EJB 어플리케이션의 성능 메트릭
(Performance Metrics for EJB Applications)

나 학 청 † 김 수 동 ††
(Hak-Chung Na) (Soo-Dong kim)

요 약 J2EE(Java 2, Enterprise Edition)의 등장으로 국내·외 수많은 기업들이 J2EE의 모델에 맞게 엔터프라이즈 어플리케이션을 개발하고 있다. 이것은 J2EE의 핵심 기술 요소인 Enterprise Java Beans (EJB)의 컴포넌트 모델이 분산 객체 어플리케이션의 개발을 간단하게 해주기 때문이다. EJB 어플리케이션은 컴포넌트 지향의 객체 트랜잭션 미들웨어를 사용하여 구현되며, 많은 어플리케이션이 분산 트랜잭션을 이용한다. EJB 서버는 이를 위한 미들웨어 서비스를 제공하여 EJB 개발자가 비즈니스 로직에 집중할 수 있도록 한다. 이러한 특징은 EJB 기술을 각광받게 하는 요인이 되었고, EJB 기반의 어플리케이션 개발에 관한 연구가 활발하게 이루어지게 하였다. 그러나 아직은 EJB 어플리케이션 운영 상태에서 성능을 측정하기 위한 메트릭에 대한 연구가 미흡하다.

본 논문에서는 운영 상태의 EJB 어플리케이션에서 서비스를 위한 워크플로우를 살펴보고, 어플리케이션 내부 작업을 여러 요소들로 분류한다. 분류된 여러 요소를 이용하여 빈(Bean) 레벨까지의 성능 측정을 위한 메트릭을 제시한다. 성능 측정에 사용되는 각 요소들을 추출하기 위해 우선 EJB 어플리케이션의 운영 상태에서 발생하는 빈의 종류에 따른 생명주기를 분석하고, 이를 기반으로 성능과 관련된 요인을 추출하여 빈의 종류에 따른 성능 요인을 메트릭에 부여할 수 있도록 한다. 또한 빈 메소드 호출시 발생하는 빈의 활성화와 메시지 전파 등의 특성을 파악하고, 어플리케이션 내에서 워크플로우에 참여하는 빈들 간의 관계를 분석하여 워크플로우에 대한 성능 측정이 가능하도록 한다. 또한 제안된 메트릭을 통하여 EJB 어플리케이션의 성능 향상을 도모할 수 있도록 한다.

키워드 : 성능 메트릭, EJB 어플리케이션, EJB 서버, 어플리케이션 측정, 부하 측정

Abstract Due to the emersion of J2EE(Java 2, Enterprise Edition), many enterprises inside and outside of the country have been developing the enterprise applications appropriate to the J2EE model. With the help of the component model of Enterprise JavaBeans(EJB) which is the J2EE core technology, we can develop the distributed object applications quite simple. EJB application can be implemented by using the component-oriented object transaction middleware and the most applications utilize the distributed transaction. EJB developers can concentrate on the business logic because the EJB server covers the middleware service. Due to these characteristics, EJB technology became popular and then the study for EJB based application has been done quite actively. However, the research of metrics for measuring the performance during run-time of the EJB applications has not been done enough.

In this paper, we explore the workflow for the EJB application service on the run-time and classify the internal operation into several elements. We propose the metrics for evaluating the performance up to the bean level by using the classified elements. First, we analyze the lifecycle according to the bean types which comes from the EJB application on the run-time as to extract each factor used in performance measurement. We also find factors related to a performance and allocate the performance factors to the metrics as the bean types. We also consider the characteristics like the bean's activation and message passing which happens during bean message call and then analyze the relations of the beans participating in the workflow of the application to make the workflow performance measurement possible. And we devise means to bring performance enhancement of the EJB application using the propose

Key word : Performance Metrics, Measurement, EJB Application, EJB Server, Evaluation, Load Balancing

† 학생회원 : 숭실대학교 컴퓨터학과
hchna@otlab.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@comp.ssu.ac.kr
접 수 일 : 2002년 3월 21일
완 료 일 : 2002년 10월 14일

1. 서 론

1.1 동기 및 배경

1998년 SUN 사에 의해 EJB 명세서가 발표된 이후, 현재 국내·외 수많은 기업들이 EJB 기반의 어플리케이

션을 개발하고 있다. 이것은 EJB에서의 컴포넌트 모델이 분산 객체 어플리케이션의 개발을 간단하게 해주기 때문이다. 분산 객체 어플리케이션은 트랜잭션적이며 확장성이 좋으며 이식 가능하다.

EJB 서버는 트랜잭션, 보안, 데이타베이스 연동 등과 같은 미들웨어 서비스를 제공하며, 클라이언트 프로그램과 EJB 어플리케이션 사이에서 메시지를 중재하여 외부로부터 직접적으로 빈을 접근할 수 없도록 한다. 또한 클라이언트 프로그램의 요청을 처리할 때 발생하는 빈의 활성화(Activation) 및 비 활성화(Passivation)와 같은 풀링(Pooling) 장치에서의 작업이나 트랜잭션 처리와 같은 EJB 서버내의 작업들은 철저히 감춰져 있다. EJB 서버는 이러한 서비스의 제공을 통해 어플리케이션 개발의 복잡도를 줄인다. 따라서 EJB 개발자는 어플리케이션의 비즈니스 로직에 집중할 수 있다[1].

이러한 특성은 EJB 기술을 선호하게 하는 요인이 되었고, EJB 기반의 어플리케이션 개발을 활발하게 하였다. 이에 따라 구축된 어플리케이션의 성능을 측정하는 것은 절실히 요구되었다. 그러나 EJB 기반의 어플리케이션에 관한 연구를 활발하게 이루어진 만큼, 어플리케이션 운영 단계에서의 성능을 측정하기 위한 메트릭에 대한 연구가 그다지 이루어지지 않았고 아직까지 부족한 현실이다. 본 논문에서는 운영 상태의 EJB 어플리케이션의 서비스를 위한 워크플로우에서 성능에 영향을 끼치는 여러 요소들을 분석하여 EJB 어플리케이션에 특성화된 메트릭을 제시한다. 성능 측정에 사용되는 각 요소들을 추출하기 위해 EJB 어플리케이션의 운영 상태에서 발생하는 빈들간의 상호작용과 빈의 활성화 작용, 메시지 전파 등의 특성들을 살펴보고, 이 특성들로부터 메트릭을 이루는 요소들을 추출한다.

1.2 논문의 범위 및 구성

EJB는 J2EE의 핵심 기술로서 컴포넌트 기반의 분산 컴퓨팅을 위한 아키텍처다. EJB 서비스는 트랜잭션적인 컴포넌트와 분산 트랜잭션, 메시징과 비동기적 작업 관리, 보안과 인증 등을 포함한다. 이러한 서비스는 미들웨어인 EJB 서버에 의해서 지원되며, EJB 어플리케이션은 미들웨어를 이용하여 구현된다[2]. 본 논문에서는 EJB 어플리케이션의 특성에 적합한 성능 측정 메트릭을 제시하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 어플리케이션 성능 평가에 관한 기존 연구를 살펴보고, 3장에서는 성능 메트릭에서 기본적으로 알아야 할 사항들을 설명한다. 그리고 4장에서는 EJB 어플리케이션의 성능 측정을 위한 메트릭을 분류하고 정의하며, 5장에서는 사례연구를 통하여 정의된 성능 메트릭의 적용방법을 보이고, 6장에서는 본 논문에서 제안한 메트릭을 평가하며, 7장에서는 결론을 맺는다.

2. 관련 연구

2.1 EJB 개요

EJB는 서버 측 컴포넌트 아키텍처로서 자바를 이용한 엔터프라이즈 분산 컴포넌트 어플리케이션의 개발을 할 때, 많이 이용을 한다. EJB 개발자는 미들웨어인 EJB 서버가 제공하는 서비스를 이용하여 좀 더 쉽고 빠르게 어플리케이션을 개발할 수 있다.

EJB는 재사용 가능한 컴포넌트라고 할 수 있는 빈이라는 용어를 사용한다. 빈은 세션(Session) 빈, 엔티티(Entity) 빈, 메시지 드리븐(Message-Driven) 빈의 세가지 유형이 있으며, 각 빈은 어플리케이션에서 가장 기본적인 단위로서 성능적 관점으로 중요하게 다루어진다. 세션 빈은 비즈니스 프로세스를 모델링하며, 영구적 데이타를 나타내지 않는다. 즉, 세션 빈은 데이타베이스 안의 공유된 데이타나 다른 엔터프라이즈 자바 빈들과 같은 자원을 접근할 수 있지만, 기본적으로 비즈니스 로직을 구현한다. 엔티티 빈은 비즈니스 데이타를 재현한다. 즉, 엔티티 빈은 데이타베이스 안의 영구적인 데이타를 재현한다. 메시지 드리븐 빈은 세션 빈과 비슷하지만 비동기적으로 메시지를 수신하여 처리한다[3].

클라이언트 프로그램은 빈 인스턴스를 Home Interface(원격 또는 로컬)와 Component Interface(원격 또는 로컬)를 통해 접근한다. 로컬과 원격 인터페이스의 차이는 클라이언트 프로그램이 같은 가상 머신(Virtual Machine) 안에 존재하는가의 여부에 따라 사용된다는 점이다. 클라이언트 프로그램은 JNDI(Java Naming and Directory Interface)를 이용하여 Home 객체를 얻고, 획득된 Home 객체를 통해 EJBObject를 생성하여 빈 인스턴스에 접근한다[3, 4].

2.2 ISO / IEC 9126의 성능 요소[5]

ISO / IEC 9126은 1998년도에 ISO(International Organization for Standardization)과 IEC(International Electrotechnical Commission)의해 제안된 품질 모델로서 소프트웨어의 품질 인증을 위해 정의된 국제 표준이다. 이 품질 모델은 외부 품질 (External Quality)과 내부 품질 (Internal Quality)에 대해 여섯 개의 특성들(Characteristics)을 명시한다. 각 특성은 부특성들(Subcharacteristics)로 나뉘어지며, 부특성들은 외부(External) 메트릭과 내부(Internal) 메트릭에 의해 측정된다. 내부 메트릭은 소프트웨어 개발 과정 중에 나오는 산출물을 측정하기 위해

이용되는 반면, 외부 메트릭은 개발이 완료된 소프트웨어 제품에 특성을 측정하기 위해 이용된다.

그림 1은 ISO / IEC 9126의 특성과 부특성을 나타낸 것으로써 특성은 기능성(Functionality), 신뢰성(Reliability), 사용성(Usability), 효율성(Efficiency), 유지보수성(Maintainability), 이식성(Portability)으로 모두 여섯 가지이다. 각 특성은 부특성들을 갖는다.

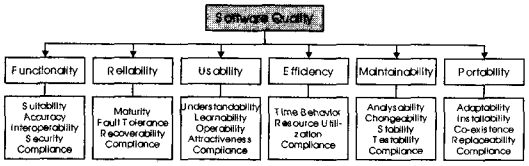


그림 1 ISO / IEC 9126의 특성과 부특성

위의 여섯 개의 특성 중 효율성(Efficiency) 특성은 "일정한 조건 하에서 사용된 자원의 양과 상관적인 적절한 성능을 제공하는 소프트웨어 제품의 능력"으로 정의된다. 효율성을 구성하는 부특성은 시간동작성(Time Behavior), 자원 활용성(Resource Utilization), 준수성(Compliance)으로 나뉘어 진다.

시간동작성은 일정한 조건 하에서 어떤 기능이 수행될 때, 응답시간과 처리시간, 처리량에 대해 적절한 비율을 제공하는 소프트웨어 제품의 능력이다. 따라서 응답시간과 Turnaround 시간, 처리량 등은 이 시간동작성에 포함되는 메트릭이다. 자원 활용성은 일정한 조건 하에서 어떤 기능이 수행될 때, 자원에 대해서 적절한 유형과 양을 사용하는 소프트웨어 제품의 능력이다. CPU나 메모리 사용률과 같은 메트릭이 자원 활용성에 포함된다. 준수성은 효율성과 관련있는 협정이나 표준에 따르는 소프트웨어 제품의 능력이다. 효율성 준수성(Efficiency Compliance)은 이 부특성에 포함된 메트릭이다.

응답시간과 Turnaround 시간, 처리량은 운영상태에서 어플리케이션의 측정하는 데 있어 유용한 메트릭이다. 각각은 다음과 같이 정의되어 있다. 응답시간은 "명시된 작업을 완료하는데 걸리는 시간"으로 정의된다. 처리량은 "주어진 시간 동안에 성공적으로 수행할 수 있는 작업의 량"으로 정의된다. Turnaround 시간은 "연관된 작업의 그룹을 시작하라는 지시를 내리고 난 후부터 그것들의 완료까지 사용자가 기다려야 하는 시간은 얼마나 되는가."로 정의된다.

2.3 클라이언트/서버(C/S)를 위한 메트릭[6]

C/S 시스템은 세 가지 메트릭이 중요하게 사용된다. 이것은 응답시간(Response Time)과 처리량 (Throughput), 비용이다. 그림 2에서 보이는 것처럼, 응답시간은 두 가지 방법으로 정의된다. 클라이언트가 새로운 트랜잭션을 시도하는 시간 t_1과 서버로부터의 응답이 클라이언트로 도달되기 위해 시작하는 시간 t_2의 간격, t_1과 응답이 완료된 시간 t_3사이의 간격, 즉 (t_2-t_1) 과 (t_3-t_1)을 응답시간으로 정의한다. 웹에서 브라우저를 사용하는 경우, 사용자가 브라우저에 나타난 링크를 클릭하는 순간 서버는 트랜잭션을 시작한다. 첫 번째 문서가 브라우저를 통해 디스플레이가 되기 시작하는 순간, 반응 간격(Reaction Interval)이 끝나고, 그 문서와 관련된 텍스트와 이미지 파일이 완전히 디스플레이 되면 응답시간간격이 끝난다[5].

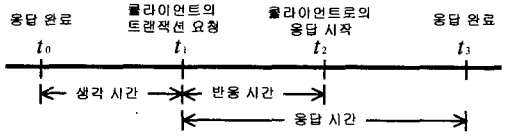


그림 2 생각 시간과 반응 시간, 응답시간의 정의

처리량은 단위 시간에 실행되는 트랜잭션의 수로서 C/S 트랜잭션의 타입에 의존된다. 즉 서버가 사용하는 트랜잭션의 종류에 따라 다르다. 비용은 성능 메트릭의 척도(Measure)와 관련이 있다. 예를 들면, 처리량의 경우에는 미리 정해진 단위에 따라 한 단위 당 지불되는 양이다.

2.4 객체 지향 어플리케이션을 위한 메트릭

소프트웨어 설계 시에 품질 평가의 기준들 중 하나는 모듈성이다. 이를 평가하는 척도에는 결합도와 응집도가 있다. 모듈 설계의 목표는 결합도를 낮추고 응집도를 높이는 것이다. 객체 지향 기술에서 기존의 연구는 결합도와 응집도를 사용한다. 객체 지향 기술에서의 결합도는 클래스 간의 상호 의존하는 정도를 나타내는 것으로 클래스 간의 결합도를 최소화시켜서 모듈의 독립성을 증가시킨다. 응집도는 클래스 안의 요소들이 서로 관련되어 있는 정도를 나타내는 것으로 소프트웨어 요소들이 서로 밀접하게 상호작용 한다면 그 요소들은 클래스 내에 포함된다[7, 8].

아키텍처 제안/검증을 위한 메트릭에서는 설계 단계에서의 정적 메트릭 뿐만 아니라 동적 메트릭을 제시한다. 정적 메트릭을 계산하기 위해 클래스 간의 관계에 대한 가중치와 두 클래스 간의 함수 호출 수, 어떤 클래스의 함수가 호출하는 외부 클래스의 수, 클래스 내부적으로 호출되는 메시지 수를 이용하여 정적 결합도와 정적 응집도를 계산한다. 동적 메트릭은 운영상태에서 발생하는 객체의 메시지 흐름을 이용하여 측정할 수 있는 메시지

의 부하와 객체 내에 포함되어 있는 함수들의 메시지 전송 회수, 호출하는 객체의 특성에 따른 가중치, 객체 간의 병렬성과 관련성, 호출하는 객체의 함수에서 외부 객체를 참조하는 함수의 메시지 수를 이용하여 동적 결합도와 동적 응집도를 계산한다[9].

2.5 Liu의 Modeling 접근[10]

Liu의 Modeling 접근방법은 소프트웨어 시스템의 성능을 연구하기 위해 사용되는 Layered Queueing Models(LQM)을 EJB 기반의 분산 엔터프라이즈 어플리케이션의 성능을 예측하기 위해 사용한다. 이 접근으로 소프트웨어 시스템의 성능 매개변수들을 식별 할 수 있으며, 주어진 Workload 패턴으로 모델들은 분석적으로 빠르게 시스템 성능을 예측될 수 있다.

LQM에서는 서버 프로세스를 "태스크(Task)"로서 모델링하며, Liu의 접근은 EJB를 태스크들로 이루어지는 기본적으로 3 계층으로 구성하여 모델링한다. 최상위 층은 EJB의 비즈니스 메소드와 대응하는 하나 이상의 요소들을 갖는다. 두 번째 층은 영속적 저장장치에 저장된 데이타에 대한 검색 또는 업데이트와 SQL문 처리, EJB로부터의 데이타베이스 연결 요청들을 받아들이는 데이타베이스(DBMS)에 대한 작업을 갖는다. 세 번째 층은 디스크 서브 시스템에 대응되는 작업을 갖는다. 디스크 서브 시스템은 하나이상의 요소를 갖는 프로세서로서 LQM으로 모델링된다. 이 때 각 요소는 Volume1, Volume2 등과 같이 디스크 요청의 유형에 대응된다. 이 계층들은 EJB 어플리케이션의 복잡도에 따라 그 계층이 더욱 분화될 수 있다. 그림 3은 EJB에 대해 기본적으로 3 계층으로 모델링될 수 있음을 보여주고 있다.

클라이언트는 EJB의 비즈니스 메소드를 호출하기 전에 빈의 EJBObject에 대한 참조를 얻어야 한다. 엔티티 빈을 사용할 때는 JNDI를 통해 빈의 홈 인터페이스를 찾고 홈 인터페이스의 Find 메소드를 호출하며, 세션 빈을 사용할 때는 홈 인터페이스의 Create 메소드를 호출한다. 따라서 EJBObject의 참조를 얻는 데 소요되는 부하를 측정하기 위해 첫 번째 계층에 "Find/Create" 요소를 추가한다.

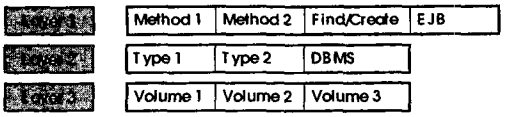


그림 3 EJB의 3 계층

LQM 모델은 EJB 어플리케이션의 성능 측정에 사용되기 위해 수정된다. 또한 모델의 수정으로 각 층의 태스크들에 대한 매개변수 값들이 결정된다. 이를 위한 작업으로 자원 사용이 EJB 빈들에 대해 메소드 레벨에서 결정될 수 있도록 어플리케이션의 윤곽을 그리고, 서비스 요청 대비 부하량에 대한 응답시간을 획득한다. 이 때, Workload는 시스템에 의해 다루어지는 동시 발생 요청의 수로서 정의된다. 클라이언트는 주어진 시나리오에 나타난 유형의 서비스 요청을 제출하고 응답을 기다린다. EJB 서버의 부하량이 측정기간 동안 일정하게 되도록 하기 위해 응답이 도착하면 클라이언트는 생각시간의 값으로 0을 갖고 다른 서비스 요청을 한다. 모델의 매개변수들의 값은 어플리케이션의 윤곽을 그릴 때 나타난 값과 벤치마크한 값을 기반으로 하여 선택된다.

성능 메트릭은 평균 응답시간 T라고 하며, 두 개의 트랜잭션에 대한 서비스 요청이 Event1과 Event2이고 각각에 대한 평균 응답시간이 T1, T2라고 할 때, 평균 응답시간 T = p*T1+(1-p)*T2로서 정의된다. 이때 p는 0부터 1사이의 값으로 Workload 패턴으로 모델링한다.

2.6 Lladó와 Lüthi의 접근

Lladó와 Lüthi의 접근은 EJB 아키텍처에 적합한 분석적인 성능 모델을 개발한다. 여기서 사용된 EJB 명세는 1.1 명세를 따른다. 시스템의 분리된 기능성을 연구하여 그 기능성들을 계층적 방법론으로 병합하며, 단위 시간당 메소드 실행 수로서 정의된 처리량과 응답시간에 관하여 그것의 성능을 예측하기 위해 메소드 호출 실행을 분석적으로 모델링한다. 빈 인스턴스가 비 활성화 상태일 때 발생하는 블로킹(Blocking)을 추출한다[11].

메소드 호출 실행에서 시스템 성능에 영향을 주는 요소는 Java Synchronization와 클라이언트와 EJB 서버 사이의 통신이다. 클라이언트와 EJB 서버 사이의 통신은 Remote Method Invocations (RMI)을 통해 이루어진다. 클라이언트의 위치에 따라 기존의 쓰레드(Thread)에서 실행될지 분리된 쓰레드에서 실행될지를 결정한다. 다른 클라이언트 가상 머신에서 발생한 호출은 항상 다른 쓰레드에서 실행한다. 그러므로 각 클라이언트 가상 머신은 최소한 하나의 쓰레드가 있다. Synchronization에서는 한 쓰레드가 코드 지역을 실행할 때, 그 코드 지역을 다른 쓰레드가 실행할 수 없도록 보호한다.

메소드는 사용자 인증을 수행하고, 실행 시에 필요한 자원의 할당은 컨테이너를 통해 이루어진다. 같은 컨테이너에 의해 생성된 엔티티 빈의 Home클래스와 빈 구현 클래스인 CGHome이나 CGBean인스턴스를 사용하는 쓰레드(클라이언트)는 그것의 컨테이너에 대해 잠금

(Lock)을 공유한다. 이 잠금은 이 객체가 컨테이너 동기화된 메소드가 호출될 때 요구된다. 각 컨테이너에서 동기화된 메소드들 중 하나만이 같은 시간에 실행될 수 있다. 이 메소드를 갖는 클래스는 컨테이너 당 한 클래스를 말한다.

그림 4는 CGBean 인스턴스가 동기화된 메소드들을 갖는 쓰레드 관리자를 사용하는 것을 보여준다. 여기서 CGBean 인스턴스는 쓰레드 관리자 잠금 자원을 공유한다.

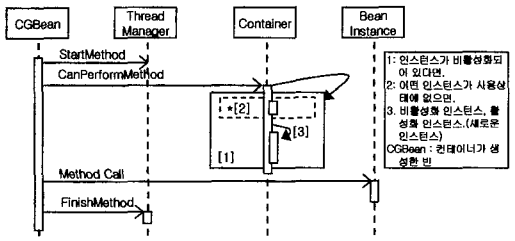


그림 4 메소드 실행 호출의 시퀀스 다이어그램

쓰레드 관리자, 컨테이너, 동기화된 메소드 실행의 다양한 단계에서 동기화의 결과로서 쓰레드들은 FCFS (First Come First Served)로 처리된다. 따라서 이를 모델링하기 위해 확장된 Queueing 모델을 적용한다[11].

EJB 성능 모델에서 나타나는 매개변수의 불명확성을 수집하기 위해 간격 매개변수를 사용한다. 시스템 설계와 구현 단계보다 이전 단계에서는 성능 모델의 매개변수 값을 거의 알 수가 없다. 간격이 분석 모델에 대해 사용되는 경우 기존의 방법들은 매개변수 재현의 유형이 이런 유형이 되도록 적용시켜야 한다. 이것은 간격 계산(Arithmetic)으로 할 수 있다. 간격 계산에서 기본적인 연산은 간격 연산으로 대신한다. 예를 들면, 간격이 $X=[\underline{x}, \bar{x}]$와 $Y=[\underline{y}, \bar{y}]$ 이면 $X + Y$ 와 $X - Y$ 는 $[\underline{x}, + \underline{y}, \bar{x}+\bar{y}]$와 $[\underline{x}, -\bar{y}, \bar{x}-\underline{y}]$로 정의된다. 제곱근이나 지수와 같은 미리 정의된 기본적인 함수들을 적용할 수 있다. 간격 계산은 실제적인 범위를 제공하지 않아서 의존 문제(Dependency Problem)가 발생한다. 이 문제는 해결하기 위해 입력 매개변수 간격들을 하부 간격들로 나누어 해결한다[12, 13, 14].

2.7 기존 연구의 한계

ISO / IEC 9126의 성능 요소는 소프트웨어에 범용적으로 적용될 수 있도록 정의되어 있다. 운영상태에서의 어플리케이션의 성능을 측정하는 데 있어서 효율성 특성의 시간 메트릭들은 매우 유용하다. 그러나 ISO / IEC 9126의 성능 요소의 범용적인 특성은 EJB 어플리케이션의 성능 메트릭으로 적용하기 위한 특성화를 요구한다. 이것은 EJB 어플리케이션의 여러 가지 특징으로 발생한다.

따라서 ISO / IEC 9126의 성능 요소를 EJB 어플리케이션이라는 특정 소프트웨어를 측정하기 위해서는 그 특징을 반영하여 EJB 어플리케이션의 성능 측정 메트릭으로 확장하여 사용해야 한다. 이것은 EJB 어플리케이션이라는 특정 소프트웨어를 측정하기 위해 그 특성을 반영해야 하기 때문이다. EJB 어플리케이션이 갖고 있는 여러 가지 특징에 따라 특성화하고 각 각의 정의를 EJB 어플리케이션에 사용하기 위해서는 좀더 명확하게 정의하고 세분화해야 한다. 이러한 정의에는 또한 빈과 그들 간의 메시지 호출, 사용되는 객체, EJB 서버 등과 같은 EJB 어플리케이션의 특징이 반영되어야 한다.

클라이언트/서버에서 사용되는 메트릭은 서버를 하나의 블랙박스로 보고 외부에서 측정되는 데이타를 이용하여 성능을 측정한다. 그러나 EJB 어플리케이션에서 요구되는 것은 운용되는 환경에서 외부에서뿐만 아니라, 그 내부에서 어떻게 운용되는지, 어떤 성능요소가 있는지를 파악하기에는 불충분하다.

객체 지향 어플리케이션에서 사용되는 메트릭은 성능 측정의 단위가 객체 단위로 이루어진다. 특히 객체 사이의 관련도에 따라 그 성능이 나타난다. 그러나 EJB 어플리케이션은 빈들로 구성되어 있으므로 객체 단위로서 측정하기에는 적당하지 않다. 하나의 빈은 두 개의 인터페이스(홈 인터페이스, 로컬 인터페이스)와 하나의 클래스(또는 추상 클래스)로 이루어 진다. 이러한 빈의 구성에 따라 빈을 구성하는 객체 간의 메시지 호출에 따른 관련도가 사용될 수 있다. 그러나 이것은 어디까지나 객체단위에 사용되는 것으로 객체보다는 큰 빈 단위의 측정이 고려되지 않아 객체 지향 어플리케이션에 사용되는 메트릭은 EJB 어플리케이션에 사용되기에는 불충분하다.

Liu의 접근은 계층화된 Queueing 모델을 사용하여 어플리케이션의 성능을 분석한다. 어플리케이션을 계층화하여 각 요소들을 측정한다. 그러나 어플리케이션 내에서 빈의 종류에 따른 생명주기의 차이에 따라 발생하는 부하와 성능 측정요소들이 충분치 않다. Lladó와 Lüthi의 접근 또한, Queueing 모델을 통해 인터벌 매개변수를 사용하여 EJB 성능 모델을 분석한다. EJB의 성능 모델에 대한 연구로서, EJB 어플리케이션이 실질적으로 운영될 때 발생하는 특성 - 하드웨어적인 환경의 영향과 분산 처리될 때 네트워크 상에서 발생되는 부하 등 - 은 고려하지 않고 있다.

우리가 논의한 기존의 메트릭들은 많은 연구가 분석, 설계 단계에서의 성능을 예측하기 위한 모델이므로, 어플리케이션 운영단계에서의 성능 측정을 위해 사용하기

에 적당하지 않다. EJB 어플리케이션은 미들웨어 서비스를 제공하는 EJB 서버를 사용해 구축된다. 따라서 상당한 부분이 블랙박스 형태로 감춰져 있기 때문에, 클라이언트 프로그램은 요청한 작업이 어플리케이션 내부에서 어떤 처리 단계로 이루어지고, 어느 지점에서 처리시간이 오래 걸리는지 파악하기 어렵다. 이와 같은 특성은 기존의 메트릭을 적절히 적용하기 어렵게 한다. 따라서 EJB 어플리케이션의 운영 단계에서 성능 측정을 위한 특성화된 메트릭이 요구된다.

3. EJB 어플리케이션 아키텍처

EJB 어플리케이션의 아키텍처는 그림 5와 같이 클라이언트 계층과 컨트롤러 계층, 비즈니스 계층, 데이타 접근 계층, 데이타베이스로 나타낼 수 있다[15]. 그림 5는 계층간의 상호 관계와 메시지 흐름을 나타내고 있다. 내용에 밑줄이 있는 사각형은 각 계층에 해당되는 빈을 나타낸다.

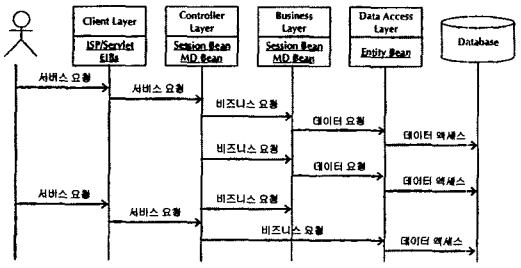


그림 5 EJB 어플리케이션 아키텍처

클라이언트 계층(Client Layer)은 엑터와 상호작용하면서, 서버에 요청을 하고 요청된 결과를 어떻게 처리할지를 정의한다. 이 층은 JSP나 서블릿(Servlet), 다른 EJB 서버에서 운영되는 빈 들, 자바 어플리케이션 등이 될 수 있다. 컨트롤러 계층(Controller Layer)은 사용자의 요구 사항을 클라이언트 층을 통해 전달 받아 요구사항을 분석하며, 수행해야 할 기능들을 정의하고 처리 순서를 제어한다. 비즈니스 계층(Business Layer)은 비즈니스 로직을 구현한다. 컨트롤러 계층과 비즈니스 계층은 세션 빈을 이용하거나, 비동기적인 메시지를 처리하는 경우는 메시지 드리븐 빈을 이용해 구현한다. 데이타 접근 계층(Data Access Layer)은 어플리케이션 데이타와 비즈니스 규칙을 지원하며, 물리적 데이타베이스에 접근 가능하게 도와주며, 데이타를 수정하고, 데이타 처리를 수행한다. 이 층은 데이타베이스에 저장된 데이타를 재현하며, 데이타로의 접근을 위해서 엔티티 빈을 이용해 구현한다. 물리적인 데이타는 데이타베이스에 저장되며, 엔티티 빈은 JDBC를 통하여 저장된 데이타에 접근한다[15].

엑터(Actor)의 서비스 요청은 클라이언트 계층의 컨트롤러 계층을 통해 EJB 어플리케이션으로 전달된다. 컨트롤러 계층은 RMI를 통해 요청을 전달하며, 데이타 접근 계층의 데이타베이스 계층에 대한 접근은 JDBC를 통하여 이루어 진다. 서비스 요청의 처리를 위해 요구되는 빈들에 대한 메시지는 메시지를 송수신하는 두 빈의 위치가 원격인 경우에 RMI를 이용하여 전달된다.

4. EJB 성능 메트릭

EJB는 java.rmi.Remote를 확장하여 javax.ejb.SessionBean과 javax.ejb.EntityBean 인터페이스를 정의한다. 클라이언트가 사용하는 빈이 같은 자바 가상 머신 상에 존재하는 경우를 제외하고는 언제나 RMI 상에서 모든 작업이 이루어진다. 또한 클라이언트 프로그램이 서비스를 요청하기 위해서는 Home 객체와 EJBObject를 순서대로 바인딩한 후에 EJBObject를 통해 빈 인스턴스의 서비스를 요청한다. 서비스를 요청을 받은 빈은 독자적으로 서비스를 제공할 수 있지만, 대개는 서비스에 필요한 다른 빈들의 메소드를 호출하여 서비스를 제공한다.

EJB 어플리케이션의 성능 요소는 클라이언트의 서비스 요청을 처리하기 위해 다른 빈들에게 메시지를 보내며, RMI를 통해 클라이언트와 빈과의 통신이 이루어 진다는 것을 고려해야 한다. 여기에는 메시지를 수신하는 빈의 타입은 물론 각 빈들이 분산되어 위치하는지 아니면 같은 서버에 위치하는 지와 같은 빈의 위치에 따른 분산 정도, 빈이 활성화되는 시간, 빈의 로딩 시간 등이 포함되어야 한다.

또한, EJB 어플리케이션의 성능 측정을 위해서는 서비스의 요청을 받고, 요청된 서비스를 처리하며, 클라이언트가 응답을 받는 일련의 프로세스를 여러 작업으로 분할하여 각 작업 별로 계산해야 한다.

4.1 메트릭 분류를 위한 기준

EJB 어플리케이션은 컴포넌트 지향의 분산 객체 트랜잭션 미들웨어(OTM)을 사용하여 구현된 확장 가능한 분산 어플리케이션이다. 즉, EJB 어플리케이션은 트랜잭션적인 컴포넌트들로 구성되며, 분산 트랜잭션을 사용한다.

EJB 어플리케이션은 클라이언트의 서비스 요청을 받은 빈이 다른 빈을 사용하지 않고 단독으로 처리하거나 다른 빈들에게 메시지를 보냄으로써 처리한다. 이 빈들은 서로 다른 EJB 컨테이너나 같은 EJB 컨테이너에 배치될 수 있다. 그림 6은 클라이언트 프로그램이 Bean A

에게 서비스를 요청하고, Bean A는 요청된 서비스를 처리하기 위해 다른 빈들에게 메시지를 보내는 것을 보여주고 있다.

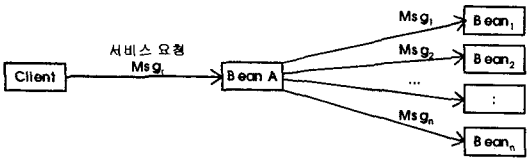


그림 6 요청된 서비스를 위한 메시지의 분할

본 논문에서는 메트릭을 클라이언트가 어떤 빈에 서비스 요청을 하는 것으로부터 요청된 서비스가 처리되고 응답되는 과정을 단계적으로 나누어 정의하며, 여기서 정의된 메트릭을 이용하여 EJB 어플리케이션의 성능을 측정한다.

4.2 메트릭의 분류와 정의

4.2.1 인스턴스 생성시간

호출된 어떤 메소드에서 C1타입의 변수 varC1을 위해 new 연산자를 사용하여 생성된 클래스 C1의 인스턴스 $Object_i$의 생성 시간 $C(Object_i)$ 은 new 연산자를 사용하여 생성을 요청하는 시간 $T_{creationReqForObject\_i}$ 과 varC1이 $object_i$의 참조를 받은 시간 $T_{receiveReference\_Object\_i}$ 사이의 시간으로 다음과 같이 정의된다.

[정의 1] $C(Object_i) = T_{receiveReference\_Object\_i} - T_{creationReqForObject\_i}$

4.2.2 객체 간의 메소드 실행 시간과 메소드 응답시간

자바 어플리케이션에서 어플리케이션은 메소드의 호출로서 이루어진다. 하나의 메소드 호출은 필요에 따라서 다른 객체의 메소드의 호출을 가져온다. 그림 7은 Object1에서 Object2의 메소드 MethodX의 호출이 Object3의 MethodY의 호출을 발생시키는 것을 보여주고 있다. 여기서 T_{xxxx} 는 호출과정 중에 발생하는 이벤트 시간이다.

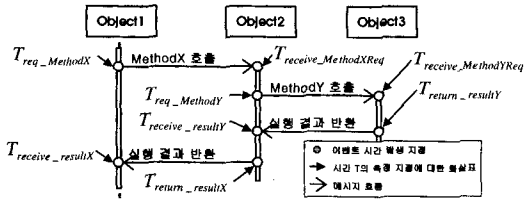


그림 7 객체 간의 메소드 실행 과정

메소드 실행시간 $EOM(Method_i)$은 메소드를 호출하는 m 객체 O_m이 n 객체 O_n의 i번째 메소드 $Method_i$을 호출하였을 때 O_n상에서 실행이 시작된 시간 $T_{start_execMethod_i}$과 완료된 시간 $T_{end\_execMethod\_i}$ 사이의 간격이며, 실행되는 메소드 로직을 위한 프로세스 시간 $I_{process_LogicOfMethod_i}$과 실행 중에 Method_i에서 호출되는 n 개의 메소드에 대해 j번째 메소드의 응답시간 $ROM(Method_j)$들의 합, Method_j 호출 시 발생하는 대기시간 $I_{waitForExecMethod\_j}$ 으로 구성되며, 다음과 같이 정의된다

[정의 2] $EOM(Method_i) = T_{end\_execMethod\_i} - T_{start\_execMethod\_i} = \sum_{j=0}^{n} ROM(Method_j) = I_{waitForExecMethod\_j} + I_{process\_LogicOfMethod\_i}$

예를 들면, 그림 7의 Object1의 Object2에 있는 메소드 MethodX의 실행시간은 $T_{return\_resultX} - T_{receive\_MethodXReq}$ 으로 계산되며, Object2의 Obejct3에 있는 메소드 MethodY의 응답시간을 포함한다.

메소드 응답시간 $ROM(Method_i)$ 은 객체 O_m이 객체 O_n의 i번째 메소드 $Method_i$를 호출하는 시간 $T_{req_Method_i}$과 실행 결과를 받는 시간 $T_{receive_result_i}$의 간격이며, 메소드의 실행시간을 포함하므로, $Method_i$의 요청 전송 시간을 $I_{send\_Method\_iReq}$ 로, 실행결과의 전송시간을 $I_{send\_resultOfMethod\_i}$ 이라고 하면 다음과 같이 정의할 수 있다.

[정의 3] $ROM(Method_i) = T_{receive\_result\_i} - T_{req\_Method\_i} = I_{send\_Method\_iReq} + I_{send\_resultOfMethod\_i} + EOM(Method_i)$

그림 6의 예에서, Object 1의 MethodX 호출에 따른 응답시간은 $T_{receive\_resultX} - T_{req\_MethodX}$ 으로 계산된다. 또한 MethodX의 응답시간은 실행시간 $EOM(MethodX)$과 MethodX 호출 요청 전송시간, 그 결과값 전송시간을 포함한다.

4.2.3 바인딩 시간

클라이언트 프로그램이 빈에서 제공하는 서비스를 사용하기 위해서는 두 단계의 작업을 완료해야 한다. 우선 클라이언트 프로그램은 JNDI를 이용하여 접근하려는 빈의 Home 객체를 찾아 바인딩한다. 두 번째로 Home 객체를 통해 서버 측에 EJBObject의 생성을 요청해 서버 측에 EJBObject가 생성되면, 클라이언트는 이 객체를 반환 받아 바인딩하는 작업이다.

Home 객체의 바인딩은 그림 8과 같이 JNDI 서비스를 이용하여 이루어지며, Home 객체를 요청하는 시간과 찾는 시간, 반환 시간으로 이루어진다. $Bean_i$를 위한 Home 객체의 바인딩 $B_{EJBHome}(Bean_i)$ 은 각 시간을 $I_{send_EJBHomeOfBean_iReq}$과 $I_{lookup\_EJBHomeOfBean\_i}$, $I_{send\_EJBHomeOfBean\_i}$ 이라고 하면

$B_{EJBHome}(Bean_i)$ 은 다음과 같이 정의된다.

[정의 4] $B_{EJBHome}(Bean_i) = I_{send\_EJBHomeOfBean\_iReq} + I_{lookup\_EJBHomeOfBean\_i} + I_{send\_EJBHomeOfBean\_i}$

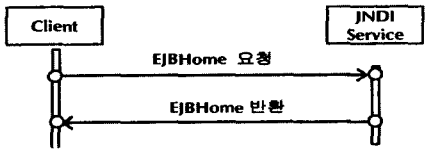
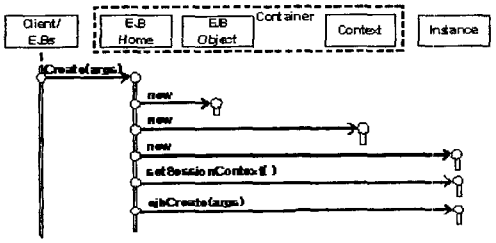


그림 8 Home 객체의 바인딩을 위한 메시지 전송

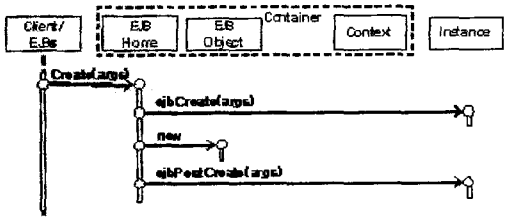
EJBObject의 바인딩은 클라이언트 프로그램에서 빈 인스턴스를 사용하기 위해 EJBObject를 생성하고 획득하는 과정이며, 빈의 유형에 따라 차이가 있다. 그림 9는 세션 빈과 엔티티 빈에서 EJBObject의 바인딩 처리를 보여주고 있다. EJBObject의 바인딩 시간은 공통적으로 RMI를 통한 메시지 전송시간 $I_{send\_EJBObjReqOfBean\_i}$ 과 EJBObject의 반환 시간 $I_{send\_EJBObjOfBean\_i}$, 컨테이너 안에서 EJBObject 생성시간 $C(EJBObjOfBean_i)$, 바인딩을 위한 클라이언트에서의 호출 메소드 대기시간을 포함한다. 이외에는 빈의 종류에 따라 차이를 보인다.

세션 빈의 경우, 그림 9의 (a)에서 보이듯이 빈 인스턴스의 생성시간을 포함한다. 이 바인딩 처리 단계에서 어플리케이션의 성능적 측면에서 의미있는 시간은 빈을 위한 Context 생성시간 $C(Context)$, 빈 인스턴스 생성시간 $C(InstOfBean_i)$, 생성된 인스턴스의 초기화를 위한 setSessionContext()와 ebjCreate(args)에 따른 응답시간 $ROM(setSessionContext)$, $ROM(ejbCreate)$와 EJB Object의 바인딩을 위한 컨테이너의 로직 처리시간 $I_{process\_LogicForBindingEJBObj}$, 클라이언트의 Create() 메소드처리를 위한 EJBObject에서의 대기시간 I_{wait_Create}이 추가된다.

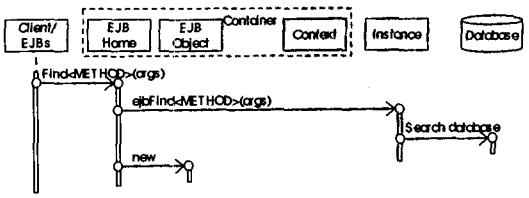
그림 9의 (b)와 (c)는 엔티티 빈에서 Create(…)와 Find(…)를 이용한 바인딩 처리를 보여주고 있다. 바인딩에 의해 사용되는 빈 인스턴스에 대해 컨테이너는 새로운 빈 인스턴스를 항상 생성하는 것은 아니고, 풀에서 유효한 빈 인스턴스를 사용할 수 있다. 따라서 엔티티 빈은 빈 인스턴스 생성시간 $C(InstOfBean_i)$ 이나 활성화 시간 $A(Bean_i)$ 을 포함한다. Create(…)를 이용하는 경우, 클라이언트의 create() 메소드의 호출이 빈 인스턴스의 ejbCreate(), ejbPostCreate()를 이용한 초기화와 EJBObject 생성 과정을 갖는다. 빈 인스턴스가 생성되거나 활성화되면, CMP를 사용하는 엔티티 빈은 컨테이너에 의해서, BMP의 경우에는 ejbCreate()를 이용하여 데이타베이스 안에 엔티티 인스턴스의 상태를 생성한다. 따라서 이 단계에서의 의미있는 시간은 ejbCreate메소드의 응답시간 $ROM(ejbCreate)$, ejbPostCreate 메소드의 응답시간 $ROM(ejbPostCreate)$, EJBObject의 바인딩을 위한 컨테이너의 로직 처리시간 $I_{process\_LogicForBindingEJBObj}$, 데이타베이스에 대한 접근 시간 $I_{accesssDB}$, 클라이언트의 Create() 메소드처리를 위한 EJBObject에서의 대기시간 I_{wait_Create} 이 추가된다.



(a) 세션 빈의 Object 생성



(b) 엔티티 빈의 Object 생성



(c) Find(…)를 이용한 엔티티 빈의 Object 생성

그림 9 빈 Object의 바인딩 시간

Find(…)를 이용하는 경우, 클라이언트가 find(…) 호출하면, 해당 엔티티 빈의 Home 객체는 빈인스턴스의 ejbFind(…)를 통해 데이타베이스로의 조건 검색을 수행한다. ejbFind(…) 수행이 완료되면, EJBObject를 생성하게 된다. 따라서 이 단계에서의 의미 있는 시간은 ejbFind메소드의 응답시간 $ROM(ejbFind)$과 EJBObject의 바인딩을 위한 컨테이너의 로직 처리시간 $I_{process\_Logic ForBindingEJBObj}$, 클라이언트의 Find() 메소드처리를 위한 Home 객체에서의 대기시간 I_{wait_Find} 이 추가된다.

따라서 빈에 따른 EJBObject 바인딩 시간 $B_{EJBObj}(Bean_i)$ 은 다음과 같이 정의할 수 있다.

[정의 5] $B_{EJBObj}(Bean_i)$ =

$(Bean_i$가 세션 빈이면)

$I_{send\_EJBObjReqOfBean\_i} + I_{wait\_Create} + C(Context) + C(InstOfBean_i) + ROM(setSessionContext) + C(EJBObjOfBean_i) + ROM(ejbCreate) + I_{process\_LogicForBindingEJBObj} + I_{send\_EJBObjOfBean\_i}$

$(Bean_i$가 엔티티 빈이고 Create에 의한 바인딩이면)

$I_{send\_EJBObjReqOfBean\_i} + I_{wait\_Create} + \{C(InstOfBean_i) | A(Bean_i)\} + C(EJBObjOfBean_i) + ROM(ejbCreate) + ROM(ejbPostCreate) + I_{process\_LogicForBindingEJBObj} + [I_{accessDB} +] I_{send\_EJBObjOfBean\_i}$

$(Bean_i$가 엔티티 빈이고 Find에 의한 바인딩이면)

$I_{send\_EJBObjReqOfBean\_i} + I_{wait\_Find} + \{C(InstOfBean_i) | A(Bean_i)\} + C(EJBObjOfBean_i) + ROM(ejbFind) + I_{process\_LogicForBindingEJBObj} + I_{send\_EJBObjOfBean\_i}$

4.2.4 빈 메소드 요청 준비시간

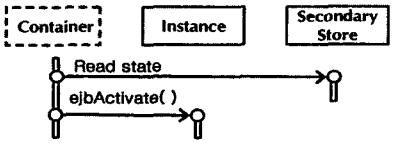
빈 클라이언트가 빈을 사용하기 위해서는 해당 빈의 EJBObject과 바인딩하기 위한 일련의 과정을 거쳐야 빈 메소드를 호출할 수가 있다. 이 과정은 빈이 배치되어 있는 서버 측의 Context 정보를 획득하고, Home 객체와 EJBObject를 바인딩하는 작업으로 이루어진다. 빈 메소드 요청 준비 시간은 원하는 빈의 메소드를 호출하기 위해서 이러한 일련의 작업에 소비되는 시간이다.

i번째 서버 $Server_i$에 배치되어 있는 j번째 빈 $Bean_j$을 요청하길 원한다고 하자. $Server_i$의 Context 정보를 획득하는 시간을 $Context(Server_i)$라고 하면, $Bean_j$ 의 Home 객체와 EJBObject 바인딩 시간은 [정의 4]와 [정의 5]에 따라 $B_{EJBHome}(Bean_j)$, $B_{EJBObj}(Bean_j)$이 된다. 일련의 작업들 중 부분적으로 이미 이루어진 작업이 있다면, 빈 메소드 요청 준비시간을 이루는 요소들은 생략이 된다. 따라서 $Bean_j$ 의 빈 메소드 요청 준비시간 $Prepare(Bean_j)$은 다음과 같이 정의할 수 있다.

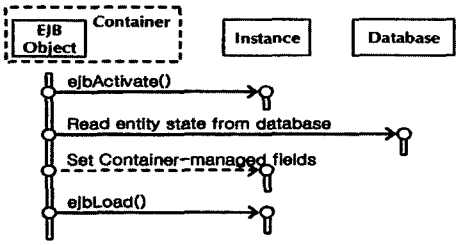
[정의 6] $Prepare(Bean_j) = [[\ Context(Server_i) +]\ B_{EJBHome}(Bean_j\ )+]\ B_{EJBObj}(Bean_j)$

4.2.5 빈 인스턴스의 활성화 시간

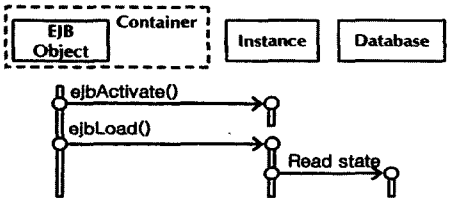
빈을 사용하는 클라이언트 프로그램이 빈 상의 메소드 호출했을 때, 빈 인스턴스가 비활성화되어 있다면, 컨테이너에 의해서 비활성화되어 있는 빈 인스턴스를 활성화한다. 빈 인스턴스의 활성화는 ejbActivate()와 제 2의 저장장치(Secondary Store)로부터 빈의 상태를 읽는 과정을 통해서 이루어지며, 엔티티 빈은 ejbLoad()를 추가로 실행한다. 이 때 제 2의 저장장치는 빈에 따라서 디스크나 데이타베이스가 될 수 있다. 그림 10은 세션 빈과 엔티티 빈의 활성화를 보여주고 있다.



(a) 세션 빈의 활성화



(b) CMP를 이용한 엔티티 빈의 활성화



(c) BMP를 이용한 엔티티 빈의 활성화

그림 10 빈의 활성화 과정

활성화 과정에서 ejbActivate()의 응답시간 $ROM(ejbActivate)$, 해당 빈의 상태를 Secondary Store로부터 읽는 시간 $I_{read\_StateOfBean\_i}$, ejbLoad()의 응답시간 $ROM(ejbLoad)$ 라고 하면, 서버 상의 i번째 빈 $Bean_i$ 의 인스턴스 활성화 시간 $A(Bean_i)$ 는 다음과 같이 정의된다.

[정의 7] $A(Bean_i) = I_{read\_StateOfBean\_i} +$

$ROM(ejbActivate)$(세션 빈이면)
$I_{read\_StateOfBean\_i}$ +
$ROM(ejbActivate)$+$ROM(ejbLoad)$
(CMP 엔티티 빈이면)
$ROM(ejbActivate)$+$ROM(ejbLoad)$
(BMP 엔티티 빈이면)

4.2.6 빈의 응답시간

클라이언트 계층은 서비스 요청을 위해 컨트롤러 계층에 존재하는 빈 상의 메소드를 호출한다. 이 서비스 요청을 받은 빈의 응답시간은 호출되는 메소드의 응답시간이다. 빈은 서비스를 처리하기 위해 다른 빈들을 사용하며, 각 빈들은 필요에 따라서 또 다른 빈들을 요구하기도 한다. 빈과 그 빈을 사용하는 클라이언트의 위치가 원격으로 존재한다면, 그 사이의 모든 전송은 RMI로 이루어진다. 따라서 클라이언트로부터의 빈 상에 있는 메소드 호출에 따른 응답시간은 이끌어지는 빈의 바인딩시간들과 메소드 응답시간들로 구성된다.

그림 11은 클라이언트 프로그램이 빈의 j번째 메소드 Method_j 를 호출하였을 때, 하나의 빈에 대해서 Method_j의 요청이 컨테이너 안의 인스턴스로 전파되어 실행되는 것을 보여주고 있다.

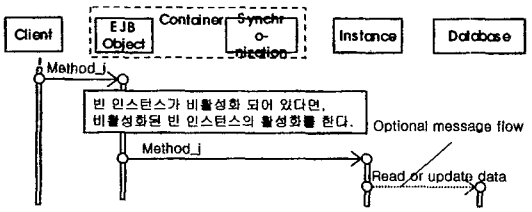


그림 11 빈 상의 메소드 호출

클라이언트 프로그램의 서버의 i 번째 빈 $Bean_i$상의 j 번째 메소드 $Method_j$ 처리에서 의미있는 시간은 클라이언트에서 EJBObject로의 $Method_j$ 요청의 전송시간 $I_{send\_Method\_jReq}$, 빈 인스턴스의 활성화 시간 $A(Bean_i)$, EJBObject에서 메소드를 전파하기 위한 대기시간 $I_{wait\_Method\_j}$, EJBObject의 빈 인스턴스에 대한 $Method_j$ 응답시간 $ROM(Method_j)$, $Method_j$ 실행결과의 전송시간 $I_{send\_resultOfMethod\_j}$ 로 구성되며, 빈 메소드 $Method_j$에 대한 $Bean_i$ 의 응답시간 $RB_i(Method_j)$는 다음과 같이 정의된다.

[정의 8] $RB_i(Method_j) = I_{send\_Method\_jReq} +$
$I_{send\_resultOfMethod\_j} + ROM(Method_j) +$
$I_{wait\_Method\_j}$ $[+A(Bean_i)]$ $[+I_{accessDB}]$

EJBObject에서의 $Method_j$ 의 응답시간 $ROM(Method_j)$은 [정의 3]에 따라 EJBObject에 대한 빈 인스턴스로의 메소드 요청 전송시간과 메소드 실행 결과 반환시간, $Method_j$ 의 실행 시간 $EOM(Method_j)$의 합이다.

클라이언트 프로그램의 $Bean_i$에 대한 서비스 요청이 그림 6와 같이 n개의 빈에 대해서 Msg_1, Msg_2,…, Msg_n으로 분할되어 처리된다면, $Bean_i$의 응답시간 RB_i은 사용된 n개의 빈의 응답시간으로 구성된다. 사용된 n개의 빈에 대해 k번째 빈 $Bean_k$에서 x번째 메소드 $Method_x$를 호출했을 때의 응답시간을 $RB_k(Method_x)$, $Method_x$ 를 요청하기 위한 준비 시간을 $Prepare(Bean_k)$ 라고 하면, $EOM(Method_j)$은 $\sum_{k=1}^{n}([Prepare(Bean_k)+]\sum_{x=1}^{m}RB_K(Method_x)) + I_{process_LogicOfMethod_j}$와 같다.

4.2.7 트랜잭션적인 빈 메소드 실행시간

트랜잭션적인 빈 메소드는 클라이언트 프로그램이 서비스 처리를 위해 하나의 빈에 대해서 그림 12과 같이 일련의 메소드 호출들을 갖는 빈 메소드이다. EJB는 빈에 대해 Bean-Managed Transaction(BMT)과 Container-Managed Transaction(CMT)로 두 가지 유형의 트랜잭션 방법을 지원하며, 세션 빈과 메시지 드리븐 빈은 둘 중 하나를 사용할 수 있고, 엔티티 빈은 CMT를 사용해야 한다. BMT에서는 javax.transaction.UserTransaction 인터페이스를 사용하여 트랜잭션을 수행한다.

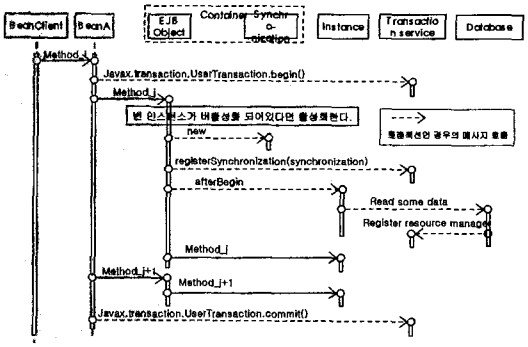


그림 12 트랜잭션 적인 메소드 호출

빈 메소드는 javax.transaction.UserTransaction 인터페이스를 사용하여 트랜잭션의 구획을 나눌 수 있다. 그림 12는 빈 클라이언트 BeanClient가 BeanA의 트랜잭션적인 빈 메소드 Method_i를 호출하여 실행했을 때, Method_i가 javax.transaction.UserTransaction 인터페이스를 이용하여 트랜잭션을 시작한 후, 일련의 빈 메소드 호출들이 발생하는 것을 보여 주고 있다. 이 때, Method_i는 javax.transaction.UserTransaction 인터페이스의 메

소드 begin()과 commit()이나 rollback()을 이용하여 시작과 끝을 결정하며, 그 사이에는 다른 빈에 대해서 일련의 빈 메소드들을 호출한다.

트랜잭션적인 빈 메소드가 실행되기 위해서는 여러 가지 객체들을 생성하고, 그것을 트랜잭션 서비스에 등록하는 트랜잭션 준비단계를 거쳐 메소드 호출의 전파가 이루어진다. 따라서 i 번째 빈 $Bean_i$의 트랜잭션적인 빈 메소드 TMethod가 빈 $Bean_j$상의 n개의 메소드들을 사용하여 실행되었 을 때, 트랜잭션적인 빈 메소드 TMethod의 실행시간은 트랜색션의 시작과 끝을 결정하는 메소드들 begin(…)과 commit(…), rollback(…)의 응답시간인 *ROM(begin)*과 *ROM(commit)*, *ROM(rollback)*을 갖는다. 또한, EJBObject에서의 트랜잭션 준비시간 $I_{prepare\_transaction}$ 과 사용되는 $Bean_j$의 n개의 메소드들의 응답시간, 트랜잭션적인 빈 메소드 TMethod의 자체 로직 처리시간 $I_{process\_LogicOfTMethod}$ 을포함한다. 따라서 i 번째 빈 $Bean_i$의 트랜잭션적인 빈 메소드 TMethod의 실행시간 $EBTM_i(TMethod)$은 다음과 같이 정의될 수 있다.

[정의 9] $$EBTM_i(TMethod) = ROM(begin) + \{ROM(commit) \mid ROM(rollback)\} + I_{prepare\_transaction} + [Prepare(Bean_j) + \sum_{k=1}^{n} RB_j(Method_k)] + I_{process\_LogicOfTMethod}$$

4.2.8 병렬적인 빈 메소드 실행시간

클라이언트 계층의 서비스 요청을 받은 빈은 요청을 위해 다른 빈들을 병렬적으로 사용할 수 있다. 그림 13은 클라이언트의 서비스 요청1이 컨트롤러 계층을 거쳐 비즈니스 계층의 SBean2에서 데이타 접근 계층의 Bean1, Bean2, Bean3를 이용해 병렬적으로 처리하는 것을 보여 주고 있다.

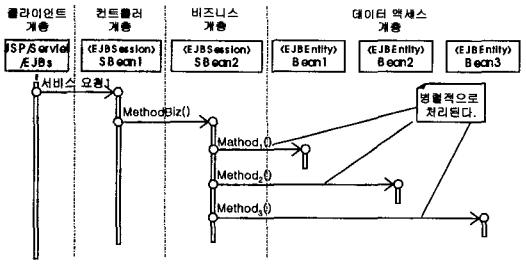


그림 13 병렬적인 빈 메소드 실행

이 때 SBean2의 응답시간은 Bean1, Bean2, Bean3의 응답시간들의 합이 아닌 병렬적으로 처리되는 빈들 중에 가장 큰 응답시간을 포함한다. 따라서 서버에 적재된 i번째 빈 $Bean_i$의 j 번째 메소드 $Method_j$가 n개의 빈을 병렬적으로 사용할 때, $Method_j$의 로직 처리시간을 $I_{process\_LogicOfMethod\_j}$ 라고 하면, 병렬적인 실행을 갖는 메소드 $Method_j$의 실행시간 $EBPM_i(Method_j)$는 다음과 같이 계산된다.

[정의 10] $$EBPM_i(Method_j) = \sum_{k=1}^{n} [Prepare(Bean_k)+] Max(RB_1(\cdots), RB_2(\cdots), \cdots, RB_n(\cdots)) + I_{process\_LogicOfMethod\_i}$$

4.2.9 빈 메소드 자체 실행시간

자체 실행 시간은 워크플로우에 참여하는 빈들에 대해서 하나의 빈이 자체적으로 수행하는 시간 - 다른 빈에서 처리되는 수행시간을 제외한 시간 - 을 말한다. 즉, 클라이언트의 서비스 요청을 사용하는 빈 BaenA가 BeanB를 사용하여 처리할 때, 이 BeanA의 실행 시간은 BeanB에 대한 빈 메소드 요청 준비시간과 BeanB의 응답시간을 제외한 시간이 된다. 이 때 BeanA의 실행시간은 빈을 사용하는 클라이언트의 서비스 요청이 사용되는 빈의 EJBObject에서의 빈 인스턴스에 대한 응답시간이라고 할 수 있으며, 빈 인스턴스에서의 메소드 실행시간과는 다르다.

클라이언트의 서비스 요청을 접수받은 빈은 많은 경우에 다른 빈들을 사용한다. 따라서 워크플로우에 참여하는 각각의 빈에 대한 성능을 측정할 경우에는 사용되는 다른 빈이나 빈들이 수행하는 시간을 제외한 시간만을 보는 것이 적합하다. 그림 14는 Bean1의 자체 실행시간을 나타낸 예로서 클라이언트 계층으로부터 서비스 요청 1을 시스템 측의 Bean1이 받아 들여 Bean2를 이용하여 서비스를 처리하는 것을 보여 주고 있다. 음영으로 나타나는 구간은 Bean1의 자체 실행 시간을 나타낸 것이다.

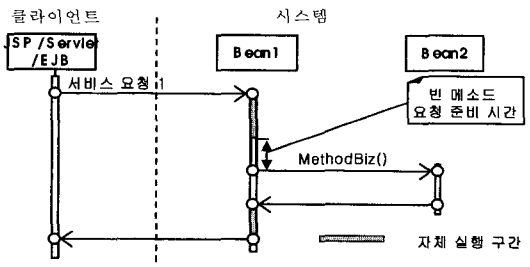


그림 14 자체 실행시간의 예

클라이언트 층의 서비스 요청을 처리하는 i번째 빈 $Bean_i$의 메소드 MethodX가 m개의 빈을 사용하고, 그 중 j번째 빈 $Bean_j$ 에 대해 k개의 메소드를 사용한다고

가정한다. 또한, 병렬적으로 수행되는 n개의 빈이 존재한다고 할 때, 의미있는 시간은 $Bean_i$의 메소드 요청 준비 시간 $Prepare(Bean_i)$과 $Bean_i$의 EJBObject에서 빈 인스턴스의 MethodX의 응답시간 $ROM(MethodX)$와 $Bean_j$의 사용되는 메소드들에 대한 응답시간, 병렬적으로 수행되는 빈들의 응답시간이다. 이를 이용하여 $Bean_i$의 MethodX 자체 실행시간 $BSEOM_i(MethodX)$는 다음과 같이 정의된다.

[정의 11] $BSEOM_i(MethodX) = ROM(MethodX) - \sum_{j=0}^{m}\sum_{k=1}^{x}(Prepare(Bean_j) + RB_j(Method_k)) - Max(RB_1(\cdots), RB_2(\cdots), \cdots, RB_n(\cdots))$

4.2.10 EJB 어플리케이션의 서비스 시간

EJB 클라이언트는 사용자가 요청한 어떤 서비스를 처리하기 위해 EJB 어플리케이션의 기능을 사용한다. 그 결과로 특정 빈의 비즈니스 메소드를 호출하고 그 결과를 처리하여 사용자에게 서비스를 한다. 사용자로부터 하나의 요청을 받은 EJB 클라이언트는 하나 이상의 빈을 이용한다. 따라서 EJB 어플리케이션의 성능을 측정할 때는 사용되는 여러 빈들에 대한 빈 메소드 요청 준비시간들과 빈 메소드들의 서비스 시간들을 구분해서 고려해야 한다. 그림 15는 EJB 클라이언트의 서비스 요청이 EJB 어플리케이션에서 처리되는 것을 보여주고 있다.

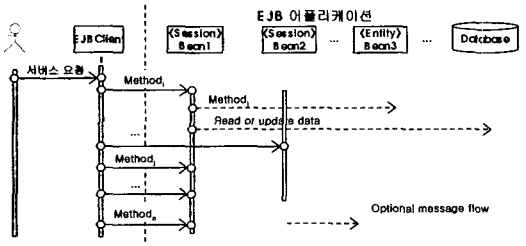


그림 15 EJB 클라이언트의 서비스 요청

EJB 클라이언트 $EJBClient_c$가 사용자에게 서비스하기 위해 EJB 어플리케이션에게 요청을 하였을 때, 사용되는 n개의 빈 중 i번째 빈 $Bean_i$의 빈 메소드 요청 준비 시간은 [정의 6]에 따라 $Prepare(Bean_j)$이다. 또한, $Bean_i$ 에 대해서 호출되는 메소드들이 K개가 존재하고, j번째 메소드 $Method_j$에 대한 응답시간은 [정의 8]에 따라 $RB_i(Method_j)$로 정의된다. 따라서 $EJBClient_c$에 대한 EJB 어플리케이션 서비스 시간은 다음과 같이 정의된다.

[정의 12] $ROEJBAPP(EJBClient_c) = \sum_{i=0}^{n}Prepare(Bean_i) + \sum_{i=0}^{n}\sum_{j=1}^{k}RB_i(Method_j)$

4.2.11 처리량

처리량은 분 당 처리되는 작업량으로 정의되며, EJB 어플리케이션의 처리량 측정은 어플리케이션과 서버, 빈 단위로 나눌 수 있다. 각 단위 별 처리량은 다음과 같이 정의한다.

i 번째 빈 $Bean_i$에서 주어진 시간 I_t (단위: 분) 동안 처리가 완료된 작업량을 W_{bean_i}이라고 하면, 처리량 $TP(Bean_i)$은 다음과 같이 정의된다.

[정의 13] $TP(Bean_i) = W_{bean\_i} / I_t$

i 번째 서버 $Server_i$에서 주어진 시간 I_t (단위: 분) 동안 처리가 완료된 작업량을 W_{server_i}이라고 하면, 처리량 $TP(Server_i)$은 다음과 같이 정의된다.

[정의 14] $TP(Server_i) = W_{server\_i} / I_t$

어플리케이션에서 주어진 시간 I_t (단위: 분) 동안 처리한 작업량을 $W_{Application}$이라고 하면, 처리량 $TP(Application)$은 다음과 같이 정의된다.

[정의 15] $TP(Application) = W_{application} / I_t$

4.2.14 서버 - 빈 - 데이타베이스 사용 메트릭

서버 - 빈 - 데이타베이스 사용 메트릭은 주어진 Work load에 따라 사용되는 서버들과 각 서버에서 사용된 빈들에서 데이타베이스에 대해서 생성, 읽기, 업데이트, 삭제의 빈도를 보여준다.

표 1 서버 - 빈 - 데이타베이스 사용 메트릭

Server \ Workload	Workload#n												
Server #1	Bean A_#1				Bean B_#1				.....	Bean Z_#1			
		R2	U3	D4	C12	R2		D4		C12		U3	D4
Server #2	Bean A_#2				Bean B_#2				.....	Bean Z_#1			
					C12		U3	D4	.....	C12	R2	U3	D4
...	...				...				.....	...			
Server #n	Bean A_#1				Bean B_#1				.....	Bean Z_#1			
	C12	R2	U3	D4	C12	R2	U3	D4	.....		R2	U3	D4

표 1은 이 메트릭에서 사용되는 구성 요소들을 보여주고 있다. 표의 상단에 주어진 Workload와 표의 좌측 수직 칸들은 서버의 이름으로 구성한다. 그리고 각 Server의 행과 Workload의 열이 교차되는 칸들에서 - 한 Server가 갖는 행 - 각 칸은 사용되는 빈의 이름과 그 빈에 대해서 데이타베이스와의 오퍼레이션 유형인 Create(C), Read(R), Update(U), Delete(D) - CRUD 오퍼레이션 유형 -와 각 유형별 발생 횟수로서 구성된다.

이 메트릭을 이용해서 어플리케이션에서 어떤 빈이 데이타 자원(데이타베이스)을 어느 정도 사용하는 지 파악하기 위해 사용한다.

5. 사례연구 및 메트릭 적용 방법

이 장에서는 4 장에서 제안된 메트릭들을 이용하는 사례연구를 한다. 이 실험의 목표는 목표 시스템에 대해서 획득된 성능 데이타를 이용하여 제안된 메트릭의 적용 가능성을 보이고, 메트릭들의 적용방법을 제시하는 것이다. 또한 메트릭의 적용을 통해서 유효성을 검증할 수 있다. 메트릭들을 적용하기 위한 목표 시스템은 트랜잭션 매우 많은 뱅킹(Banking) 어플리케이션이며, 대상 업무로서 은행의 여신 업무를 사용하여 메트릭들을 적용해 본다. 사례연구로는 본 논문에서 제시한 여러 메트릭들 중에 12개의 메트릭을 적용하였다.

5.1 메트릭 적용 환경

메트릭 적용을 위한 시스템 환경은 표 2와 같으며, 뱅킹 어플리케이션과 성능 측정을 위한 클라이언트 프로그램은 동일한 환경에서 구동된다.

표 2 메트릭 적용을 위한 시스템 환경

시스템 환경	
운영체제	Windows XP
자바 가상 머신	JDK 1.3.1
EJB 서버	WebLogic 6.1
데이타 베이스	Oracle 8i

뱅킹 어플리케이션의 S/W 구조는 그림 16과 같이 구성된다. 뱅킹 시스템은 하나의 Managing EJB 서버와 Managing 서버가 관리하는 세 개의 Managed EJB 서버들, 각 Managed EJB 서버에 의해 사용되는 세 개의 데이타 베이스들로 구성된다. Managing EJB 서버는 클라이언트의 요청을 받아들이고 비즈니스 로직을 처리하며, 컨트롤러 계층과 비즈니스 계층의 세션 빈들이 배치된다. Managed EJB 서버에는 데이타베이스와 통신하며, 데이타 접근 계층의 엔티티 빈들이 배치된다.

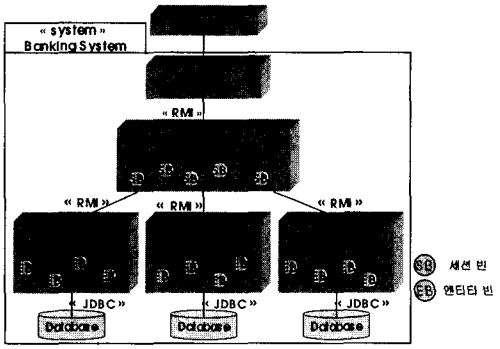


그림 16 뱅킹 어플리케이션의 S/W 구조

5.2 메트릭 적용 사례

본 논문에서는 메트릭 적용을 위해 은행의 여신 업무를 사용한다. 여신업무는 고객을 위해 대출요청에 따른 상담 및 대출 상품 확정, 승인을 하며, 본부심사 결과에 따라 추가적인 서류 및 보증, 담보 등 요구사항을 갖춘 후 대출을 실시하게 된다. 대출 후에는 대출 상품에 따라 주기적으로 발생하는 상환금을 처리하고, 그 밖에 여러 가지의 사후관리 업무를 실시하는 업무이다.

이 여신 업무 중 상환 업무에서 상환이 연체된 대출자에 대한 청구서 생성 워크플로우를 그림 17에서 보여 주고 있다. 이 워크플로우는 상환금이 연체가 된 상환 명세서를 검색하고, 해당 고객에 따른 대출 내역을 확인하여 현재 상환 금액을 청구하는 작업을 나타내고 있다. 그림 17은 클라이언트의 요청에 따라 워크플로우를 구성하는 각 빈들을 3장에서 소개한 EJB 어플리케이션 아키텍처의 계층들인 클라이언트 계층, 컨트롤러 계층, 비즈니스 계층, 데이타 접근 계층으로 배치된 것을 보여준다.

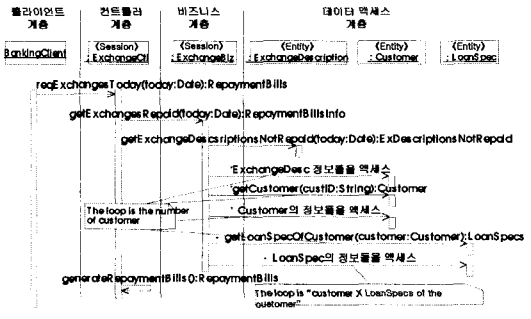


그림 17 연체 상환자에 대한 청구서 생성 워크 플로우

클라이언트 계층의 BankingClient는 Exchange 세션 빈의 EJBObject를 통하여 비즈니스 계층에 있는

ExchangBiz 세션 빈의 getExchangesRepaid(…)를 호출하고, 그 결과로 현재까지 미지불된 상환 청구서 정보들을 받아 클라이언트가 요청하는 형태로 변환하여 반환한다. getExchangesRepaid(…)는 데이타 접근 계층의 엔티티 빈들을 이용하여 연체된 상환 명세서들과 해당 고객들, 상품들을 가져와 상환 청구서 정보를 구성하여 컨트롤러 계층으로 반환한다.

그림 17의 워크플로우를 보면, BankingClient는 요청을 처리하기 위해 뱅킹 어플리케이션의 서비스를 받는다. 이 때 뱅킹 어플리케이션의 서비스 시간은 [정의 12]에 따라 다음과 같이 구성된다.

$$ROEJBAPP(BankingClient) = Prepare(ExchangeCtl) + RB_{ExchangeCtl}(reqExchangesToday)$$

위에서 나타난 뱅킹 어플리케이션의 서비스 시간은 세션 빈 ExchangeCtl의 빈 메소드 reqExchangesToday(…)만으로 구성되는 것을 확인할 수 있다. 따라서 빈 메소드 reqExchangesToday(…)에 대해서 시간 메트릭들을 이용하여 분석한다.

그림 18은 그림 17의 워크플로우를 동시 사용자의 수 100명에 대해서 수행할 때 측정한 데이타이다. 이 그래프의 X축은 어플리케이션 클라이언트와 어플리케이션의 워크플로우를 구성하는 빈들의 빈 메소드들을 나타낸다. 그래프의 Y축은 EJB 클라이언트로서 사용자에게 서비스를 주기 위한 메소드 BankingClient.Main와 세션 빈인 ExchangeCtl, ExchangeBiz의 빈 메소드 reqExchangesToday(…), getExchangesRepaid(…) 에 대해서 호출자에서의 응답시간, 메소드 실행시간, 빈 메소드 자체 실행시간에 대한 측정 값을 ms(Millisecond)로 나타낸 것이다. 빈 메소드 자체 실행시간은 측정된 값들을 기반으로 [정의 11]에 따라 계산되었다.

BankingClient.Main(…)에서 [정의 11]에 따라 계산된 사용자에게 서비스하기 위한 자체 로직 처리는 75.12ms로서 뱅킹 어플리케이션에서 EJB 어플리케이션의 서비스 시간이 대부분을 차지하고 있는 것을 확인할 수 있다.

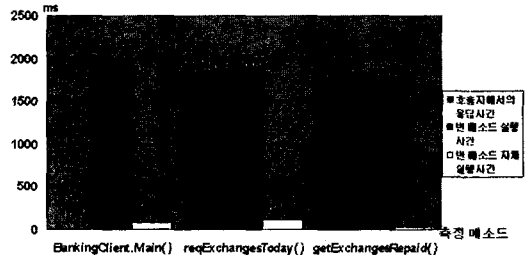


그림 18 워크플로우에 참여하는 빈 메소드의 측정 데이타

세션 빈 ExchangeCtl의 빈 메소드 reqExchangesToday(…)와 세션 빈 ExchangeBiz의 빈 메소드 getExchangesRepaid(…) 항목에서 측정된 빈 메소드 실행시간과 호출자에서의 응답시간과의 차이는 빈 메소드 요청 및 실행 결과의 전송시간과 빈의 활성화 시간, 빈 인스턴스 생성 시간에 의해 발생된다. 이것은 [정의 8]의 빈 메소드 응답시간을 이용해 분석할 수 있다.

빈 메소드 reqExchangesToday(…)는 ExchangeBiz 빈을 이용하여 획득한 지불될 청구서 정보를 기반으로 청구서들을 생성하기 위해 소비된 시간은 120.21ms를 이다. 빈 메소드 getExchagesRepaid(…)는 엔티티 빈들로부터 얻은 정보를 이용하여 지불될 청구서 정보를 생성하기 위해 22ms를 소비하였다. 이 시간은 다른 빈들과 상관없이 자체 로직을 수행하기 위해 소비된 시간이다. 따라서 성능을 향상시키기 위해서는 [정의 1]에서의 인스턴스 생성시간과 [정의 2], [정의 3]에서의 객체 간의 메소드 실행 시간과 메소드 응답시간을 이용하여 빈 메소드 내부에서 발생하는 객체 생성과 메소드 호출을 고려하여 하위 레벨에서의 측정정보를 획득, 분석하고, 분석된 정보를 바탕으로 코드 레벨에서의 최적화를 해야 한다.

그림 18의 그래프에서 빈 메소드 getExchanges Repaid(…)의 실행시간은 1691.93 ms로 응답시간 1708.56 ms의 대부분을 차지하고 있는 것이 확인된다. 자체 실행시간은 22 ms로서 실행 시간의 대부분이 엔티티 빈들에 대한 처리시간으로 소비되는 것을 확인할 수 있다. 빈 메소드 getExchagesRepaid(…)에 대해 세션 빈 ExchangeCtl에서의 응답시간은 [정의 8]에 의해서 다음과 같이 계산될 수 있다.

$$RB_{ExchangeBiz}(getExchangesRepaid) = I_{send\_getExchangesRepaidReq} + I_{send\_resultOfgetExchangesRepaid} + ROM(getExchangesRepaid) + I_{wait\_\ getExchangesRepaidReq} [+A(ExchangeBiz)]$$

이며,

이 때, 빈 메소드 getExchangesRepaid(…)의 실행 시간 $EOM(getExchangesRepaid)$는 정의에 따라 다음과 같이 정의된다.

$$EOM(getExchangesRepaid) = Prepare(ExchangeDescription) + Prepare(Customer) + Prepare(LoanSpec) + RB_{ExchangeDescription}(getExchangeDescriptionsNotRepaid) + RB_{ExchangeDescription}(ExchangeDescription\ 정보들\ 획득) + RB_{Customer}(getCustomer) + RB_{Customer}(Customer\ 정보들\ 획득) + RB_{LoanSpec}(getLoanSpecOfCustomer)$$

+ $RB_{LoanSpec}$ *(LoanSpec의 정보들 획득)*

빈 메소드 getExchangesRepaid(…)의 응답시간에 따라 위와 같이 계산하는 것으로 실행 시간 *EOM (getExchangesRepaid)*을 이끌어 내고, 측정할 성능 요소를 파악할 수 있다. 그림 19는 [정의 8]에 따라 *EOM (getExchangesRepaid)* 에 대해서 빈 메소드 내부의 성능 요소 별 소비 시간을 측정한 것이다. 여기서 상환이 연체된 대출자들은 302명이 검색되었고, 이 대출자들에 대해 빈 메소드는 실행되었다.

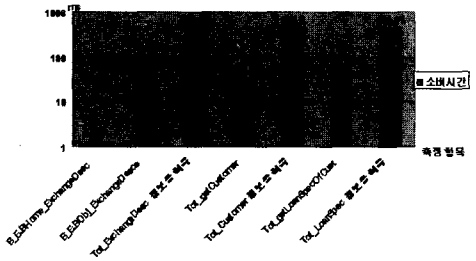


그림 19 getExchagesRepaid(…)의 항목별 응답시간

그림 19의 그래프에서 볼 수 있듯이 빈 메소드 getExchangesRepaid(…)는 엔티티 빈 ExchangeDescrip-tion과 Customer, LoanSpec의 정보 획득시간으로 각각 511.41, 476.6, 464.04로서 많은 시간을 소비하고 있다. 즉, getExchangesRepaid(…)의 성능은 사용되는 엔티티 빈에 많은 부분을 의존하는 것을 확인할 수 있다. 따라서 우선적으로 성능을 향상시킬 수 있는 지점은 엔티티 빈에 대한 정보를 획득하는 과정이다. 엔티티 빈에 대한 정보획득 과정은 네트워크 상의 RMI를 통한 원격 호출로 이루어 진다. 따라서 각 엔티티 빈들에 대해 정보를 획득할 때, 소비되는 시간을 향상 시키기 위해서는 원격 메소드 요청의 수를 최적화시켜야 한다.

ExchangeDescription의 Home 객체 바인딩 시간과 EJBObject 바인딩 시간은 각각 1.3과 110.2로서 측정되었다. Home 객체 바인딩 시간은 다른 요소들 보다 작은 시간을 소비하고 있으며, [정의 4]에서 확인할 수 있듯이 개선할 요소가 그다지 없다. EJBOjbect 바인딩 시간의 구성 요소들은 [정의 5]에서 확인할 수 있다. 성능을 향상시킬 수 있는 요소는 빈 인스턴스의 생성과 활성화 시간을 조정하는 것이다. 빈이 배치된 EJB 서버의 JDBC 연결 풀링의 설정이나 풀링된 빈의 초기 개수, 케쉬될 수 있는 빈의 최대 개수와 같은 서버 측 설정을 통하여 향상 시킬 수 있다. 또한 이 경우처럼 find(…) 메소드를 사용하는 EJBObject의 바인딩의 경우에는 데이타베이스의 튜닝으로 성능을 향상시킬 수 있다.

그림 20은 각 엔티티 빈의 정보 획득을 위해 의존 값 객체를 사용하여 원격 메소드 요청의 수를 최적화하고, 빈에 대한 서버 측 설정을 수정한 후 측정한 빈 메소드 getExchangesRepaid(…)의 항목별 소비시간을 비교한 것이다.

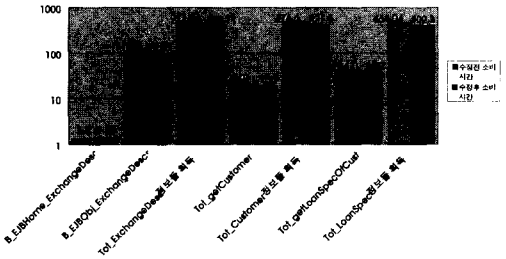


그림 20 수정 후 getExchagesRepaid(…)의 항목별 응답시간 비교

각 항목들에 따라 성능 향상이 이루어졌음을 확인할 수 있다. 엔티티 빈 ExhcangeDescription의 Home 객체를 얻어 오는 시간은 수정 전보다 큰 변화가 없지만, 성능 향상 지점으로 보았던 ExchangeDescription, Customer, LoanSpec 빈의 EJBObject 바인딩 시간들은 각각 22.4 ms와 1.5 ms, 3.97 ms를 더 적게 소비하였고, 정보 획득 시간은 각각 67.7 ms와 55 ms, 61.74 ms로 각 지점에서 많은 성능 향상을 이루었음을 확인할 수 있다.

5.3 메트릭 적용 지침

앞 장에서의 사례연구를 바탕으로 본 논문에서 정의한 성능 메트릭을 적용하는 지침을 소개한다. 표 3은 본 논문에서 제시된 메트릭을 요약한 것이다.

5.2장의 사례연구를 통해서 본 논문에서 정의한 메트릭의 적용을 보였다. EJB 클라이언트는 EJB 어플리케이션의 서비스를 이용하여 사용자의 요청을 처리한다. 이 때 EJB 어플리케이션의 서비스 시간은 [정의 12]에 따라 계산된다. [정의 12]의 어플리케이션의 서비스 시간에 따라 EJB 어플리케이션의 성능요소는 사용되는 각각의 빈들에 대해 빈 메소드 요청 준비 시간과 빈의 응답 시간들로 분할한다. 이에 따라서 EJB 어플리케이션에서 측정되어야 하는 항목들을 결정한다.

빈의 응답시간은 호출되는 빈 메소드의 실행 유형에 따라 빈 메소드 실행 시간을 [정의 2]의 메소드 실행 시간과 [정의 9]의 트랜잭션적인 빈 메소드 실행시간, [정의 10]의 병렬적인 빈 메소드 실행시간 메트릭을 선택하여 사용한다. 빈 메소드 실행시간 메트릭과 [정의 11]의

자체 실행 시간, [정의 6]의 빈 메소드 준비시간과 같은 시간 메트릭들을 이용하여 세부 메트릭으로 분리한다. 이러한 과정을 통하여 빈 메소드 내에서 측정되어야 할 성능 요소를 결정한다.

또한 빈 메소드에서 소비되는 시간을 다른 빈에 의존되는 소비시간과 자체 로직에서 소비되는 시간을 분리하여, 성능 향상 지점을 판단하고, 로직을 최적화 할 것인지 의존되는 빈의 성능을 향상 시킬 것인 지를 결정하여 성능을 향상 시킬 수 있다. 우리는 이것을 5.2장의 메트릭 적용 사례에서 보였다.

표 4는 빈의 응답시간과 빈 메소드 실행시간 메트릭들, 자체 실행시간 메트릭으로부터 획득되는 정보를 보여주고 있다.

메트릭을 이용하여 EJB 어플리케이션의 서비스 시간을 소비하는 각 빈들로부터 성능이상(부하와 같은)이 발견되는 빈을 찾아낼 수 있다. 이것은 서로 다른 서버상

표 3 정의된 메트릭

	설 명
C(Objecti)	i 번째 객체 $Object_i$의 생성 시간
EOM($Method_i$)	객체 O_m이 O_n의 메소드 $Method_i$를 호출했을 때, O_n에서의 $Method_i$가 실행되는 시간과 완료되는 시간 간격으로 나타나는 $Method_i$ 실행시간.
ROM(Mehtod$_i$)	객체 O_m이 O_n의 $Method_i$를 호출했을 때, O_m에서 메소드를 요청하고 요청한 메소드의 실행 결과를 받는 시간 간격으로 나타나는 $Method_i$ 응답시간.
$B_{EJBHome}(Bean_i)$	i 번째 빈 $Bean_i$를 위한 Home 객체의 바인딩 시간.
$B_{EJBObj}(Bean_i)$	i 번째 빈 $Bean_i$를 위한 EJBObject의 바인딩 시간.
Prepare($Bean_i$)	i 번째 빈 $Bean_i$의 빈 메소드를 요청하기 위해 준비하는 시간
A($Bean_i$)	i 번째 빈 $Bean_i$의 인스턴스 활성화 시간.
$RB_i(Method_j)$	i 번째 빈 $Bean_i$ 상의 $Method_j$를 호출 했을 때, Beani의 응답시간.
$EBTM_i$(TMethod)	i 번째 빈 $Bean_i$의 트랜잭션적인 빈 메소드 TMethod의 실행시간.
$EBPM_i(Method_j)$	서비스를 처리하기 위해 다른 빈들을 병렬적으로 사용하는 i 번째 빈 Beani의 j 번째 메소드 $Method_j$의 실행시간.
$BSEOM_i(Method_j)$	서비스를 처리하는 i 번째 빈 $Bean_i$의 j번째 메소드 $Method_j$의 자체실행시간
ROEJBAPP($EJBClient_c$)	클라이언트 $EJBClient_c$가 하나의 요청을 처리하기 위해 EJB 어플리케이션으로부터 서비스 받는 시간
TP($Bean_i$)	i 번째 빈 $Bean_i$의 분 당 처리량.
TP($Server_i$)	i 번째 서버 $Server_i$의 분 당 처리량.
TP(Application)	어플리케이션의 분 당 처리량.
서버 - 빈 - 데이타베이스 사용 메트릭	주어진 Workload에 대해서 사용되는 서버와 빈, 데이타베이스에 대해서 사용 정도를 위한 정보를 포함한다.

표 4 응답시간과 실행시간에 관련된 빈 메트릭들을 이용한 획득 정보

획득 정보	설 명
측정해야 하는 성능 요소 결정	하나의 빈 메소드에 대해 자체 실행 시간, 네트워크 상의 전송시간으로 구분되며, 여러 빈들을 이용할 경우, 각각의 빈에 대해 서비스를 받기 위해 준비하는 시간과 서비스 시간으로 성능 요소가 결정된다.
하나의 빈 메소드에 대한 로직과 의존 되는 빈의 성능 정보 획득	현재 빈 메소드에 대해서 성능 향상 지점을 판단하고, 로직을 최적화 할 것인지 의존되는 빈의 성능을 향상 시킬 것인 지를 결정하여 성능을 향상 시킬 수 있다.
워크플로우에 참여하는 각 빈의 실행시간 정보 획득	EJB 어플리케이션의 워크플로우를 구성하는 각각의 빈들에 대한 성능 정보를 획득하여 다른 시간 요소들과 비교, 분석할 수 있다. 또한 성능 향상지점을 판단할 수 있다.
빈과 빈 클라이언트 간의 전송 속도	빈의 응답시간은 서비스 요청 전송시간과 결과 반환시간을 포함하고 있다. 빈의 응답시간을 구성하는 다른 성능요소와의 비교를 통하여 네트워크간의 시간을 추정해 낼 수 있다.
정확한 빈의 처리시간 정보 획득	빈 클라이언트에 대해서 빈에 대한 요청과 결과는 네트워크를 통하여 전달된다. 전달시간간은 네트워크에 따라서 가변적이므로 전달시간과 처리시간을 분리가 되어있는 응답시간 메트릭을 통해서 빈에서의 처리시간만을 추출할 수 있다.
빈 메소드 요청 준비 시간 정보 획득	빈 메트릭들에는 사용되는 빈 메소드에 대해서 요청 준비 시간 메트릭이 하나의 성능 요소를 고려하였다.
빈 메소드의 실행시간	빈 메소드의 실행 시간을 획득함으로써 클라이언트에 대한 서비스 시간을 소비하는 다른 시간 요소들과의 비교, 분석을 할 수 있다.

에 위치하는 빈들과 빈 클라이언트 사이에 발생하는 네트워크 상의 가변적인 요소들을 추출하고, EJB 어플리케이션의 성능 요소들을 세분화함으로써 가능하다. 본 논문에서 제시하는 메트릭을 사용하여 다음과 같이 성능 향상 방법을 획득할 수 있다.

첫째로 어떤 빈이 성능이상이 발견되었고, 그 빈에 대해 바인딩 시간을 많이 소비한다면, 어플리케이션의 성능향상을 이루기 위해 빈이 운영되는 환경을 조정할 수 있다. 그 빈에 대해 성능향상을 이루어 빈 단위에서의 처리량이 증가할 수 있으나, 보다 큰 단위인 서버단위나 어플리케이션 단위에서의 처리량에 나쁜 영향을 주진 않아야 한다.

두째로 워크플로우에 참여하는 여러 빈들 중에 어떤 빈이 부하가 높고, 그 빈의 서버 – 빈 – 데이타베이스 사용 메트릭에서 높은 CRUD 오퍼레이션 사용이 나타난다면, 불필요한 CRUD 오퍼레이션의 사용을 억제하고 데이타베이스 측에서 성능 최적화를 한다. 또한, 빈을 위한 데이타베이스 연결 풀링이나 자원 할당을 조정하여 해결 할 수 있다.

그리고 워크플로우에서 특정 빈에서의 대기시간이 높고, 그 빈의 서버 – 빈 – 데이타베이스 사용 메트릭에서 낮은 CRUD 오퍼레이션 사용이 나타나거나 나타나지 않았다면, 빈에서 사용하는 서버 측 자원들(데이타베이스 연결 풀링과 같은 자원, 빈을 위해 할당된 자원들)을 조정하거나, 그 빈의 로직의 최적화로 성능을 향상시킬 수 있다.

또한, 서버 – 빈 – 데이타베이스 사용 메트릭을 사용하여 각 빈들에 대해서 발생하는 데이타베이스의 CRUD 오퍼레이션을 파악하여 각 빈들의 데이타 자원 사용률을 분석할 수 있다.

6. 평 가

ISO / IEC 9126는 소프트웨어 제품들을 위한 범용적인 품질 측정 모델로서 국제 표준이다. 또한 이는 EJB 어플리케이션의 성능을 측정하는 데 있어서 매우 유용하다. 그러나 메트릭들은 소프트웨어에 범용적으로 정의되어 있으므로, EJB 어플리케이션의 그대로 사용하기엔 적절치 않다. 이는 메트릭에 EJB 어플리케이션의 특성이 반영되어 있지 않기 때문이다. EJB 어플리케이션의 성능에 적용하기 위해 메트릭의 범위를 재정의 해주고, 여러 메트릭을 추가적으로 정의할 필요가 있었다.

우리는 4장을 통하여 효율성을 구성하는 시간동작성과 자원 활용성 메트릭에 대해서 여러 메트릭들로 재정의하였다. 빈과 그들간의 메시지 호출, 사용되는 객체, EJB 서버 등과 같이 EJB 어플리케이션이 갖고 있는 여러 특징에 따라 객체와 빈 단위로 여러 메트릭들을 특화시켰다. 또한 빈 메소드의 실행 종류에 따라 메트릭을 추가하였다. 즉, 응답시간과 처리시간에 대해서 빈의 종류에 따른 생명주기를 반영하여 각 요소 별로 특성화시켰으며, 처리량에 대해서 빈, 서버, 어플리케이션 단위로 재정의하였다 또한, 자원 활용성 메트릭에는 서버-빈-데이타베이스 사용 메트릭을 재정의하였다 이와 같이 메트릭의 특화는 ISO / IEC 9126 표준의 정의를 기반으로 이루어 졌다.

다음 표 5는 EJB 어플리케이션에 대해서 기존에 사용되던 메트릭과 정의된 메트릭을 다섯 가지 항목에 대해서 비교한 것이다.

C/S 메트릭은 서버 단위에서의 측정뿐만 아니라 운영상태에서의 어플리케이션에서도 성능을 측정하기 위해 이용할 수 있다. 그러나 어플리케이션을 이루는 EJB 빈들과 객체를 측정하기 에는 충분치 않다. 또한, 이러한 측면은 빈 메소드 호출을 비롯해 워크플로우를 이루는 빈들에 대한 측정에 이용할 수 없도록 한다.

객체 지향 어플리케이션의 메트릭은 객체 단위의 측정에 대해 지원한다. 그러나 EJB 빈 단위나 빈 메소드 호출에 대해서는 지원하지 않는다. 따라서 어플리케이션

표 5 기존 메트릭과 정의된 메트릭 비교

	EJB 어플리케이션에 대한 성능 측정					
	서버 단위	빈 단위	객체 단위	빈 메소드 호출	워크플로우에 참여하는 빈들의 측정	운영상태에서의 지원
C/S에서의 메트릭	●					●
객체지향 어플리케이션의 메트릭	△		●			
Liu의 접근	●	●		△	△	
Lladó와 Lüthi의 접근	●	●		●	●	
본 논문의 메트릭	●	●	●	●	●	●

△ : 부분 지원, ● : 지원

내의 워크플로우를 구성하는 빈들에 대한 측정과 운영상태에서의 측정에는 충분하지 않다.

Liu의 접근은 어플리케이션을 이루는 서버들에 대해서 계층화된 Queueing 모델을 사용한 성능분석이 가능하다. 또한 빈 단위 측정과 함께 어플리케이션을 이루는 빈들을 계층화 하여 워크플로우에 참여하는 빈들에 대해서도 측정할 수 있다. 그러나 빈에서 사용되는 여러 객체 대한 측정에 대해 그다지 고려치 않는다. 빈 메소드 호출에 따른 빈 인스턴스의 존재 유무에 따라 발생하는 빈 활성화와 메소드 실행에 따라 사용되는 객체들을 고려치 않는다. 또한, 이 접근은 시스템 성능 예측 모델로 운영상태에서의 측정에는 그다지 적합하지 않다.

Lladó와 Lüthi의 접근은 EJB 어플리케이션의 성능 측정에 대해 여러 측면을 만족하고 있다. 우선 Queueing 모델을 사용하여 서버 단위뿐만 아니라, 빈 단위에서의 측정을 지원하며, 빈 메소드 호출에서 발생되는 빈 활성화나 메시지 전파와 같은 작업을 고려하여 빈 메소드 호출 시의 측정을 할 수 있도록 한다. 그러나 이 접근 또한 시스템 성능 예측 모델로서 운영상태에서의 측정에 불충분하다.

본 논문에서 정의된 메트릭들은 어플리케이션, 서버, 빈 단위뿐만 아니라, 워크플로우를 구성하는 빈들에 대해 어플리케이션 성능에 영향을 주는 여러 요소들을 추출하였다. 이것은 운영상태의 어플리케이션에 대해 어플리케이션에 대한 성능 측정을 비롯해서 어플리케이션을 구성하는 서버, 빈들에 대한 성능 분석을 쉽게 할 수 있도록 하기 위해서이다. 또한, 빈 메소드 호출 메커니즘에 따라 발생하는 빈 인스턴스 활성화와 메시지 전파를 고려하여 워크플로우를 구성한 빈 메소드에 대한 정확한 성능 측정을 할 수 있었다. 각각의 빈들에 대해서 사용하는 빈의 성능과 빈의 자체 로직시간을 구분함으로써 성능이상의 발생지점을 명확하게 판단할 수 있도록 하였다.

7. 맺음말

본 논문에서는 EJB 어플리케이션에서 성능을 측정하기 위해 EJB의 여러 특성에 따른 메트릭을 제시하였다. 어플리케이션 단위에서의 처리량과 주어진 워크플로우에 따른 EJB 어플리케이션의 클라이언트 계층에서의 응답시간으로부터 어플리케이션을 구성하는 서버, 빈들에 이르기까지 세부적인 요소를 메트릭에서 포함하였다.

이러한 요소들은 EJB 어플리케이션은 분산 컴포넌트 어플리케이션으로 동기화, 클라이언트의 RMI를 통한 서비스 사용, 분산을 이루는 각 서버간의 네트워크 장애, 병렬적 처리 등과 같은 성능적 요인에 기인한다. 어플리케이션을 이루는 각 빈들에 메시지가 전달될 때, 발생하는 활성화와 Home 객체와 EJBObject의 바인딩, EJBObject를 통한 메시지 전달 등은 EJB 어플리케이션의 특징들을 충분히 고려하였다.

메트릭을 적용할 수 있도록 하기 위해 빈 단위에서의 여러 유형의 운영 형태를 나누고, 각각에 따른 메트릭을 제시하였다. 또한 5장의 사례연구를 통하여 본 논문에서 제안한 메트릭의 적용방법을 보였고, 유효성을 검증하였다. 우리는 6장의 평가를 통하여 기존의 연구가 분석, 설계단계에서의 성능 예측이 대부분이었으며, 운영 상태의 EJB 어플리케이션의 성능 측정에 충분치 않음을 보였다. 본 논문에서 제시한 메트릭은 국제 품질 표준인 ISO / IEC 9126을 기반으로 확장하였으며, EJB 어플리케이션의 여러 가지 특성들을 충분히 반영하였다. EJB 어플리케이션을 운영하는 시스템 운영자는 본 논문의 메트릭을 이용한 어플리케이션의 성능 측정을 통해 어플리케이션의 병목 부분을 파악할 수 있다 또한, 분석된 정보는 어플리케이션을 구성하는 서버의 튜닝에 사용할 수 있다.

참 고 문 헌

[1] Halter S., Munroe S., *Enterprise Java Performance*, Prentice Hall PTR, Aug. 2000.

[2] Java Community, *ECperf Specification*, Sun Microsystems, at URL : http://java.sun.com/j2ee/ecperf/download.html, May 29, 2001.

[3] Sun Microsystems, *Enterprise JavaBeans Specification, Version 2.0*, at URL : http://java.sun.com/products/ejb/docs.html, Aug. 14, 2001.

[4] Roman Ed., *Mastering Enterprise JavaBeans, Second Edition*, WILEY, 2002.

[5] ISO/IEC JTC1/SC7 Secretariat, *FCD 9126-1,2 Information Technology - Software product quality*, ISO/IEC JTC1/SC7 Secretariat, 1998.

[6] MacCabee M., "Client/server end-to-end response time: real life experience", Proc. 1996 Comput. Measurement Group Conf., Orlando. FL, pp.839-849, Dec. 8-13, 1996.

[7] Fenton N., Pfleeger S., *Software Metrics: A Rigorous & Practical Approach*, PWS Publishing Company, 1997.

[8] Cho E., Kim M, Kim S., "Component Metrics to Measure Component Quality", Asia-Pacific Software Engineering Conference(APSEC2001), Macao, China, PP.419-426, Dec. 4-7, 2001.

[9] 김철진, 조은숙, 김수동, "효율적인 객체지향 설계 및 성능 측정을 위한 정적/동적 메트릭",한국정보과학회논

문지(B), 제 25 권, 제 11 호, pp.1657-1666, 1998, 11.
[10] Liu T., et al., "Layered Queueing Models for Enterprise JavaBean Applications", Fifth IEEE International Enterprise Distributed Object Computing Conference, Sept. 2001.
[11] Lladó C.M., Harrison P.G., "Performance evaluation of an enterprise javabean server implementation.", In: Proc. 2nd, Int. Workshop on Software and Performance (WOSP 2000, September 17-20, Ottawa, Canada), 2000.
[12] Lüthi J., Lladó C.M., "Sensitivity Analysis of an EJB Performance Model using Interval Parameters.", In: Proc. 2. Workshop Performance Engineering in der Softwareentwicklung (PE 2001 , München, Germany, April 18, 2001), pp. 1-5, April 2001.
[13] Lüthi J., Lladó C.M., "Interval Parameters for Capturing Uncertainties in an EJB Performance Model.", In: ACM Performance Evaluation Review, Vol. 29, No.1, Special Issue "Joint International Conference on Measurement & Modeling of Computer Systems (SIGMETRICS 2001/PERFORMANCE 2001 , Camebridge, MA, USA, June 16-20, 2001)", June 2001.
[14] Lüthi J., Lladó C.M., "Splitting Techniques for Interval Parameters in Performance Models." Technischer Bericht (Technical Report) 2000-07, Fakultät für Informatik, Universität der Bundeswehr München , D-85577 Neubiberg, Germany, December 2000.
[15] Hofmeister C., Nord R., Soni D., *Applied Software Architecture*, Addison-Wesley, 2000.

나 학 청
2001년 순천향대학교 전산학과 졸업. 2001년 3월 ~ 현재 숭실대학교 컴퓨터학과 석사과정 재학중. 관심분야는 EJB 소프트웨어 성능 공학, 컴포넌트 개발 기법

김 수 동
정보과학회논문지 : 소프트웨어 및 응용 제 29 권 제 10 호 참조