미들웨어 독립적인 분산 컴포넌트 성능측정 도구 설계

(Design of a Platform Independent Performance Measurement Tool for Distributed Components)

황 길 승 † 이 긍 해 ††
(Kil Seung Hwang) (Keung Hae Lee)

요 약 컴포넌트 기반 소프트웨어에서는 사용될 컴포넌트의 성능이 개발되는 소프트웨어의 품질 확보에 있어서 매우 중요한 요소이다. 컴포넌트 성능에는 흔히 컴포넌트 모델이나 미들웨어에 종속적인 성능측정 도구가 이용된다. 이러한 성능측정 방법에서는 소프트웨어의 개발환경이 변경될 경우 측정 도구도 함께 수정되어야 한다는 문제점을 가지고 있다. 또한, 여러 가지 다른 모델의 컴포넌트들을 한 시스템으로 통합하는 경우에도 유사한 어려움이 존재한다. 본 논문은 이러한 문제에 대한 해결 방법으로 컴포넌트 모델이나 미들웨어에 독립적인 성능측정 방법을 제안한다. 제안된 방법은 미들웨어에 공통적으로 적용 가능한 성능측정기 모델에서 특정 미들웨어를 위한 성능측정기 모델로의 모델변환 과정을 통해 성능데이타간의 상호운용성을 보장한다. 이 방법을 이용하면 동일한 컴포넌트 모델에 따른 컴포넌트들뿐만 아니라 서로 다른 컴포넌트 모델을 기반으로 하는 컴포넌트들에 대한 성능측정이 가능하다.

키워드 : CBD, 컴포넌트 성능측정

Abstract The performance of a component can significantly influence the overall performance of the system being developed in the component based software development. Existing performance measurement tools for components are often built specific to the component model or middleware. One disadvantage of this approach is that when the system is moved to a new platform during the life-cycle of the system, the measurement tools also need to be adapted. A similar problem is observed when components built for different platforms are integrated with a new integration technology like Web Services. This paper presents a new model for measuring component performances, which is independent of particular component models or middleware. This method presents a interoperability for each performance data by model transformation process from middleware independent performance measuring models to middleware dependent performance measuring models. Our model allows objective performance data to be obtained that can be used to compare performances of components based on different component models or middleware platforms.

Key words : CBD, Performance Measuring, Component

1. 서 론

CBD(Component Based Development)는 기능별로 모듈화된 다수의 컴포넌트를 조립하여 소프트웨어를 개발하는 방법으로써 개발자가 빠르게 소프트웨어를 개발할 수 있도록 도와준다. 그리고 재사용성을 향상시켜 향후 소프트웨어의 비즈니스 로직 변경에 유연하게 대응할 수 있도록 해준다.

컴포넌트 기반 소프트웨어 개발에 있어서 개별컴포넌트의 성능은 전체 소프트웨어의 성능에 많은 영향을 준다. 그러므로 개별 컴포넌트의 객관적인 성능을 확인하는 것은 전체 소프트웨어의 성능을 위해 매우 중요하다.

· 본 논문은 과학기술부 한국과학재단 지정 한국항공대학교 부설 인터넷 정보검색연구센터의 지원에 의함
· This research was supported by IRC(Internet Information Retrieval Research Center) in Hankuk Aviation University. IRC is a Kyounggi-Province Regional Research Center designated by Korea Science and Engineering Foundation and Ministry of Science & Technology
† 정 회 원 : 한국전자통신연구원(ETRI) 기반기술연구소
kshwang@etri.re.kr
†† 종신회원 : 한국항공대학교 컴퓨터공학과 교수
khlee@mail.hau.ac.kr
논문접수 : 2003년 12월 4일
심사완료 : 2004년 5월 14일

그러나 현재는 컴포넌트의 성능을 측정하기 위한 표준화된 방법이 존재하지 않을 뿐만 아니라, 다양한 컴포넌트 모델이 존재하기 때문에 성능측정은 특정 컴포넌트 모델에 종속적으로 수행될 수 밖에 없다는 문제점이 있다. 이러한 문제점은 (1)소프트웨어 개발환경의 변경 시 성능측정 및 검증에 필요한 추가비용을 발생시키며 (2)웹 서비스 등으로 대표되는 서비스 통합의 경우에 통합대상이 되는 이질적인 서비스들 사이의 성능확인을 어렵게 할 수 있다.

본 연구에서는 이러한 문제점을 해결하기 위해 분산 컴포넌트의 미들웨어 독립적인 성능측정 방법과 이 방법을 적용한 성능측정기 개발방법을 제안한다. 제안된 방법은 미들웨어에 공통적으로 적용 가능한 성능측정기 모델에서 특정 미들웨어를 위한 성능측정기 모델로의 모델변환 과정을 통해 성능데이타간의 상호운용성을 보장한다. 제안된 방법은 동일한 미들웨어를 기반으로 하는 컴포넌트에 대한 성능비교 뿐만 아니라 서로 다른 미들웨어 기반의 컴포넌트들에 대한 성능측정과 비교가 가능하게 한다.

본 연구의 결과는 서로 다른 컴포넌트 미들웨어 사이의 유연한 성능측정 방법을 제공함으로써 향후 컴포넌트 시장의 활성화 시 객관적인 컴포넌트 선택 기준과 방법을 제시할 수 있을 것으로 기대된다.

논문의 구성은 다음과 같다. 2장에서는 본 연구와 관련 있는 다른 연구들에 관해 설명하고 3장에서는 컴포넌트의 성능측정 아키텍처와 성능측정 요소에 대해 설명한다. 4장에서는 미들웨어 독립적인 컴포넌트 성능측정 모델을 설명하고 5장에서는 구현된 성능측정기를 통한 실험결과를 설명한다. 그리고 6장에서 결론을 맺는다.

2. 관련연구

분산 환경에 배치되어 있는 컴포넌트의 성능을 객관적인 방법으로 측정하기 위한 연구는 다양한 방법으로 시도되어 왔다. 이 장에서는 본 논문의 주제와 관련된 연구 및 제품의 적용사례에 대해서 설명하고 기존 방법들의 한계에 대해 분석한다.

2.1 관련 문헌연구

원격에 위치한 분산 컴포넌트의 성능을 측정하는 방법은 다양하게 연구되어 왔다. 하지만 이러한 기존의 성능측정 방법은 서버의 성능에 영향을 주거나[1-4], 특정 플랫폼이나 제품에 종속적인 경우[5,6], 그리고 성능측정이 특정 컴포넌트 모델에 종속적인 경우[1-8]가 대부분이다.

Baskar 등[5]은 CORBA 플랫폼에서 Visibroker의 Interceptor를 이용하여 컴포넌트 lifecycle에 대한 이벤트를 데이타화 하고 이를 이용하여 성능을 파악하는, 배치된 컴포넌트에 영향을 주지않는 Non-intrusive한 모델을 제시하였다. 이 모델은 독창적인 이벤트 공유 방법을 가지지만 Visibroker를 사용하는 CORBA System에서만 적용할 수 있다는 단점을 가지고 있다.

Adrian Mos와 John Murphy는[4] 대상이 되는 EJB와 같은 JNDI 이름을 가진 proxy EJB의 배치를 통해 Response time이나 Execution time의 데이타를 추출하는 방법을 사용하였다. Client와 대상이 되는 EJB사이에 proxy의 역할을 하는 EJB를 자동생성하여 배치함으로써 Client와 EJB사이의 Response와 Request에 대한 이벤트들의 정보를 가로챌 수 있다. 이 방법은 proxy EJB를 만드는 등의 성능측정을 위한 번거로운 단계들이 존재하고 proxy EJB의 존재가 기존 EJB의 성능에 영향을 준다는 단점을 가진다.

non-intrusive한 컴포넌트 성능측정 방법은 배치된 컴포넌트에 영향을 주지않고 성능측정이 이루어지므로 객관적인 성능데이타의 추출을 가능하게 한다. 본 논문에서 제안하는 성능측정 방법은 원격에서 Black Box방법[9,10]으로 non-intrusive[5]하게 성능측정을 수행한다. 이 방법을 사용하면 컴포넌트의 성능을 클라이언트의 입장에서 객관적으로 파악할 수 있다.

다양한 컴포넌트 미들웨어의 존재에 따른 상호운용성 문제는 OMG(Object Management Group)를 중심으로 해결방법이 연구되어 왔다[11-13]. 더불어 컴포넌트의 성능측정 분야에서도 성능데이타의 상호운용성을 보장하기 위한 연구가 진행되고 있다[14,15]. 하지만 이러한 연구들은 대부분이 WhiteBox 테스팅 기법을 이용하거나[15], 표본 성능데이타를 이용한 시뮬레이션 기법을 사용한다[14].

K.Zielinski 등[15]은 이질적인 환경에서 개발된 다양한 소프트웨어들에 대한 성능을 특정 서버에서 모니터링 할 수 있는 방법을 제안하였다. 이 방법은 성능데이타의 상호운용성에 관한 연구가 오래전부터 진행되어 오고 있었다는 것을 보여주는 논문으로써, 성능코드를 포함하는 프록시를 성능측정 대상이 되는 소프트웨어에 삽입하여 이 프록시와의 통신을 통해 성능을 모니터링 하고자 하였다. 하지만 현재의 분산 컴포넌트 환경에서 이 방법을 적용시키기 위해서는 컴포넌트 개발단계에서 프록시의 역할을 하는 코드를 삽입하여야 하는 과정이 필요하다. 이는 배치된 분산 컴포넌트에 대한 성능을 측정하는 방법으로써는 부적합하다.

위의 방법들은 성능측정 과정이 서버측의 성능에 영향을 주거나 실제 성능측정에 활용되지 못한다는 단점을 가지고 있다. 본 논문에서는 미들웨어 독립적인 성능측정 방법과 구조를 정의하고 개발과정에서 모델과 구현을 분리함으로써 성능측정 결과에 상호운용성을 보장

할 수 있는 모델을 제안한다.

2.2 제품 적용사례

각 컴포넌트 모델을 기반으로 한 대부분의 application server들은 server에 배치된 컴포넌트의 성능을 모니터링하기 위해 일반적인 수준의 성능 모니터링 기능을 제공하고 있다. 하지만 그 중 대부분은 network/protocol 수준에서의 성능정보를 제공하고 있기 때문에 실제로 컴포넌트나 메소드 수준의 성능정보를 요구하는 개발자에게 큰 도움을 주지 못한다. 또한, Application server에서 다양한 성능정보를 제공하기 위한 몇몇의 third party plug-ins[16,17]들을 사용하는 경우도 있지만 이러한 방법은 특정 플랫폼의 특정 Application server를 대상으로 하기 때문에 개발환경 선택의 유연성이 충분히 제공되지 못한다는 단점을 가진다.

이와는 다른 분류로 컴포넌트에 대한 stress test를 수행하거나 각 컴포넌트들의 행위에 대한 정보를 제공하는 component testing tool[18,19]이 있다. 이러한 tool들은 각 client들에 대한 test client program을 자동으로 생성하고 이 script를 동시에 많은 수로 실행함으로써 성능을 모니터링한다. 하지만 이 tool들은 반드시 Application server와 연동되어야 하기 때문에 유연한 testing환경 선택이 어렵다는 단점이 있다. 그리고 이 방법은 컴포넌트의 성능에 영향을 주는 외적인 요소들(i.e., Application server의 성능,network delay, etc.)을 배제하고 오직 server측에서만 성능을 측정하기 때문에 원격의client에서 성능을 판단하기에는 부족한 정보를 제공한다.

이러한 문제점들을 보완하기 위해 본 논문의 성능측정 방법은 Black Box 방식으로 이루어진다. 컴포넌트에 대한 Black Box 방식의 성능측정은 컴포넌트 로직의 성능 뿐만 아니라 network수준의 성능, Application server의 성능을 포함하여 측정된다. 그러므로, 클라이언트 수준에서 신뢰성있고 적용가능한 성능데이타를 산출 할 수 있다.

3. 컴포넌트의 성능측정

3.1 컴포넌트의 성능측정 요소

컴포넌트 모델에서는 다수의 컴포넌트를 연관관계를 통해 하나의 컴포넌트의 형태로 표현하는 것이 가능하다. 그리고 컴포넌트 기반 소프트웨어에서는 컴포넌트의 선택 및 컴포넌트의 연관관계 정의를 통해 전체 소프트웨어가 구성된다. 그러므로 컴포넌트 기반 소프트웨어에 있어서 사용된 컴포넌트의 성능은 전체 소프트웨어의 성능과 밀접한 관련이 있다.

컴포넌트의 성능측정에 있어서 성능측정의 기준이 되는 성능측정 요소의 정의는 중요하다. Adrian Mos[4]의 연구에서는 System의 overhead를 검출하기 위해 Response time을 성능측정 요소로 사용하였고, Sridharan 등[5]의 연구에서는 서버의 상태를 나타내는 성능측정 요소로 CPU Time, Memory Utilization, 그리고 생성된 ORB Thread의 개수를 사용하였다. Precise사의 모니터링 도구인 Precise/Indepth[16]는 시스템의 bottleneck과 overhead를 모니터링 하기 위하여 CPU Usage, Database로의 Transaction time, 그리고 Response Time등을 대표적인 성능측정 요소로 사용하였다. Empirix사의 EJB 컴포넌트 성능모니터링 도구인 BeanTest[18]는 Response Time per Method와 발생한 exception의 수 등을 주요 성능측정 요소로 사용하였다. 이와 같이 많은 유사한 연구 및 제품들이 다양한 형태의 성능측정 요소들을 정의하여 사용하고 있다. 하지만 본 논문의 성능측정 모델은 다양한 성능측정 요소를 가지기 어렵다. 그 이유는 성능측정이 배치된 컴포넌트나 서버측 환경과는 독립적인 Black Box 방식으로 수행되기 때문이다. 그러므로 Transaction Time이나 Exception Count 등의 서버측 컴포넌트의 내부 로직과 관련있는 성능측정 요소는 사용할 수 없다.

따라서 본 논문에서는 Black Box방식의 성능측정에 적용할 수 있고 컴포넌트의 성능을 대표할 수 있는 성능측정 요소를 다음과 같이 정의한다.

1. 메소드별 응답시간(Method Response Time) : 메소드별 응답시간은 컴포넌트의 단위기능의 응답시간이다. 이 요소는 사용자가 요구하는 컴포넌트의 기능별 성능을 나타내는 기준이 된다. 메소드별 응답시간은 메소드가 호출된 시점에서부터 메소드 결과값이 리턴된 시점까지의 시간의 차를 의미한다.

$$T_{MRT} = T_{ODT} + T_{EXT}$$

T_{MRT} : 메소드 응답시간

T_{ODT} : 외부요인(Network delay등)에 의한 시간적 손실

T_{EXT} : 컴포넌트 내의 메소드 로직 수행시간

2. 컴포넌트 응답시간(Component Response Time) : 컴포넌트 응답시간은 컴포넌트 인스턴스 생성시간을 포함한 컴포넌트 내의 모든 비즈니스 메소드의 응답시간의 합이다. 이것은 컴포넌트의 전체의 성능을 평가할 수 있는 요소로서의 의미를 가진다.

$$T_{CRT} = T_{CIT} + \sum_{i=0}^{n} T_{MRTi}$$

T_{CRT} : 컴포넌트 응답시간

T_{CIT} : 컴포넌트의 인스턴스를 생성하는 시간

n : 컴포넌트의 메소드의 개수

본 논문의 컴포넌트 성능측정 모델은 위에서 설명한 두가지 성능측정 요소를 적용하여 성능측정을 수행한다.

성능측정의 결과로 산출되는 성능데이타는 메소드별 응답시간과 컴포넌트 응답시간을 통한 기능별 성능을 포함한다.

3.2 분산 컴포넌트 아키텍쳐

CBD기반 개발방법은 소프트웨어의 기능을 구성할 때 고유한 기능을 가지는 소프트웨어 컴포넌트를 사용한다. 분산 응용을 지원하기 위해 제안된 분산 컴포넌트 모델에서는 원격 컴포넌트를 간단히 호출할 수 있도록 하기 위해서 표준화된 아키텍쳐를 가진다. 일반적인 구조는 그림 1과 같다.

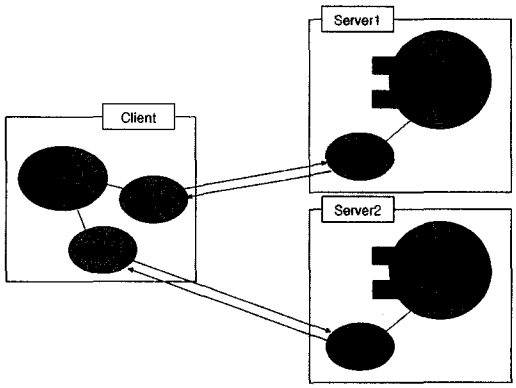

그림 1 분산객체 아키텍쳐

개별 분산 컴포넌트는 두 가지 필수 구성요소를 가진다. 비즈니스 메소드들의 명세를 포함하는 컴포넌트 인터페이스, 그리고 서버와 클라이언트 사이의 통신을 담당하는 proxy가 그것이다. 컴포넌트가 두가지 구성요소를 포함한다는 것과 이 구성요소를 이용한 클라이언트와의 통신방법은 컴포넌트 모델의 종류에 관계없이 유사하다[20,21].

3.3 분산 컴포넌트의 성능측정 방법

분산환경에서 배치된 컴포넌트에 접근하여 컴포넌트의 메소드를 호출하기 위해서는 각각의 컴포넌트 모델들이 정의하는 명세를 준수하여야 한다. 분산 컴포넌트 서버에 접근하기 위한 통신 프로토콜, 컴포넌트의 바인딩을 위한 절차 등은 컴포넌트 모델마다 다르게 정의하고 있다. 하지만 추상적인 단계에서 볼 때, 컴포넌트로의 접근 방식은 특정 컴포넌트 모델에 상관없이 유사하다.

예를 들어 J2EE서버에 배치되어 있는 EJB 컴포넌트에 접근하기 위해서는 RMI/IIOP 프로토콜을이용하여 서버에 접근하고 JNDI로 컴포넌트를 식별하며 홈인터페이스와 원격인터페이스의 인스턴스 생성을 통해 메소드를 호출할 수 있다. 마찬가지로 .NET 서버에 배치되어있는 DCOM 컴포넌트에 접근하기 위해서는 TCP 프로토콜을 이용하여 서버에 접근하고 CoCreateInstance 메소드를 호출하여 컴포넌트의 인스턴스를 생성하여 Operation을 호출한다.이러한 과정 중에서 Stub을 통한 원격 컴포넌트로의 바인딩이나, 컴포넌트의 인터페이스를 통한 메소드의 호출은 컴포넌트 모델의 종류에 관계없이 유사한 부분이다.

본 논문에서는 특정 분산 컴포넌트 모델에 독립적으로 분산환경의 컴포넌트에 대한 성능측정을 수행할 수 있는 컴포넌트 성능측정 아키텍쳐 및 성능측정기의 모델을 제안한다. 이 아키텍쳐는 컴포넌트 모델들이 동일하게 가지고 있는 추상적인 단계의 접근방법을 사용한다. 성능측정을 위한 성능측정 아키텍쳐는 그림 2와 같다.

분산 컴포넌트에 접근하기 위해서 클라이언트에는 반드시 접근대상이 되는 컴포넌트의 인터페이스와 스텁이 존재하여야 한다. 성능측정기는 컴포넌트의 메소드들이 명세되어 있는 인터페이스 파일을 분석하여 메소드들의 정보를 추출하고 이를 이용하여 성능측정을 위한 테스트 클라이언트 프로그램을 생성한다. 이 때 컴포넌트의 메소드를 테스트 프로그램에서 호출하기 위해서 메소드의 파라미터들의 타입을 분석하고 해당하는 테스트 데이타들을 자동 생성하여 테스트 프로그램에 적용한다. 테스트 프로그램은 특정 컴포넌트 모델에 종속적인 소스코드들로 이루어져 있으며 컴포넌트 모델의 종류에 따라 다른 형태의 성능측정 소스코드들이 삽입된다. 이 테스트 프로그램이 실행됨으로써 성능측정 결과가 산출되고 이 데이타를 특정 repository에 저장하면서 성능측정을 위한 프로세스는 종료된다. 성능측정 요소별 라이브러리는 테스트 프로그램에서 실제로 성능측정을 수행하기 위해 필요한 기능들이 존재하는 클래스로써 테스트 프로그램 상에서 호출되어 실행된다. 이것은 성능측정기와는 독립적으로 소프트웨어 라이브러리의 형태로 존재하며 모델링 레벨에서 별도로 다루어지지는 않고 구현 단계에서 바인딩 되는 형태를 가진다.

성능측정 도구는 크게 다음과 같은 모듈들로 나뉘어진다.

1. 인터페이스 분석기 : 인터페이스 분석기는 컴포넌트의 인터페이스 파일을 분석하여 테스트 프로그램을 생성하기 위해 필요한 메소드들에 대한 정보(i.e. 메소드이름, 리턴타입, 파라미터 정보등)를 추출하는 역할을 한다.
2. 테스트 데이타 생성기 : 테스트 데이타 생성기는 인터페이스 분석기에서 분석한 메소드의 파라미터 정보를 바탕으로 메소드 호출에 필요한 파라미터 타입에 따른 데이타들을 생성하고 이를 테스트 프로그램에서의 메소드 호출시 적용시킨다.
3. 테스트 프로그램 생성기 : 테스트 프로그램 생성기는 인터페이스 분석기에서 분석한 컴포넌트의 메소드 정

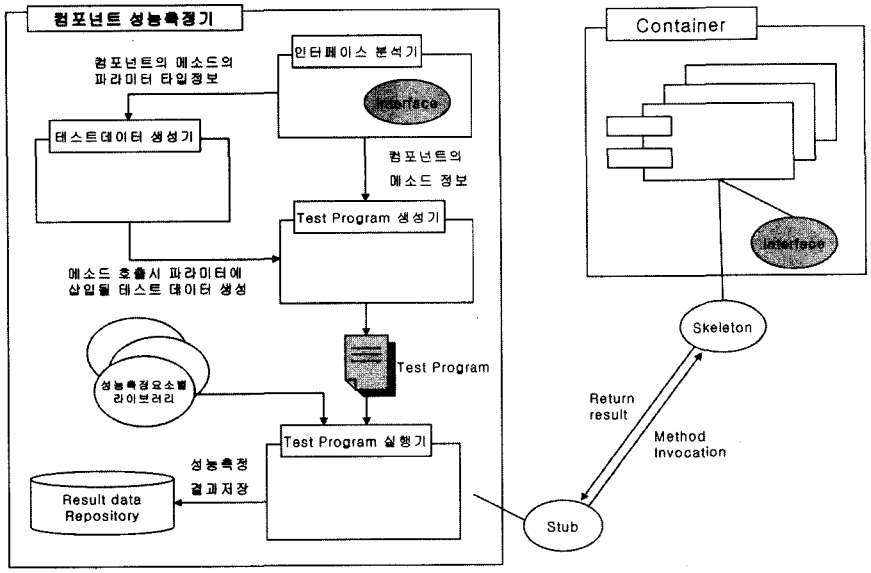

그림 2 분산 컴포넌트의 성능측정 방법

보들과 테스트 데이타 생성기에서 생성한 파라미터 데이타, 그리고 컴포넌트에 접근하기 위한 클라이언트 코드의 템플릿들을 조합하여 성능측정 코드가 삽입된 테스트 클라이언트 프로그램을 생성한다.

4. 테스트 프로그램 실행기 : 테스트 프로그램 실행기는 테스트 프로그램 생성기에서 생성된 테스트 프로그램을 컴파일하고 성능측정을 위한 라이브러리들을 링크하여 실행하고 실행후 수집된 테스트 결과 데이타를 특정 repository에 저장한다.
5. 결과 표시기 : 수집되어 저장된 테스트 결과 데이타를 사용자가 이해하기 쉬운 그래프의 형태로 표현하는 역할을 한다.

4. 성능측정기 개발 프로세스

본 논문에서 제안하는 성능측정기의 개발방법은 Unified Software Development Process[22]를 기반으로 한다. 개발방법은 다음과 같은 과정을 가진다.

1. 개발에 필요한 요구사항을 수집하고 명세화한다.
2. 요구사항 명세를 usecase별로 분류하고 분석하여 Requirement Analysis Model(RAM)을 작성한다.
3. RAM을 상세화, 구조화하여 Platform Independent Model(PIM)을 생성한다.
4. PIM을 기초로 개발될 소프트웨어의 특성에 맞는 기술 종속적인 요소를 더하여 Platform Specific Model(PSM)을 생성한다.
5. 생성된 PSM에서 수동 또는 자동으로 코드를 생성하여 소프트웨어를 완성한다.

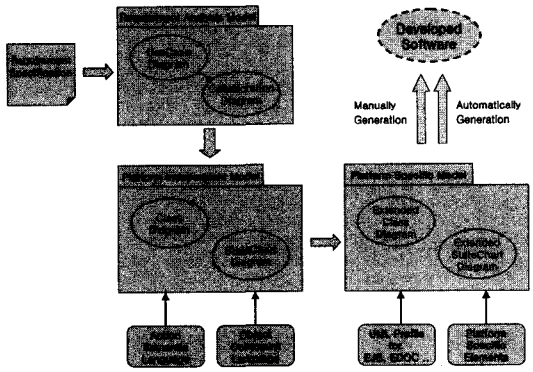

그림 3 성능측정기 개발 프로세스

성능측정기 개발 프로세스에서는 생성된 모델을 다양한 컴포넌트 모델로 적용하기 위해서 모델링 단계를 플랫폼 독립모델과 종속모델로 구분하였다[11-13]. 이 구분은 성능측정 대상이 되는 모든 컴포넌트 모델에 적용 가능한 모델과 특정 컴포넌트 모델에 종속적인 상세모델로 구분함으로써 상호운용성과 확장성을 보장할 수 있다.

4.1 성능측정기의 요구사항 분석모델

성능측정기를 설계하기 위해서 구체적인 행위나 관계를 모델링하기 전에 우선 성능측정기의 목적과 요구사항, 요구되는 모듈 등을 대략적으로 서술하고 이러한 것들을 단계적이고 효율적으로 정리하는 과정이 필요하다. 실제로 성능측정기에 요구되는 요구사항들과 대략적인 프로세스를 구조화하여 사용자의 관점에서 본 전체 시

스템의 그림을 그리는 것은 중요하다. 성능측정기에 대한 요구사항은 그 후의 모델링이나 구현에 그대로 반영되며 정확하고 체계적인 모델링 및 구현을 위해서는 정확한 요구사항 분석이 필요하기 때문이다[22,23].

비즈니스 모델링 단계에서 컴포넌트 성능측정기에 요구되는 조건들은 다음과 같은 몇 가지로 정리된다.

1. 컴포넌트의 성능측정 수행방법이 정의되어야 한다.
2. 성능측정에 요구되는 데이타의 수집방법이 정의되어야 한다.
3. 성능측정 요소들에 대해 정의되어 있어야 한다.
4. 성능측정 요소별 성능측정값 수집방법이 정의되어 있어야 한다.
5. 성능측정 결과를 어떻게 표시할 것인지 정의되어 있어야 한다.

위의 요구조건에 따른 비즈니스 모델의 구조화된 형태는 그림 4와 같은 usecase diagram으로 표현될 수 있다. Client라는 액터에 의한 성능측정 usecase는 다섯 가지의 세부 usecase를 포함한다. 이러한 usecase는 성능측정기 전체 프로세스중 각각의 하위 프로세스로 정의할 수 있으며 연관되어 동작하는 소프트웨어 모듈로 나타내어 진다. 그리고 모델을 상세화할 때 각각의 서브 시스템으로 표현된다. 그림 4의 usecase diagram을 상세화하고 각 객체들 사이의 행위들에 대한 기초적 명세를 위해 그림 5과 같이 분석모델을 정의한다.

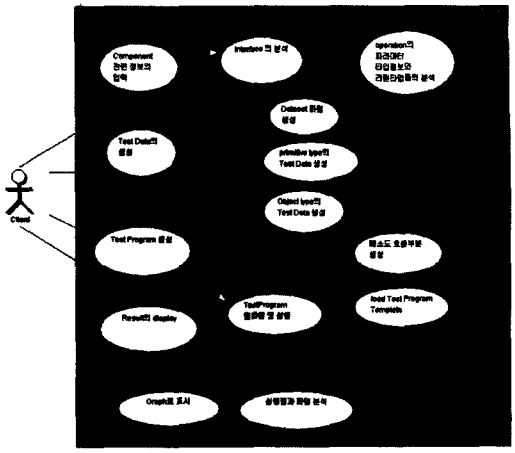

그림 4 컴포넌트 성능측정기의 usecase Diagram

4.2 성능측정기의 플랫폼 독립모델

플랫폼 독립모델(PIM)은 소프트웨어의 정적인 모습과 동적인 모습을 추상적으로 표현한다. 성능측정기의 PIM을 정의하는 것은 성능측정 대상이 되는 컴포넌트 모델에 상관없이 적용 가능한 도메인 표준모델을 제시한다는 점에서 중요하다.

PIM은 UML표기법으로 표현될 수 있다. 소프트웨어 시스템의 정적인 모습은 그림 6과 같이 Class Diagram으로 표현하고 상태변화와 상태에 따른 행위들을 표현

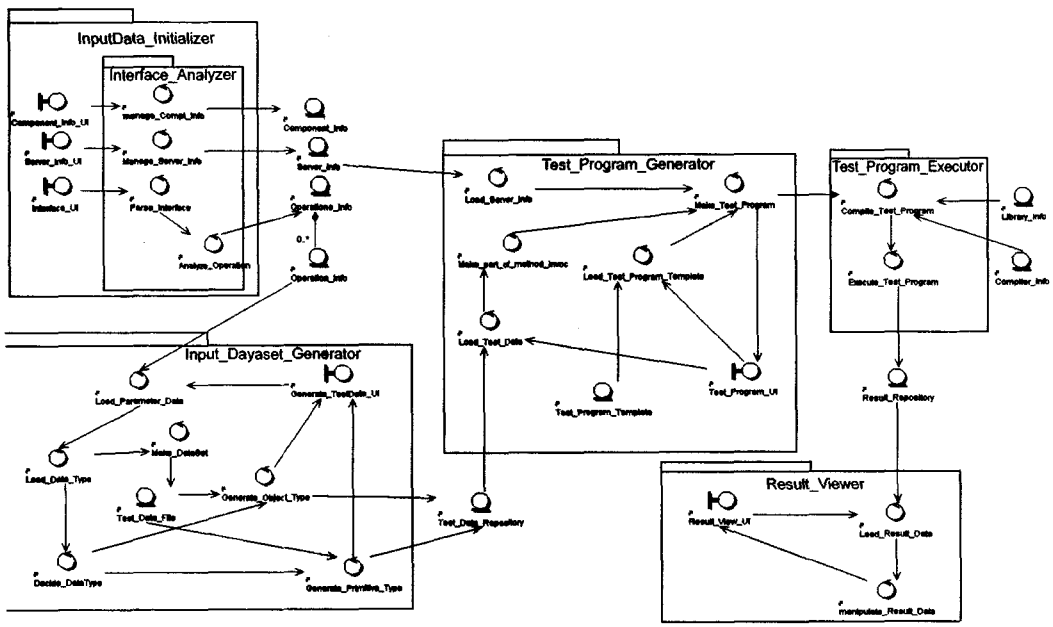

그림 5 컴포넌트 성능측정기의 분석모델

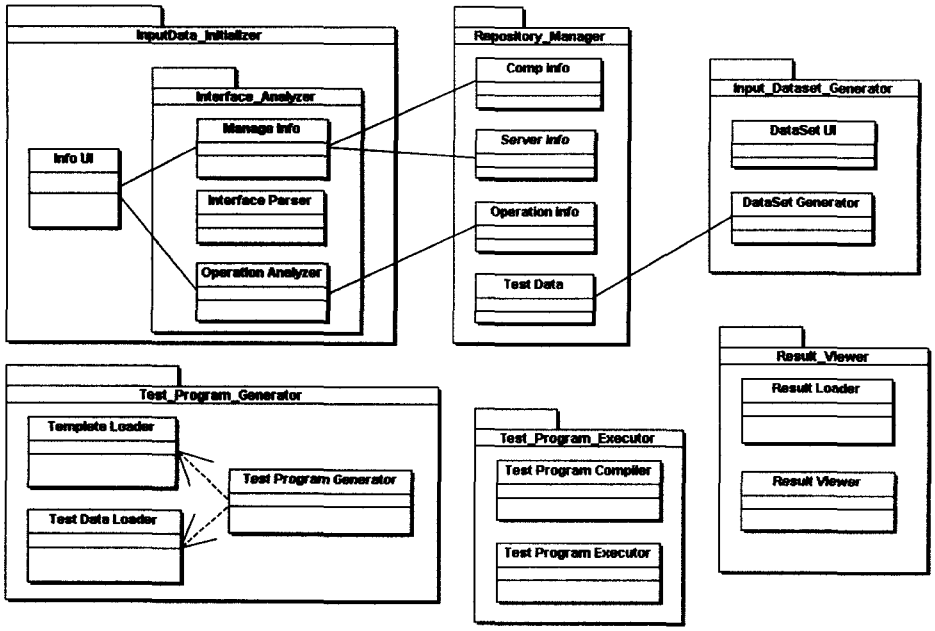

그림 6 컴포넌트 성능측정기의 Class Diagram

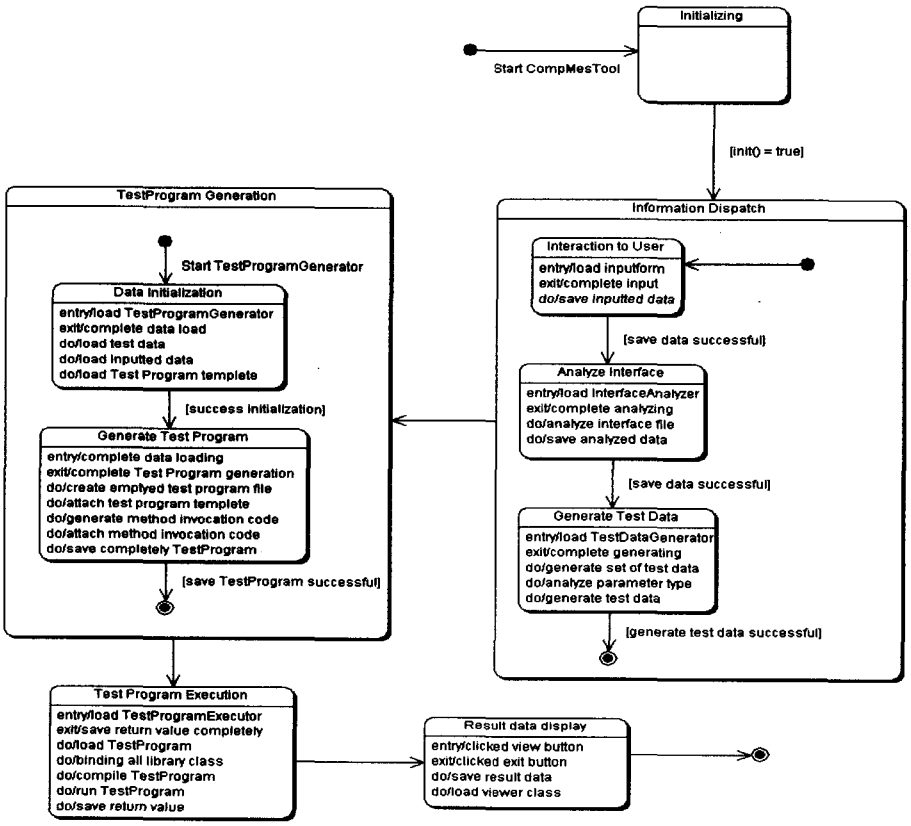

그림 7 컴포넌트 성능측정기의 State Diagram

한 동적인 모습은 그림 7과 같이 State Diagram과 Action semantic으로 나타낸다[24]. 이런 diagram들을 통해서 컴포넌트 성능측정기의 구조와 관계, 행위, 상태의 변화등을 알 수 있다.

4.3 성능측정기의 플랫폼 종속모델

PIM은 특정기술에 독립적으로 작성된다. PIM이 완료되면 좀 더 상세한 부분으로의 접근, 즉 플랫폼 종속적인 요소들을 첨가하여 PSM을 생성하게 된다.

컴포넌트 성능측정기는 특정 분산 컴포넌트 미들웨어를 기반으로 하지 않는다. 그러므로 PSM으로의 매핑시 특별한 UML Profile을 필요로 하지 않는다. 하지만 어떤 분산 컴포넌트 플랫폼을 성능측정의 대상으로 하느냐에 따라 다른 PSM으로의 매핑 구조를 가지게 된다. 컴포넌트 성능측정기의 PIM에서 PSM으로 매핑할 때 적용되어야 하는 요소들은 다음과 같다.

- 성능측정시 사용자로부터 수집되어야 하는 데이타의 종류
- 성능측정시 성능측정 대상이 되는 컴포넌트의 인터페이스 분석방법
- 테스트 프로그램 생성시 language의 종류
- 성능측정시의 연결정보(프로토콜, 포트 등)
- 성능측정 요소별로 로딩되는 성능측정 라이브러리
- 테스트 프로그램에서 컴포넌트의 호출을 위해 생성되는 테스트 데이타의 타입
- 테스트 프로그램 컴파일 및 실행시 필요한 컴파일러의 종류
- 테스트 프로그램 컴파일 및 실행시 필요한 라이브러리

성능측정기의 PIM에 위에서 열거한 요소들을 적용하여 상세화하면 성능측정기의 PSM을 수동으로 생성할 수 있다.

그림 8은 성능측정기에서 데이타 입력과 인터페이스 분석을 담당하는 모듈의 PSM이다. 그림 8의 PSM은 EJB 컴포넌트 성능측정을 위한 성능측정기를 설계할 경우에 작성될 수 있는 예이다. 이 PSM에 적용된 기술 종속적인 요소는 다음과 같다.

- 분산 컴포넌트로의 접근을 위한 서버정보(EJB 서버 종류, 연결 프로토콜, 포트 등)
- 컴포넌트 인터페이스정보(Remote, Home Interface 등)
- 인터페이스 분석방법 (Java Reflection 등)
- 데이타 타입(String, Vector 등)

4.4 시스템 구현

정의된 PIM으로부터 특정 컴포넌트 모델의 PSM으로의 매핑을 통하여 서로 같은 구조와 같은 성능측정 요소를 적용한 성능측정기를 구현할 수 있다. 각각의 컴포넌트 모델에 따라 구현된 성능측정기들에서 산출되는 성능데이타는 동일한 성능측정 과정과 구조, 그리고 동

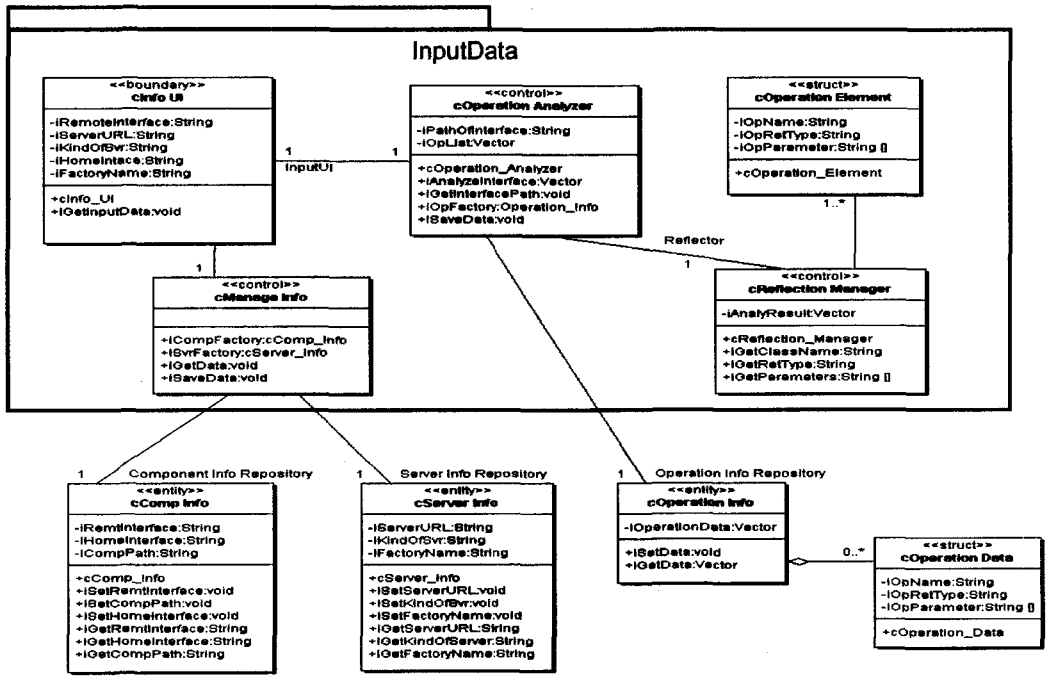

그림 8 컴포넌트 성능측정기의 PSM

일한 성능측정 요소들을 가지고 있기 때문에 객관성을 가질 수 있다. 그리고 PIM 레벨에서의 데이타 타입이 동일하기 때문에 상호 비교가 가능하다.

미들웨어나 컴포넌트 모델에 독립적인 컴포넌트 성능측정기는 다양한 분야에서 활용될 수 있다.

우선, 개발환경의 변화에 따른 비용소비를 줄일 수 있다. COTS(Commercial Off The Shelf)등을 사용하여 소프트웨어를 구성할 때, 개발될 소프트웨어의 성능을 위해서 컴포넌트의 성능측정 및 검증은 중요한 요소이다. 그러므로 미들웨어 독립적인 성능측정기는 개발환경이 변화하였을 때 성능검증에 필요한 비용을 절감할 수 있다.

그리고, Web Service등을 이용한 서비스 통합의 경우, 컴포넌트 모델이 다른 이질적인 컴포넌트들을 wrapping하고 조합하여 하나의 서비스로 구성할 수 있다. 이 때, 우수한 성능을 가진 서비스를 선택해서 구성하기 위하여 이질의 컴포넌트 사이의 성능비교 수행이 요구되므로 미들웨어 독립적인 성능측정이 필요하다.

이러한 컴포넌트 성능측정기의 미들웨어 독립성은 컴포넌트 성능측정이라는 도메인에 공통적으로 적용할 수 있는 모델을 정의함으로써, 다양한 컴포넌트 성능측정기간의 상호운용성을 보장하는 방법으로 활용될 수 있다는 의미를 가질 수 있다.

5. 구현 및 실험결과

5.1 실험환경

가장 대표적인 컴포넌트 모델인 SUN Microsystems사의 J2EE(Java 2 Enterprise Edition), OMG(Object Management Group)의 CCM(CORBA Component Model), 그리고 MicroSoft사의 DCOM(Distributed Component Object Model)을 성능측정 대상 컴포넌트 모델로 하였다. 성능측정의 대상이 될 컴포넌트는 대표적인 정렬 알고리즘 4가지를 구현한 컴포넌트를 사용하였고 컴포넌트의 실행에 이용된 데이타는 9,350Byte의 임의의 문자열을 사용하였다.

성능측정 대상 컴포넌트들은 동일한 명세를 기반으로 하였다. 즉, DCOM과 CORBA는 동일한 IDL을 바탕으로 하였고 작성된 IDL을 기준으로 하여 EJB의 Remote Interface를 작성하였다. 그리고, 서버의 시스템 환경이 클라이언트에서의 성능측정에 어떤 영향을 주는지를 알아보기 위하여 컴포넌트가 배치된 서버의 환경에 약간의 차이를 두었다. CORBA와 J2EE의 성능측정은 원격의 클라이언트에서 수행되었고 DCOM의 성능측정은 로컬 클라이언트에서 수행되었다. 실험에 사용된 시스템 환경과 개발환경은 다음과 같다.

표 1 실험환경

시스템 환경(Server side)	
CPU	Pentium 4-1.8Ghz
Memory	576Mbyte
Connection	100Mbit LAN
개발환경	
J2EE (Remote)	jdk1.3.1_08, j2sdkee1.3.1_02
DCOM (Local)	Visual Studio 6.0, C++
CORBA (Remote)	Visibroker 4.0, java

5.2 실험결과

작성된 PSM들을 기반으로 하여 성능측정기를 구현하였다. 각 성능측정기들은 성능측정을 위한 테스트 프로그램을 개발환경에 맞게 생성하고 이 테스트 프로그램을 실행함으로써 성능측정을 수행한다. 실험은 정확한 결과의 도출을 위해서 각 컴포넌트의 메소드를 10회 반복실행하여 메소드 응답시간의 평균값을 산출하였다. 그리고 컴포넌트의 인스턴스를 10회 생성하여 그 평균값을 컴포넌트 응답시간에 반영하였다. 메소드별 응답시간과 컴포넌트 응답시간의 결과는 표 2와 같다.

몇번의 반복된 실험을 통해서, 각 PSM에 따라 구현

표 2 응답시간의 실험결과

EJB 성능측정 결과 (단위:ms)				
	BubbleSort	HeapSort	InsertionSort	MergeSort
메소드별 응답시간	13227.375	13625.125	13514.000	13314.000
컴포넌트 응답시간	13428.375	13816.375	13731.000	13503.500
DCOM 성능측정 결과 (단위:ms)				
	BubbleSort	HeapSort	InsertionSort	MergeSort
메소드별 응답시간	969	165	297	161
컴포넌트 응답시간	1048	211	328	224
CORBA 성능측정 결과 (단위:ms)				
	BubbleSort	HeapSort	InsertionSort	MergeSort
메소드별 응답시간	19041.666	12773.500	14531.000	12686.000
컴포넌트 응답시간	19666.666	13335.500	15171.000	13342.000

표 3 Application Server의 부하에 따른 성능결과

동시측정시의 EJB 성능측정 결과 (단위:ms)				
	BubbleSort	HeapSort	InsertionSort	MergeSort
메소드별 응답시간	13227.375	13625.125	13514.000	13314.000
컴포넌트 응답시간	13428.375	13816.375	13731.000	13503.500
개별 측정시의 EJB 성능측정 결과 (단위:ms)				
	BubbleSort	HeapSort	InsertionSort	MergeSort
메소드별 응답시간	2182	2611	2452	2167
컴포넌트 응답시간	2301	2813	2671	2360

된 성능측정도구들은 서로 다른 컴포넌트 모델을 기반으로 하는 동일한 기능의 컴포넌트들로부터 비교가능한 성능데이타를 성공적으로 산출한다는 것을 확인할 수 있었다. 그리고 Client측에서 확인할 수 있는 컴포넌트의 성능 결과값에는 컴포넌트의 내부 로직의 성능 뿐만 아니라 컴포넌트의 성능에 영향을 주는 다양한 외적 요소들이 작용한다는 것을 알 수 있었다.

Black Box 방식으로 성능측정을 수행할 때 Network delay 또는 Application Server의 성능 등의 요소들이 산출되는 컴포넌트의 성능에 영향을 준다. 표 2의 실험결과를 볼 때, 로컬 클라이언트에서 수행한 DCOM의 성능측정 결과가 리모트의 클라이언트에서 수행한 EJB나 CORBA의 성능측정 결과보다 우수한 것으로 나타났다. 이 결과는 클라이언트에서 컴포넌트의 성능을 판단할 때 Network의 속도나 delay 등이 중요한 요소가 될 수 있다는 것을 의미한다. 그리고 표 3과 같이 Application Server에 임의로 부하를 가했을 때의 성능측정 결과와 그렇지 않은 때의 성능측정 결과값이 차이가 나는 것도 확인할 수 있었다.

6. 결 론

본 논문에서는 분산환경에서 배치된 컴포넌트의 성능과 측정하는 방법을 제안하였고 두가지 성능측정 요소를 정의하였다. 그리고 이를 적용하여 분산 컴포넌트의 성능을 측정할 수 있는 컴포넌트 성능측정기의 플랫폼 독립적인 모델을 정의하였고 구현을 통하여 성능측정기의 모델이 플랫폼에 따라 구현되고 실행될 수 있다는 것을 증명하였다. 또한, 각 플랫폼에 종속적으로 구현된 컴포넌트 성능측정기는 서로 다른 미들웨어 및 컴포넌트 모델 간에 상호 운용 가능한 성능데이타를 산출하는 것을 실험을 통해 확인하였다.

컴포넌트 기반 어플리케이션을 위해서는 높은 성능을 가지는 컴포넌트의 선택이 무엇보다 중요하다. 이를 위해서는 객관적인 방법으로 신뢰성 있는 성능측정 결과를 도출할 수 있는 성능측정 방법과 컴포넌트 성능측정기의 필요성은 높다.

본 논문에서 제시한 분산된 컴포넌트의 성능을 측정하는 방법과 성능측정 요소, 그리고 개발 방법을 이용하면 분산 컴포넌트 플랫폼이나 컴포넌트 모델에 종속됨 없이 컴포넌트의 성능을 측정하고 객관적인 결과 데이타를 도출하는 컴포넌트 성능측정기를 구현할 수 있다. 이 결과를 이용하면 동일한 컴포넌트 모델을 기반으로 하는 다수의 컴포넌트 뿐만 아니라 서로 다른 컴포넌트 모델을 기반으로 하는 동일한 명세의 컴포넌트 들에 대한 성능측정 수행이 가능하다. 정의된 개발방법을 통해 다수의 컴포넌트 모델이 존재하는 현재의 컴포넌트 시장에서 컴포넌트의 성능측정이라는 도메인의 표준 모델로서 다양하게 활용될 수 있을 것으로 기대된다.

참 고 문 헌

[1] Jerry Ga, Eugene Y. Zhu, Simon shim, "Testing component-based software," STARWEST '99, 1999.

[2] Jerry Ga, "Component testability and component testing challenges," CMU Software Engineering, Carnegie Mellon University, 2000.

[3] Jerry Ga, Eugene Y. Zhu, Simon shim," Monitoring Software components and component based software," San Jose State University, 1999.

[4] A. Mos and J. Murphy, "Performance Monitoring of Java Component-Oriented Distributed Applications" Proc. of 9th IEEE Conference on Software, Telecommunications and Computer Networks (SoftCOM), October 9-12, 2001.

[5] B.Sridharan, B. Dasarathy andA. P. Mathur, "On Building Non-intrusive Performance Instrumentation Blocks for CORBA-based Distributed Systems," 4th IEEE International Computer Performance and Dependability Symposium, Chicago March 2000.

[6] I. H. Kazi, et al. "JaViz : A client/server Java profiling tool," IBM System Journal VOL 39, NO 1, 2000.

[7] Ye Wu, Dai Pan and Mei-Hwa Chen, "Techniques for Testing Component-Based Software," Seventh International Conference on Engineering of Complex Computer Systems, Skövde, Sweeden, June 11-13, 2001.

[8] M. J. Harrold, D. Liang and S. Sinha, "AnApproach to Analyzing and Testing Component-Based Systems," Proc. of 1st Int ICSE Workshop on Testing Distributed Component-Based Systems, Los Angeles, CA, May 1999.
[9] 이긍해, "컴포넌트 성능측정 기법에 관한 연구보고서", 한국항공대학교, 2002.
[10] 황길승, 이긍해, 권오천, 신규상, "Black Box 방식의 EJB컴포넌트 성능측정", 한국정보과학회, 봄학술발표논문집(B) pp. 382-384, 2002.
[11] Richard Soley and OMG Staff Strategy Group, "Model Driven Architecture," Draft3.2, http://doc.omg.org/omg/2000-11-05, 2000.
[12] Richard Soley, "Model Driven Architecture : An Introduction," http://www.omg.org/mda/presentations.htm
[13] OMG Architecture Board MDA Drafting Team, "Model Driven Architecture: A Technical Perspective," http://cgi.omg.org/docs/ormsc/01-07-01.pdf
[14] Adrian Mos, John Murphy, "Performance Management in Component-Oriented Systems Using a Model Driven Architecture™ Approach," Sixth International Enterprise Distributed Object Computing Conference (EDOC'02) September, 17-20, 2002.
[15] K.Zielinski, et al, "A tool for monitoring software-heterogeneous distributed object applications," 15th International Conference on Distributed Computing Systems (ICDCS'95) May 30 - June 02, 1995, Vancouver, Canada.
[16] Precise, Precise/Indepth for the J2EE platform, http://www.precise.com/products/indepth.asp
[17] Sitraka, PerformaSure, http://www.sitraka.com/software/performasure
[18] Empirix, BeanTest, http://www.empirix.com/empirix/web+test+monitoring/products/
[19] Segue, SilkTest, http://www.segue.com/html/s_solutions/s_silktest/s_silktest_toc.htm
[20] Rahim Adatia, et al. Professional EJB, Wrox Press, 2001.
[21] Robert Orfali, Dan Harkey, Client/Server Programming with Java and CORBA, WILEY Press, 1997.
[22] Ivar Jacobson, Grady Booch, James Rumbaugh, The Unified Software Development Process, Addison-Wesley, 2001.
[23] 김현남, "UML과 래셔날 로즈를 이용한 비즈니스 모델링", http://www.javastudy.co.kr
[24] Stephen J. Mellor, Marc J. Balcer, Executable UML - A foundation for Model Driven Architecture, Addison-Wesley, 2002.
[25] Java Virtual Machine Profiler Interface (JVMPI), http://www.javasoft.com/products/jdk/1.2/docs/guide/jvmpi/jvmpi.html.
[26] Aniruddha Gokhale, Douglas C. Schmidt, Balachandran Natarajan, Nanbor Wang, "Applying Model-Integrated Computing to Component Middleware and Enterprise Applications," COMMUNICATION OF ACM Vol. 45, No. 10, October 2002.
[27] Matjaz B. Juric, Professional J2EE EAI, Wrox Press, 2002.

황 길 승

2002년 한국항공대학교 컴퓨터공학과 학사. 2004년 한국항공대학교 컴퓨터공학과 석사. 2004년~현재 한국전자통신연구원(ETRI) 기반기술연구소 임베디드S/W기술센터 S/W공학연구팀. 관심분야는 Product Line Software Engineering, 소프트웨어 테스팅

이 긍 해

1980년 서강대학교 전자공학과 학사. 1986년 Viginia Tech 전산학과 석사. 1990년 Viginia Tech 전산학과 박사. 1990년~1992년 IBM 연구원. 1999년~2000년 IBM Almaden Research Center 방문과학자. 1993년~현재 한국항공대학교 컴퓨터공학과 교수. 관심분야는 소프트웨어공학, 객체지향 언어 및 시스템, 인터넷 컴퓨팅