 어플리케이션으로부터 요청을 받았을 때, 서비스가 받은 요청을 처리할 수 있도록 디코딩이 끝난 시점(Document Decoding Processing End Time) 이다. DocDecoProcStartTime은 서버에 있는 서비스가 서비스 어플리케이션으로부터 요청을 받았을 때, 서비스가 받은 요청을 처리할 수 있도록 인코딩이 시작된 시점 (Document Decoding Processing Start Time) 이다. 문서 디코딩 처리 시간 메트릭은 문서 디코딩 처리가 끝나는 시점과 문서 디코딩 처리가 시작되는 시점의 시간차로 측정된다. DocDecoProcTime 값의 범위는 0 보다 큰 실수 시간 값을 가지며, DocDecoProcTime 값이 낮은 값을 가질 수록 높은 성능을 내는 것을 의미한다.

TotalSrvTaskTime은 전체 서비스 태스크 실행 시간 (Total Service Task Time)으로 서비스를 구성하는 태스크들의 서비스 응답 시간의 합으로 측정된다. 전체 서비스 태스크 실행 시간에 대한 메트릭은 (메트릭 4)와 같다.

$$TotalSrvTaskTime = \sum_{i=1}^{n} SrvTaskRespTime$$

(메트릭 4) 전체서비스 태스크 실행시간 메트릭

(메트릭 4)에서 SrvTaskRespTime은 서비스 태스크의 응답 시간 (Service Task Response Time)으로 서비스 태스크는 서비스 구성의 실행 단위이다. 서비스는 1..n의 태스크들로 구성되기 때문에 전체 서비스 태스크 실행시간은 서비스를 구성하는 태스크들의 응답시간의 합으로 측정된다. 서비스 태스크 응답시간 메트릭은 세 개의 서브 메트릭으로부터 측정되며 (메트릭 5)와 같다.

$$SrvTaskRespTime=SrvAdapTime+SrvTaskCompTime+SrvTaskInterTime$$

(메트릭 5) 서비스 태스크 응답시간 메트릭

(메트릭 5)에서 SrvAdapTime는 서비스 어뎁테이션 시간 (Service Adaptation Time)으로 서비스가 서비스 소비자의 요구사항에 따라 특화될 때 소요되는 시간이다. 서비스 어뎁테이션 시간 메트릭은 (메트릭 6)과 같이 서비스 어뎁테이션 시작시점(Service Adaptation Start Time, SrvAdapStartTime)과 서비스 어뎁테이션 종료 시점 (Service Adaptation End Time, SrvAdapEndTime)의 시간차로 측정된다. SrvAdap Time 값의 범위는 0보다 큰 실수 시간 값을 가지며, SrvAdap Time 값이 낮은 값을 가질 수록 높은 성능을 내는 것을 의미한다.

SrvAdapTime=SrvAdapEndTime - SrvAdapStartTime

(메트릭 6) 서비스 어뎁테이션 시간 메트릭

SrvTaskCompTime는 서비스 태스크 수행 시간 (Service Task Computing Time)으로 서비스를 구성하는 태스크의 수행소요 시간이다. 서비스 태스크 수행 시간은 (메트릭 7)와 같이 서비스 태스크 수행 시작 시점 (Service Task Computing Start Time, SrvTaskCompStartTime)과 서비스 태스크 수행 종료 시점 (Service Task Computing End Time, SrvTask CompEndTime)의 시간차로 측정된다. SrvTaskCompTime 값의 범위는 0 보다 큰 실수 시간 값을 가지며, SrvTaskCompTime 값이 낮은 값을 가질 수록 높은 성능을 내는 것을 의미한다.

SrvTaskCompTime = SrvTaskCompEndTime - SrvTaskCompStartTime

(메트릭 7) 서비스 태스크 수행시간 메트릭

SrvTaskInterTime는 서비스 태스크 상호작용 시간 (Service Task Interaction Time)으로 서비스의 태스크를 수행 중에 서비스 어플리케이션 또는 서비스 소비자와의 상호작용이 발생할 경우, 이 상호작용을 수행하는데 소요되는 시간이다. 서비스 태스크 상호작용 시간 메트릭은 다섯 개의 서브 메트릭으로부터 측정되며 (메트릭 8)과 같다.

SrvTaskInterTime=DocEncoProcTime+InterReqDelivTime +InterRespDelivTime+DocDecoProTime - ThinkingTime

(메트릭 8) 서비스 태스크 상호작용 시간 메트릭

(메트릭 8)에서 DocEncoProcTime는 문서 인코딩 처리 시간으로 서비스의 태스크를 수행 중에 서비스 어플리케이션에 입력이 필요할 때, 소비자에게 필요한 정보를 입력 받기 위하여 서비스 어플리케이션에 출력할 정보를 인코딩하는데 소요되는 시간이다. 문서 인코딩 처리 시간은 (메트릭 9)와 같이 문서 인코딩 처리 시작 시점(Document Encoding Processing Start Time, DocEncoProcStartTime)과 문서 인코딩 처리 종료 시점 (Document Encoding Processing End Time, Doc EncoProcEndTime)의 시간차로 측정된다. DocEncoProcTime 값의 범위는 0 보다 큰 실수 시간 값을 가지며, DocEnco ProcTime 값이 낮은 값을 가질 수록 높은 성능을 내는 것을 의미한다.

DocEncoProTime = DocEncoProEndTime - DocEncoProStartTime

(메트릭 9) 문서 인코딩 처리시간 메트릭

InterReqDelivTime는 상호작용 요청 전송 시간 (Interaction Request Delivery Time)으로 서비스의 태스크가 수행되면서 서비스 어플리케이션에 요청을 보낼 때 소요되는 전송 시간이다. 상호작용 요청 전송 시간은 (메트릭 10)과 같이 서버측에서 상호작용 요청 전송 시간은 상호작용 요청 전송 시작 시점 (Interaction Request Delivery Start Time, InterReq DelivStartTime)과 서비스 어플리케이션에서 요청을 받은 시점인 상호작용 요청 전송 종료 시점 (Interaction Request Delivery End Time, InterRequDelivEndTime)의 시간차로 측정된다. InterReqDelivTime값의 범위는 0 보다 큰 실수 시간 값을 가지며, InterReqDelivTime값이 낮은 값을 가질 수록 높은 성능을 내는 것을 의미한다.

InterReqDelivTime=InterReqDelivEndTime - InterReqDelivStartTime

(메트릭 10) 상호작용 요청 전송시간 메트릭

ThinkingTime은 서비스 어플리케이션에서 서비스로부터 요청을 받은 서비스 소비자가 서비스에게로 응답을 다시 보내기까지의 소요 시간 (Thinking Time) 이다. 서비스 소비자의 응답 시간은 서비스의 정확한 성능측정을 위해서 필요한 값이며, 서비스 소비자의 응답 시간이 서비스의 실제 성능에 영향을 미치지 않는다. 본 논문에서는 서비스 소비자의 응답시간을 측정 후, 정확한 서비스 성능을 측정하기 위하여 서비스 처리 완료 소요 시간에서 제외 시킨다. 서비스 소비자의 응답 시간 메트릭은 (메트릭 11)과 같이 서비스 소비자가 받은 요청 시점(Thinking Start Time, ThinkingStartTime)과 서비스 소비자가 보낸 응답 시점 (Thinking End Time, ThinkingEndTime)의 시간차로 측정된다. ThinkingTime값의 범위는 0 보다 큰 실수 시간 값을 가지며, ThinkingTime값이 낮은 값을 가질 수록 높은 성능을 내는 것을 의미한다.

ThinkingTime=ThinkEndTime - ThinkingStartTime

(메트릭 11) 서비스 소비자의 응답시간 메트릭

InterResDelivTime은 서비스 어플리케이션에서 서비스 태스크 요청에 대한 응답으로써, 서비스로 보내는데 소요되는 상호작용 응답 전송 시간(Interaction Response Delivery Time) 이다. 상호작용 응답 전송 시간 메트릭은 (메트릭 12)와 같이 상호작용 응답 전송 시작 시점 (Interaction Response Delivery Start Time, InterResDelivStartTime)과 상호작용 응답 전송 종료 시점 (Interaction Request Delivery End Time, InterResDelivEndTime)의 시간차로 측정된다. InterRes

DelivTime값의 범위는 0 보다 큰 실수 시간 값을 가지며, InterResDelivTime값이 낮은 값을 가질 수록 높은 성능을 내는 것을 의미한다.

$$InterResDdivTime = InterResDdivEndTime - InterResDdivStartTime$$

(메트릭 12) 상호작용 응답 전송시간 메트릭

DocDecoProcTime은 문서 디코딩 처리 시간 (Document Decoding Processing Time)으로 서비스 어플리케이션으로 전송된 입력정보 문서를 서비스에서 처리할 수 있도록 디코딩 하는데 소요되는 시간이다. 문서 디코딩 처리 시간 메트릭은 앞에서 정의한 (메트릭 3)과 같이 문서 디코딩 처리 시작 시점(Document Decoding Processing Start Time, DocDecoProc StartTime)과 문서디코딩 처리 종료 시점 (Document Decoding Processing End Time, DocDecoProcEndTime)의 시간차로 측정된다. DocEncoProcTime는 문서 인코딩 처리 시간(Document Encoding Processing Time)으로 서비스에서 처리된 결과를 서비스 어플리케이션에 보내기 위해서 인코딩 하는데 소요되는 시간이다. MsgResDelivTime은 메시지 응답 전달 시간 (Message Response Delivery Time)으로 (메트릭 13)과 같이, 서버에 있는 서비스에서 인코딩 된 결과 메시지를 서비스 어플리케이션으로 전송하는 시점(Message Response Delivery Start Time, MsgRes DelivStartTime)과 서비스 어플리케이션에서 인코딩 된 메시지를 받은 시점(Message Response Delivery End Time, MsgRes DelivEndTime) 사이의 시간차로 측정된다. MsgResDelivTime값의 범위는 0 보다 큰 실수 시간 값을 가지며, MsgResDelivTime 값이 낮은 값을 가질 수록 높은 성능을 내는 것을 의미한다.

$$MsgResDelivTime = MsgResDelivEndTime - MsgResDelivStartTime$$

(메트릭 13) 메시지 응답 전달시간 메트릭

4.2 서비스 처리량 (Service Throughput, STP)

서비스의 성능에 영향을 줄 수 있는 요소 중 하나가 서비스 처리량이다. 서비스 처리량은 정해진 단위 시간 동안 서비스가 받은 요청에 대해서 얼마나 많은 서비스를 처리할 수 있는 정도, 즉 처리된 서비스의 수를 측정하는 메트릭이다. 서비스 처리량은 SrvTurnaTime 과 서비스가 실행되고 있는 서버의 성능과 밀접한 연관관계가 있다. 서비스 처리량 메트릭은 메트릭14와 같이 단위 시간(Unit of Time) 당 처리된 서비스 요청의 수 (Number of Completed Service Request, NumofComplSrvReq)로 측정된다. (메트릭 14)에서 분모인 단위 시간은 여러 종류의 시간 단위가 될 수 있다. 보통 단위시간은 나노초 (nanosecond), 밀리초(millisecond), 초(second), 분(minute), 시간(hour), 또는 하루(day) 단위로 정하여 순차적 또는 병렬적으로 발생하는 서비스 요청을 처리하는 경우가 일반적이다. 따라서, 단위 시간당 처리할 수 있는 서비스 요청이 많음은 SrvTurnaTime가 짧아 지는데 많은 영향을 미친다. 물론 서비스 처리량의 경우, 서비스 요청이 들어오는 전송 시간과 요청된 서비스의 응답 전송 시간이 고려되지 않았기 때문에, 서비스 처리량 값에 따라서 SrvTurnaTime이 정비례한다고 할 순 없지만 SrvTurna Time에 밀접한 영향을 준다.

$$STP = \frac{Number\ of\ Completed\ Service\ Request}{Unit\ of\ Time}$$

(메트릭 14) 서비스 처리량 메트릭

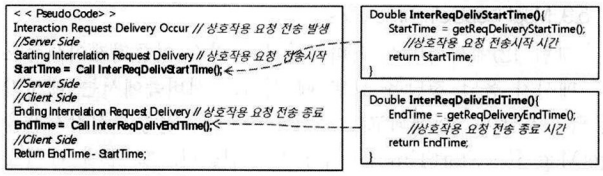
SrvTunrTime은 서비스에 대한 하나의 요청을 처리하는데 얼마나 빨리 처리가 되느냐를 측정하는 관점이라면, STP는 단위 시간당 얼마나 많은 양의 서비스를 제공할 수 있느냐를 측정하는 관점이므로, 시간과 용량 (Capacity)의 관점으로 볼 수 있다. (메트릭 14)에서 분자인 처리된 서비스 요청의 수는 WSDL (Web Service Description Language) 인터페이스에 의해서 공개된 서비스가 성공적으로 처리한 요청의 수를 나타낸다. 서비스 처리량의 값의 범위는 0 보단 큰 실수 값을 가지며, 서비스 처리량이 높은 값을 가질수록 높은 성능을 내는 것을 의미한다. 또한 서비스가 얼마나 많은 요청들을 처리할 수 있는지는 얼마나 많은 서비스 소비자들이 웹에서 접근하여 서비스를 사용할 수 있는지 정도를 나타낸다.

5. 성능 메트릭 측정기법

본 장에서는 4장에서 정의한 메트릭을 이용하여 보다 효율적으로 서비스 성능을 측정하는 기법을 정의한다. 4장에서 제안된 메트릭 중에는 실제 시간 값들을 가져오는 다이렉트 (Direct) 메트릭과 이러한 메트릭을 이용하여 계산되는 인다이렉트(Indirect) 메트릭으로 구성되며, 이러한 다이렉트 메트릭을 통하여 모든 인다이렉트 메트릭이 측정된다. 따라서 본 장에서는 실제 시간 값을 가져오는데 필요한 메트릭인 MsgReqDelivTime, DocDecoProcTime, SrvAdapTime, SrvTaskCompTime, DocEncoProcTime, InterReqDelivTime, ThinkingTime, InterRespDelivTime, MsgRespDelivTime, STP에 대해서 값을 얻어오는 기법을 제시한다.

5.1 MsgReqDelivTime 측정

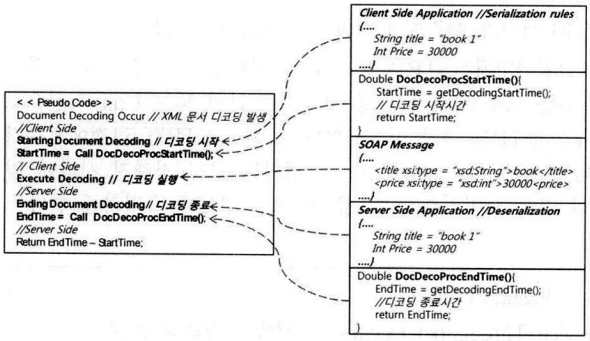
(그림 4)에서 서비스 어플리케이션측으로부터 서비스 요청이 발생되면 서비스 요청을 보내기 전에 서비스 어플리케이션 측에서 메시지 요청 전달이 실행되고, 서비스 요청 시작시간을 get ServiceStartTime을 이용하여 얻는다. 그 후에는 서비스 요청을 실행하고, 서비스 요청이 완료되면 서비스 서버측에서 getServiceEndTime을 이용하여 서비스 요청 종료시간을 얻는다. 이를 기반으로 서비스 요청 시작시간과 종료시간의 시간차를 구하여 MsgReqDelivTime을 얻는다.



(그림 4) MegRegDelivTime 측정을 위한 Pseudocode

5.2 DocDecoProcTime 측정

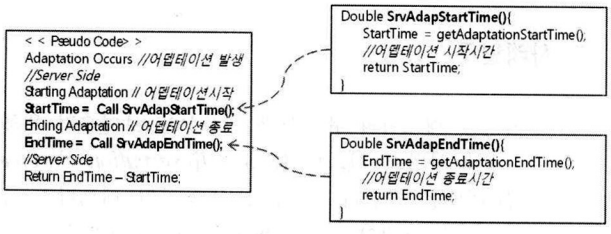
(그림 5)에서 서비스 요청이 완료된 후에 서비스에서 처리할 입력정보를 디코딩하기 위해 문서 디코딩 프로세스가 발생되면, 서버측에서 문서 디코딩 처리가 실행되고 디코딩 시작시간을 getDecodingStartTime을 이용하여 얻는다. 그 후에 디코딩 과정이 실행 되고, 실행이 완료되면 서버측에서는 디코딩 종료시간을 getDecodingEndTime을 이용하여 얻는다. 이를 기반으로 디코딩 시작시간과 종료시간의 시간차를 구하여 DocDecoProcTime을 얻는다.



(그림 5) DocDecoProcTime 측정을 위한 Pseudocode

5.3 SrvAdapTime 측정

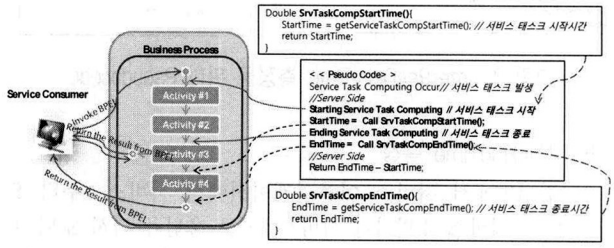
(그림 6)에서 서비스가 서비스 소비자의 요구사항에 따라 특화될 때 서비스 어뎁테이션이 발생되며, 서비스측에서는 서비스 어뎁테이션이 실행되고 어뎁테이션 시작시간을 getAdaptation StartTime을 이용하여 얻는다. 그 후에는 어뎁테이션이 실행되고, 실행이 완료되면 서버측에서는 getAdaptation EndTime을 이용하여 어뎁테이션 종료시간을 얻는다. 이를 기반으로 어뎁테이션 시작시간과 종료시간의 시간차를 구하여 SrvAdapTime을 얻는다.



(그림 6) SrvAdapTime 측정을 위한 Pseudocode

5.4 SrvTaskCompTime 측정

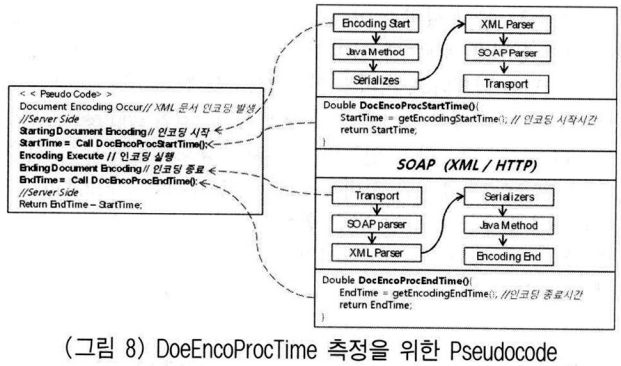
(그림 7)에서 서비스 태스크 수행시간은 서비스 서버와 어플리케이션간의 상호작용이 일어나기 전에 발생되며, 서버측에서는 서비스 태스크가 수행되고 서비스 태스크 시작시간을 get Service TaskCompStartTime을 이용하여 얻는다. 그 후에 서비스 태스크가 실행이 되고, 실행이 완료되면 다시 서버측에서 getServiceTaskCompEndTime을 이용하여 서비스 태스크 종료시간을 얻는다. 이를 기반으로 서비스 태스크 실행 시작시간과 종료시간의 시간차를 구하여 Srv TaskCompTime을 얻는다. 그리고 실제 밑의 그림에서 실행된 서비스 태스크는 Activity #1, 2 그리고 #4 이다.



(그림 7) SrvTaskCompTime 측정을 위한 Pseudocode

5.5 DocEncoProcTime 측정

(그림 8)에서 서비스의 태스크를 수행중에 서비스 어플리케이션에 입력이 필요할 때 소비자에게 필요한 정보를 입력받기 위하여 서비스 어플리케이션에 출력한 정보를 인코딩해야 한다. 따라서 서버측에서는 문서 인코딩 처리가 실행되고 인코딩 수행 시작시간을 getEncodingDoCStartTime을 이용하여 얻는다. 그 후에 인코딩이 실행되고, 실행이 완료되면 다시 서버측에서 getEncodingDoCEndTime을 이용하여 인코딩 수행 종료시간을 얻는다. 이를 기반으로 인코딩 실행 시작시간과 종료시간의 시간차를 구하여 DocEncoProcTime을 얻는다. 또한, 인코딩의 과정은 최초 자바 메소드를 Serializes하고 이것을 다시 XML Parser를 이용하여 파싱하고, 다시 SOAP Parser를 이용하여 파싱을 하고 난 다음에 전송을 한다. 그리고 이렇게 실행이 되고 난 후에는 위와 같은 방식을 반대로 수행하면 최초의 자바 메소드형태를 얻을 수 있다.



(그림 8) DoeEncoProcTime 측정을 위한 Pseudocode

5.6 InterReqDelivTime 측정

(그림 9)에서 서비스의 태스트가 수행되면서 서비스 어플리케이션에 서비스 요청을 보낼 때 서버측에서는 상호작용 요청 전달이 실행되고 상호작용 요청 전송 시작시간을 getReq Delivery StartTime을 이용하여 얻는다. 그 후에 상호작용 요청

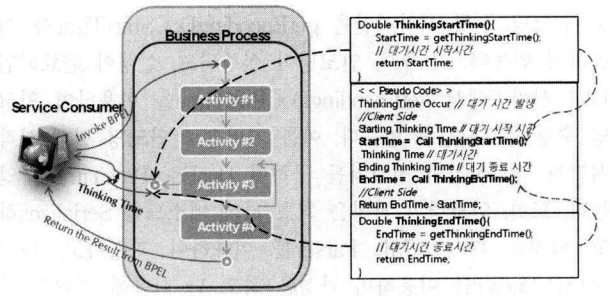
이 전송되고, 전송이 완료되면 다시 서버측에서 getReqDelivery EndTime을 이용하여 상호작용 요청 전송 종료시간을 얻는다. 이를 기반으로 상호작용 요청 전송 시작시간과 종료시간의 시간차를 구하여 InterReqDelivTime을 얻는다.

```
< < Pseudo Code> >
Interaction Response Delivery Occur // 상호작용 응답 전송 발생
//Client Side
Starting Interrelation Response Delivery // 상호작용 응답 전송시작
StartTime = Call InterResDelivStartTime();<-----
//Client Side
//Server Side
Ending Interrelation Request Delivery // 상호작용 응답 전송 종료
Call InterResDelivEndTime();<-----
//Server Side
Return EndTime - StartTime;

Double InterResDelivStartTime(){
    StartTime = getResDeliveryStartTime();
    //상호작용 응답전송시작 시간
    return StartTime;
}

Double InterResDelivEndTime(){
    EndTime = getResDeliveryEndTime();
    //상호작용 응답 전송 종료 시간
    return EndTime;
}
```

(그림 9) InterRegDelivTime 측정을 위한 Pseudocode

5.7 ThinkingTime 측정

(그림 10)에서 서비스 어플리케이션에서 서비스로부터 요청을 받은 서비스 소비자가 서비스에게로 응답을 다시 보낼 때 서비스 어플리케이션측에서는 ThinkingTime이 발생되고, 서비스 어플리케이션에서 ThinkingTime 시작시간을 getThinking StartTime을 이용하여 얻는다. 그 후에 ThinkingTime이 실행되고, 실행이 완료되면 다시 서비스 어플리케이션측에서 getThinkingEndTime을 이용하여 ThinkingTime 종료시간을 얻는다. 이를 기반으로 ThinkingTime 시작시간과 종료시간의 시간차를 구하여 ThinkingTime을 얻는다.



(그림 10) ThinkingTime 측정을 위한 Pseudocode

5.8 InterRespDelivTime 측정

(그림 11)에서 서비스 어플리케이션에서 서비스 태스크 요청에 대한 응답을 보낼 때 서비스 어플리케이션측에서 상호작용 응답 전달이 실행되고, 서비스 어플리케이션에서 서비스 서버측으로 응답을 보낼 때 상호작용 응답 전송 시작시간을 getResDelivery StartTime을 이용하여 얻는다. 그 후에 상호작용 응답이 전송되고, 전송이 완료되면 서버측에서 getResDeliveryEndTime을 이용하여 상호작용 응답 전송 종료시간을 얻는다. 이를 기반으로 상호작용 응답 전송 시작시간과 종료시간의 시간차를 구하여 InterRespDelivTime을 얻는다.

```
< < Pseudo Code> >
Interaction Response Delivery Occur // 상호작용 응답 전송 발생
//Client Side
Starting Interrelation Response Delivery // 상호작용 응답 전송시작
StartTime = Call InterResDelivStartTime();<-----
//Client Side
//Server Side
Ending Interrelation Request Delivery // 상호작용 응답 전송 종료
Call InterResDelivEndTime();<-----
//Server Side
Return EndTime - StartTime;

Double InterResDelivStartTime(){
    StartTime = getResDeliveryStartTime();
    //상호작용 응답전송시작 시간
    return StartTime;
}

Double InterResDelivEndTime(){
    EndTime = getResDeliveryEndTime();
    //상호작용 응답 전송 종료 시간
    return EndTime;
}
```

(그림 11) InterRespDelivTime 측정을 위한 Pseudocode

5.9 MsgRespDelivTime 측정

(그림 12)에서서비스 서버측에서 서비스 어플리케이션측으로 메시지 응답 전달을 보낼 때 서비스 서버측에서는 메시지 응답 전달이 실행되고, 메시지 응답 전송 시작시간을 getMsg Res StartTime을 이용하여 얻는다. 그 후에 메시지 응답이 전송되고, 전송이 완료되면 서비스 어플리케이션측에서 getMsg ResEndTime을 이용하여 메시지 응답 전송 종료시간을 얻는다. 이를 기반으로 메시지 응답 전송 시작시간과 종료시간의 시간차를 구하여 MsgResDelivTime을 얻는다.

```
< < Pseudo Code> >
Service Response Occur //서비스 응답 발생
//Server Side
Starting Transmit Service Response // 서비스 응답 시작
StartTime = Call MsgResDelivStartTime();<-----
//Server Side
//Client Side
Ending Transmit Service Response // 서비스 응답 종료
EndTime = Call MsgResDelivEndTime();<-----
//Client Side
Return EndTime - StartTime;

Double MsgResDelivStartTime(){
    StartTime = getMsqResStartTime();
    //서비스 응답 시작시간
    return StartTime;
}

Double MsgResDelivEndTime(){
    EndTime = getMsqResEndTime();
    //서비스 응답 종료시간
    return EndTime;
}
```

(그림 12) MsgRespDelivTime 측정을 위한 Pseudocode

5.10 서비스 처리량 (Service Throughput) 측정

(그림 13)에서STP는 등록된 웹 서비스 중에서 요청이 완료된 수를 모두 합하여 측정 시간의 기본이 되는 Unit 시간으로 나누면 STP을 측정 할 수 있다. 그리고 STP는 5.1절에서부터 5.9절까지의 시간 값을 얻는 방법과는 달리 정해진 시간에 얼마나 많은 서비스를 처리 할 수 있는 정도를 측정하게 된다.

```
<<Pseudo Code>>
Service Throughput Occurs // 서비스 처리량 발생
Double Compute STP (){
        Total = getNumberofCompletedServiceRequest()
        {// 성공적으로 처리된 전체 서비스의 수
        TotalServiceRequest ++;
        } //Server Side
        Time = getUnitofTime // 시간단위(nanosecond
          or millisecond or second)
          Return TotalServiceRequest / UnitofTime
          }
```

(그림 13) Service Throughput 측정을 위한 Pseudocode

6. 사례연구

본 장에서는 메트릭의 유효성과 실용성을 보여주기 위해서, 호텔 예약 서비스 시스템 (*Hotel Reservation Service System, HRSS)*을 구현하고, HRSS에 제안된 메트릭과 측정기법을 적용한 사례연구를 수행한다. 본 사례연구에서는 먼저 HRSS의 운영환경과 시나리오에 대하여 기술하고, HRSS에 적용한 메트릭의 측정된 결과를 기술한다.

6.1 시나리오

본 절에서는 사례연구에서 적용될 HRSS의 운영환경과

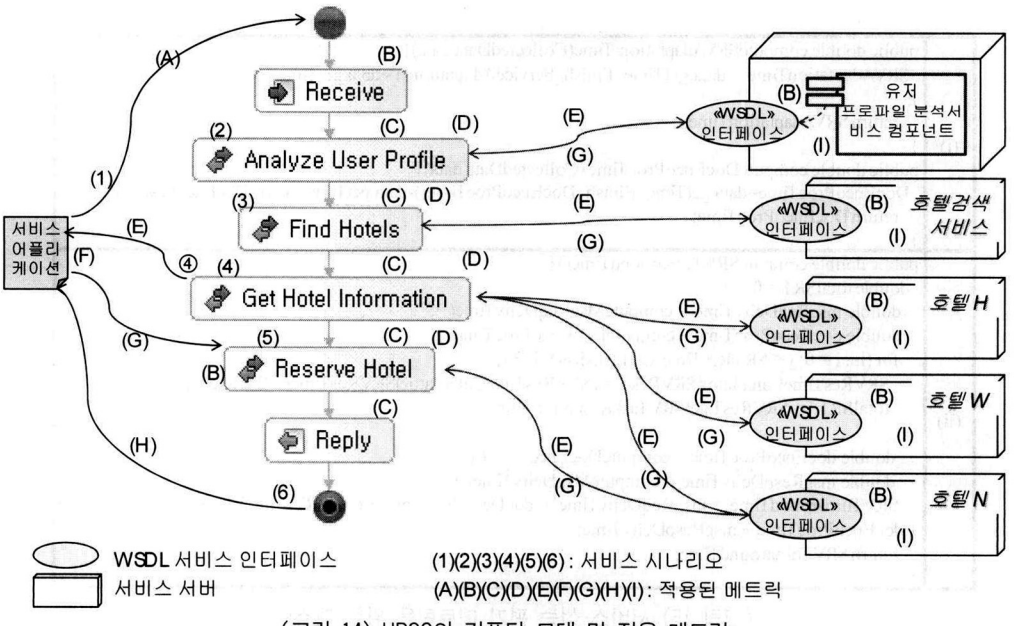
시나리오에 대하여 기술하며, 시나리오의 구성은 <표 1>과 같다. HRSS 시나리오에서 기술된 (1)에서 (6)의 과정은 하나의 복합 서비스인 비즈니스 프로세스 (Business Process) 형태로 제공되며, 이 복합 서비스는 여러 단일 서비스의 상호작용에 의해서 수행된다.

〈표 1〉 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 소비자 선택한 호텔을 예약한다. (6)예약된 호텔 정보를 호텔 예약 소비자에게 출력한다.

운영환경명세: HRSS는 서비스 소비자 계층 (Service Consumer Layer), 서비스 계층 (Service Layer) 과 같이 2개의 계층으로 구성되어 있다. 서비스 소비자 계층에는 서비스 소비자의 서비스 어플리케이션이 배치 되어있고, 서비스 계층에는 호텔 예약 서비스가 배치 되어있다. HRSS의 각 계층 구현과 실행을 위한 하드웨어 환경은 다음과 같다. CPU (인텔 펜티엄 3.4GHz), Memory (1GB RAM), OS (MS Windows XP), Network (Ethernet 10BASE-T) 으로 구성되어 있으며, 각 계층은 물리적인 독립서버에서 배치 되었다.

(그림 14)에서 (A) ~ (I)는 HRSS의 성능을 측정하기 위하여 적용된 메트릭을 기술한 것이다. 적용된 메트릭은 4장에서 정의된 메트릭을 기반으로 한다. (A)는 MsgReqDelivTime 메트릭이다. 서비스 어플리케이션이 비즈니스 프로세스에 ID/PW와 호텔 예약에 필요한 정보를 전달하기 위해서 메시지를 보낸 전송시간을 측정하기 위해 적용된 메트릭이다. (B)는 DocDecoProcTime 메트릭으로, 서비스 어플리케이션 또는 비즈니스 프로세스의 활동으로부터 전송된 문서들을 디코팅하는 데 걸리는 시간을 측정하기 위해 적용된 메트릭이다. (C)는 SrvTaskRespTime으로, 비즈니스 프로세스의 각 활동을 수행하는 데 소요되는 시간을 측정하기 위해 적용된 메트릭이다. (D)는 DocEncoProcTime으로, 비즈니스 프로세스의 각 활동들이 각 서비스들을 호출하기 위해 필요한 정보를 인코딩 하는데 소요되는 시간을 측정할 목적으로 적용된 메트릭이다. (E)는 InterReqDelivTime 메트릭으로서, 비즈니스 프로세스의 활동에서 서비스 서버로 서비스를 호출하기 위하여 요청을 보낼 때 소요되는 시간 측정에 적용하였고 비즈니스 프로세스 활동에서 어플리케이션 프로그램으로 요청을 보낼 때 소요되는 시간 측정에 적용하였다. (F)은 ThinkingTime으로서, 비즈니스 프로세스 활동 중에서 어플리케이션 프로그램으로 요청을 보낸 후 응답대기시간 측정에 적용하였다. (G)는 InterRespDelivTime으로 어플리케이션 프로그램에서 응답을 비즈니스 프로세스의 활동으로 전송할 때 소요되는 시간 측정에 적용하였고, 서비스 서버에 있는 서비스로부터 응답이 비즈니스 프로세스 활동으로 전송될 때 소용되는 시간 측정에 적용되었다. (H)는 MsgRespDelivTime으로서 비즈니스 프로세스 활동에서 어플리케이션 프로그램으로 응답 메시지를 전송하는 데 소요되는 시간 측정을 위하여 적용하였다. (I)는 SrvTask CompTime으로서, 서비스



(그림 14) HRSS의 컴퓨팅 모델 및 적용 메트릭

서버의 서비스의 태스크들이 수행되는데 소요되는 시간 측정을 위하여 적용하였다.

(그림 14)는 HRSS 시나리오가 적용된 컴퓨팅 모델과 HRSS의 성능을 측정하기 위해서 적용된 메트릭을 기술한 것이다. (1) ~ (6)은 HRSS의 시나리오의 순서가 컴퓨팅 모델에서 어느 부분에 매핑되는지를 표현한 것이다. (그림 14)에서 (1)은 서비스 어플리케이션이 ID/PW를 입력하여 호텔 예약 비지니스 프로세스에 접근하고, 호텔 예약에 필요한 정보를 비즈니스 프로세스에 전달하는 과정이다. (2)는 입력된 아이디를 기반으로 사용자의 프로파일을 분석한다. 비즈니스 프로세스의 사용자 프로파일 분석 활동(Analyze User Profile Activity)을 실행하면, 유저 프로파일 분석서비스 컴포넌트가 실행됨으로써 사용자의 프로파일을 분석한다. (3)은 서비스 소비자로부터 입력된 정보를 기반으로 후보 호텔을 찾는 호텔 검색 서비스를 호출하는데, 호텔 검색 활동(Find Hotel Activity)를 수행함으로써 호텔 검색 서비스가 호출된다. (4)는 검색된 후보 호텔 리스트를 받아서 예약 가능한 후보 호텔을 검사하고, 이 정보를 소비자에게 보여 줌으로서 소비자의 입력을 받는다. (5)는 예약할 호텔에 대한 정보를 소비자에게 입력 받아, 예약 호텔 활동 (Reserve Hotel Activity)를 수행함으로써, 호텔 N 서비스에 요청을 보내 호텔예약을 수행한다. (6)은 예약된 정보를 서비스 소비자에게 전달하여 결과를 확인할 수 있도록 한다.

6.2 서비스 성능 계산

본 절에서는 6.1절에서 HRSS 시나리오에 적용된 메트릭을 이용하여 얻은 시간 값을 기반으로 하여 전체적인 서비스 성능을 계산한다. (그림 15)는 서비스 성능 평가 메트릭을 위한 메소드의 일부분을 기술한 것이다. (i)은 서비스 어뎁테이션 소요 시간을 계산하기 위한 메소드와 문서 인코딩 처리 시간을 계산하기 위한 메소드의 알고리즘을 기술한 것이다. (ii)는 서비스 처리 완료 소요 시간을 계산하기 위한 메소드의 알고리즘을 기술한 것이다.

<표 2>는 HRSS의 서비스 성능을 측정하기 위하여, (1) ~(11)까지의 메트릭에 필요한 데이터 값들과 메트릭이 적용된 결과 값을 기술하고 있다. 메트릭에 필요한 데이터들은 비즈니스 프로세스와 단일 서비스에서 측정된 데이터 값들이며, <표 2>에서 기술된 (1)에서 (11)까지의 측정 시간들은 한번의 SrvTurnaTime을 측정하기 위해서 사용된 데이터들이다. 한번의 서비스 요청을 보내고 비즈니스 프로세스와 단일 서비스를 거쳐 수행된 결과 값이 획득될 때까지의 단계 별 소요시간을 측정한 것이다. 측정된 시간 데이터의 단위는 시간의 정확성을 높이기 위하여 나노초 단위를 사용하였다. 최종적으로 서비스의 SrvTurnaTime은 <표 2>를 기반으로 계산된다.

〈표 2〉 메트릭 측정 결과

메트릭	시작시간	종료시간	소요된 시간 (EndTime-StartTime)
(1) MsgReqDelivTime	421286016988059ns	421286027163244ns	10175185ns
(2) DocDecoProcTime	421286027163245ns	421286035245672ns	8082427ns
(3) SrvAdapTime	0	0	0
(4) SrvTaskCompTime	421286035245673ns	443992546492426ns	22706511246753ns
(5) DocEncoProcTime	443992546492427ns	443992554574854ns	8082427ns
(6) InterReqDelivTime	443992554574855ns	443992564750040ns	10175185ns
(7) ThinkingTime	443992564750041ns	443999564750041ns	7000000000ns
(8) InterRespDelivTime	443999564750042ns	443999574925227ns	10175185ns
(9) DocDecoProcTime	443999574925228ns	443999583007655ns	8082427ns
(10) DocEncoProcTime	443999583007656ns	443999591090083ns	8082427ns
(11) MsgResDelivTime	443999591090084ns	443999601265269ns	10175185ns

(i)

```
public double computeSRVAdaptation Time(CollectedData data){
 SRVAdaptationTime = data.getTime_Finish_ServiceAdaptation() - data.getTime_ Start_ServiceAdapt ation(
);
   return SRVAdaptationTime;
}
public double computeDocEncoProcTime (CollectedData data){
 DocEncoProcTime=data.getTime_Finish_DocEncoProcTime()-data.getTime_Start_DocEncoProcTime();
  return DocEncoProcTime;
}
```

(ii)

```
public double computeSRVTurnaroundTime(){
 double totalSRT = 0;
  double msgReqDelivTime = computeMsgReqDelivTime( );
  double docDecoProcTime = computeDocDecoProcTime( );
  for (int i = 0; i < SRVResTime CalList.size(); i++){
   SRVResTimeCalculator SRVResT = (SRVResTimeCalculator)SRVResTimeCalList.get(i);
   totalSRT += SRVResT.getSRVTaskResponseTime();
  }
   double docEncoProcTime = computeDocEncoProcTime( );
   double msgRespDelivTime = computeMsgDelivTime( );
  SRVTurnaroundTime = msgReqDelivTime + docDecoProcTime + totalSRT - thinkT.getTotalThinkTime()
+ docEncoProcTime + msgRespDelivTime;
  return SRVTurnaroundTime;
}
```

(그림 15) 서비스 성능 평가 메트릭을 위한 메소드

〈표 3〉 서비스 성능 계산

메트릭	계산방법	소요된 시간
SrvTaskInterTime	(5) + (6) + (7) + (8) + (9)	7036515224ns
SrvTaskRespTime	(3) + (4)·+ [SrvTaskInterTime]	22713547761977ns
TotalSrvTaskTime	[SrvTaskRespTime] + n	22713547761977ns + n
SrvTurnaTime	(1) + (2) + [TotalSrvTaskTime] + (10) + (11)	22713584277201ns

<표 3>은 HRSS의 서비스 처리 완료 소요시간 측정방법과 측정된 결과를 기술한다. <표 2>에서 직접적으로 측정될 수 있는 메트릭을 기반으로 <표 3>에서는 SrvTurnaTime을 측정한다.

1회의 서비스 요청에 대한 HRSS 서비스의 서비스 처리 완료 소요시간 <표 2>와 <표 3>에서처럼 도출될 수 있다. 서비스 처리 완료 소요시간 측면에서 HRSS 서비스의 정확한 성능 분석 평가를 위하여, 최대 SrvTurnaTime, 최소 SrvTurnaTime, 평균 SrvTurnaTime, SrvTurnaTime의 표준편차를 측정한다. <표 4>에서는 HRSS 서비스의 최대 SrvTurnaTime, 최소 SrvTurnaTime, 평균 SrvTurnaTime, SrvTurnaTime의 표준편차를 측정하기 위하여, 각 메트릭의 최대값, 최소값, 평균, 표준편차를 기술한다. 이 값들은 총 10회 요청을 HRSS서비스로 보내서 측정된 결과값 이다.

서비스 처리량 관점으로 HRSS 서비스의 성능을 측정하기 위하여, 단위 시간을 10초로 설정하고 10초의 단위 시간 동안 처리한 요청 메시지의 수를 측정하였다. 단위 시간 10초 동안 총 314개의 요청메시지를 HRSS 서비스에 보냈으며, 이중 실제 성공적으로 처리된 메시지의 수는 292건이 처리되었다. 따라서 5.10절에서 정의한 메트릭에 의해 STP는 292 / 10으로 계산되며, 결과값은 1초당 약 29건의 메시지가 처리되었다. 또한 HRSS 서비스의 최대 처리량을 측정하기 위하여 단위 10초당 최대로 처리할 수 있는 메시지 수를 측정하였다. 10초당 487,542건의 요청을 서비스로 보냈으며, 이중 실제 성공적으로 처리된 메시지의 수는 438,787건의 메시지가 처리되었다. 1초당 최대 메시지 처리량은 약 43,878건의 메시지가 처리되었다.

7. 평가 및 결론

본 장에서는 4장과 5장에서의 서비스 성능측정을 위해 정의된 메트릭들과 성능측정 기법을 기반으로 하여 본 논문의 관련연구와 비교평가 한다. 비교하기 위한 다음과 같다. 첫 번째, '서비스 성능을 측정하기 위한 품질속성들이 잘 정의되어 있는가?' 라는 비교항목에 대한 근거는 성능을 측정하기 위해서는 서비스 성능에 대한 세부 품질속성 추출이 중요하다. 그 이유는 서비스 성능 품질속성을 기반으로 하여 메트릭이 도출 되어야 하기 때문에 품질속성이 잘 정의되어야 한다. 두 번째, '품질속성 측정을 위한 메트릭들이 정의되어 있는가?'라는 비교항목에 대한 근거는 첫 번째 정의를 기반으로 하여 각각의 세부 품질속성은 메트릭으로 정의되어야 한다. 그 이유는 각각의 세부 품질속성과 메트릭들이 맵핑 되지 않는다면 정확한 성능측정을 할 수 없기 때문이다. 세 번째, '서비스의 성능을 측정하기 위한 각 메트릭에

〈표 4〉 SrvTurnaTime의 최대값, 최소값, 평균, 및 표준편차

메트릭	최대값	최소값	평균	표준편차
(1) MsgReqDelivTime	26696663ns	9638891ns	14127106ns	5164667
(2) DocDecoProcTime	11047775ns	8017035ns	8784197ns	907843
(3) SrvAdapTime	0	0	0	0
(4) SrvTaskCompTime	24929170189069ns	22493286677642ns	23078776652440ns	702840018797
(5) DocEncoProcTime	11044951ns	8061212ns	8848957ns	931979
(6) InterReqDelivTime	25411643ns	10123962ns	13916498ns	4479460
(7) ThinkingTime	8000000000ns	5000000000ns	6700000000ns	1059349905
(8) InterRespDelivTime	25633421ns	10123962ns	14002283ns	4549376
(9) DocDecoProcTime	11082323ns	8061212ns	8726757ns	945226
(10) DocEncoProcTime	11046105ns	8015464ns	8850378ns	941551
(11) MsgResDelivTime	22323050ns	9639184ns	13411153ns	3909257
(12) SrvTurnaTime	24937314475000ns	22498358358564ns	23085567319770ns	703142302656

〈표 5〉 기존 연구와의 비교 항목

비교항목	Zhang의 연구	Song의 연구	Kim의 연구	본 논문의 연구
1. 서비스 성능을 측정하기 위한 품질속성들이 잘 정의되어 있는가?	부분지원	부분지원	지원안함	지원
2. 품질속성 측정을 위한 메트릭들이 정의되어 있는가?	지원	지원안함	지원	지원
3. 서비스의 성능을 측정하기 위한 각 메트릭에 대한 측정기법이 정의되어 있는가?	지원안함	부분지원	지원안함	지원

대한 측정기법이 정의되어 있는가?'라는 비교항목에 대한 근거는 일반적인 메트릭이 아닌 실제 각 메트릭에 대해서 어떻게 측정 값을 얻을 수 있는가에 대한 기법 즉, 구현이 가능해야 한다. 그 이유는 메트릭을 통하여 실행시에 측정된 값을 얻을 수 있어야 하기 때문이다.

SOA에서 다양한 서비스 제공자(Provider)에 의해서 제공되는 서비스는 분산된 환경에 배치 되고 운영되는 것이 일반적인 특징이다. 이러한 특징은 서비스의 품질 속성 중 하나인 서비스 성능에 직접적인 영향을 준다. 또한 서비스 성능은 서비스의 여러 품질 속성 중 하나이기 때문에, 서비스 성능 문제로 인하여 서비스 전체 품질에 영향을 줄 수 있다. 본 논문에서는 SOA에서 발생할 수 있는 서비스의 성능 문제를 해결하기 위한 전제 조건인 서비스의 성능을 측정하기 위한 메트릭을 다섯 가지 레벨측면에서 정의하였고, 각 메트릭을 실용적으로 측정할 수 있는 측정기법을 제안하였다. 그리고, 본 메트릭의 적용성과 실용성을 보여주기 위하여 본 논문에서 제안한 메트릭과 측정기법을 호텔 예약 서비스 시스템에 적용한 사례연구 결과를 기술하였다.

본 논문에서 기존 연구에서 고려 하지 않았던, 서비스 처리 완료 소요 시간의 세부 메트릭에 대하여 깊이 있게 제안하였고, 제안된 서비스 성능 측정을 위한 메트릭 측정기법을 구체적으로 정의하였다. 따라서 제안된 세부 메트릭과 메트릭 측정 기법을 통하여 서비스의 성능을 상세하게 측정할 수 있으며, 측정된 세부 메트릭의 값을 통하여 어느 부분에서 성능 저하가 발생됐는지를 평가할 수 있다. 이를 기반으로 향후에 서비스 성능을 최적화 시킬 때 데이터로 활용될 수 있다.

참 고 문 헌

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

[2] Brien, L., Bass, L., and Merson, P., Quality Attributes and Service-Oriented Architectures, Technical Note CMU/SEI -2005-TN-014, September, 2005.

[3] Woodall, P., et al., "Investigating service-oriented system performance: a systematic study," Software: Practice and Experience, Vol.37, Issue.2, 2007.

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

[5] Arsanjani, A., Service-Oriented Modeling and Architecture, IBM Developerworks, 2004.

[6] Marks, E. and Bell, M., Service-Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology, Wiley, 2006.

[7] Newcomer, E. and Lomow, G., Understanding SOA with Web Services, Addison-Wesley Professional, 2004.

[8] Ferguson, D. and Stockton, M., "Service-Oriented Architecture: Programming Model and Product Architecture," IBM Systems Journal, Vol.44, No.4, pp.753-780, 2005.

[9] Zhang, Y., Lin, K., and Hsu, J., "Accountability Monitoring and Reasoning in Service-Oriented Architectures," The Journal SOCA, Vol.1, No.1, 2007.

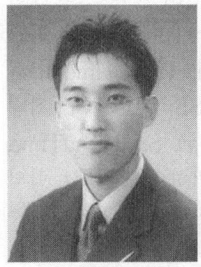
[10] Song, H., et al., "Metric, Methodology, and Tool for Performance -Considered Web Service Composition," In the Proceedings of ISCIS 2005, LNCS 3733, pp.392-401, November, 2005.

[11] Singh, M. and Huhns, M., Service-Oriented Computing, John Wiley & Sons, Ltd, 2005.

[12] Kim, D., et al., "Improving Web Services Performance using Priority Allocation Method," In the Proceedings of NWeSP 2005, pp.22-26, August 2005.

[13] Chang, S., et al., "Design of a Dynamic Composition Handler for ESB-based Services," In the Proceedings of ICEBE 2007, 24-26 October, 2007.

[14] Her, J., Choi, S., Oh, S. and Kim, S., "A Framework for Measuring Performance in Service-Oriented Architecture," Proceedings of the International Conference on Next Generation Web Services Practices (NWeSP'07), 29-31 October, 2007.



오 상 헌

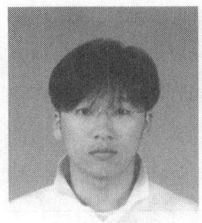
e-mail : shoh@otlab.ssu.ac.kr

2004년 건양대학교 정보전산학과 (공학사)

2006년 숭실대학교 컴퓨터학과 (공학석사)

2006년~현 재 숭실대학교 컴퓨터학과 박사과정

관심분야 : 소프트웨어 품질 공학(Software Quality Engineering), 품질모델(Quality Model), 서비스 지향 아키텍처(SOA) 품질 평가 및 관리



최 시 원

e-mail : swchoi@otlab.ssu.ac.kr

2000년 삼육대학교 컴퓨터학과(공학사)

2002년 숭실대학교 컴퓨터학과(공학석사)

2002년~현 재 숭실대학교 컴퓨터학과 박사과정

관심분야 : 품질모델(Quality Model), 서비스 지향 아키텍처(SOA), 서비스 품질 관리(Service Quality Managerment)

김 수 동

e-mail : sdkim@ssu.ac.kr

1984년 Northeast Missouri State University 전산학(학사)

1988년~1991년 The University of Iowa 전산학(석사/박사)

1991년~1993년 한국통신 연구개발단 선임연구원

1994년~1995년 현대전자 소프트웨어연구소 책임연구원

1995년 9월~현 재 숭실대학교 컴퓨터학부 교수

관심분야 : 서비스 지향 아키텍처(SOA), 객체지향 S/W공학, 컴포넌트 기반 개발 (CBD), 소프트웨어 아키텍처