# 아키텍쳐 유효성 검토를 위한 자바빈즈 컴포넌트 기반의 시뮬레이션 도구

황영석, 정재호, 이강선
명지대학교 컴퓨터학과
email: yshwang@mju.ac.kr, jjho@mju.ac.kr, ksl@mju.ac.kr

# JavaBeans-based Simulation Environment for System Architecture

Youngseuk Hwang, Jaeho Jung, Kangsun Lee
Department of Computer Science and Engineering
MyongJi University

요 약

본 논문에서는 UML의 배치 다이어그램이 가지는 런타임 아키텍쳐 정보를 통해 시뮬레이션 모델을 구성하고, 이 모델을 실제 시뮬레이션하여 구현 단계 이전에 시스템 아키텍쳐의 유효성과 성능 정보를 검토하는 CoSim(Hardware Software Co-Simulator System)을 제시한다. CoSim은 자바빈즈 컴포넌트 기반으로, 크게 Modeler, Translator, Scenario로 구성된다. 시스템 개발자는 Modeler를 이용하여 시뮬레이션 모델을 작성하며, Translator는 모델에 대한 시뮬레이션 자바 코드를 생성하고, 그 결과물을 바탕으로 Scenario는 비주얼한 정보를 제공한다. 따라서 모델이 실제 플랫폼 상에서 작동되기 이전에 아키텍쳐 성능에 관련된 유용한 정보를 제공하여 개발 위험도를 감소시키고 비용의 절감을 가져 올 수 있다. CoSim은 Modeler, Translator, Scenario 별로 자바빈즈 컴포넌트 라이브러리를 제공함으로써 모델링의 재사용성과 확장성 및 생산성을 높여 줄 수 있다.

## 1. 서론

웹기반 어플리케이션의 증대 및 사용 인구의 증가는 내용(contents)면에서의 경쟁력 뿐만이 아니라 성능(performance)면에의 경쟁력을 요구하고 있다.[1] 예를 들어 긴 downloading 시간 등 성능면에서의 문제점은 컨텐츠의 유용함에도 불구하고 경쟁력을 떨어뜨리는 주요 원인이다. 웹기반 어플리케이션의 분석 및 설계 시 제기될 수 있는 대표적인 성능 관련 질문은 다음과 같다. [1-2]

- 서버의 수가 고객 집중 시간대의 요구를 감당해 낼 수 있는가?
- 설계된 시스템의 아키텍쳐가 확장성(scalibility)이 있는가?
- 시스템의 성능향상이 요구될 때 시스템 아키텍쳐의 어떤 부분을 향상시켜야 하는가? 예를들어 데이터 베이스 서버, 웹서버, 어플리케이션 서버, 네트워크 bandwidth 중 어떤 것을 향상시키는 것이 가장 효율적인가?

이러한 질문은 거의 모든 시스템의 구축 시 빈번하게 제기되고 있지만, 일반적인 웹 기반 어플리케이션 혹은 분산 어플리케이션의 시스템 성능 관련 정보를 분석 및 설계 단계에서 미리 예측할 수 있는 방법 및 지원도구는 많이 알려져 있지 않다. 이는 아키텍쳐상의 표현되어 있는 네트워크 지연

이 비결정적(nondeterministic)이어서 수치적인 분석이 불가능하며, 또한 성능 분석을 위한 테스팅 과정 및 측정 과정이 많은 비용을 소요하기 때문이다. 결국 성능면에서의 만족 여부는 소프트웨어 생성 주기의 후반부 - 테스트 과정 혹은 실제 배치(deployment) 후) - 에나 이루어질 수 있으므로 최악의 경우 재설계(re-design) 및 재구현(re-implementation) 과정을 초래할 수 있다.[1,3]

본 논문에서는 시스템 분석 및 설계시에 시뮬레이션을 통하여 (design-time simulation) 시스템의 성능 관련 정보를 미리 분석하고 예측할 수 있는 CoSim (Hardware & Software Co Simulator System)을 소개한다. CoSim은 객체지향 모델링 언어인 UML의 배치 다이어그램을 이용하여 시스템의 아키텍처를 묘사한다. UML의 배치다이어그램은 노드와 링크를 통해 시스템을 구성하는 소프트웨어 컴포넌트가 무엇인지 또한 어떠한 하드웨어적 요소로 구성되어 있는지를 묘사한다. [4-5] CoSim은 UML 배치다이어그램의 노드와 링크 부분을 확장하여 성능 예측에 필요한 파라미터 • 시뮬레이션 실행 정보 • 시뮬레이션 목적함수(서버의 부하정도, 서버의 트랜잭션 처리시간, 활용정도 등) 등을 추가한다. 묘사된 시스템 아키텍처 관련 정보는 CoSim 내부에서 시뮬레이션 모델로의 성립 여부를 조사한 후, 올바른 모델의 경우 Java 형태의 시뮬레이션 코드를 자동 생성하게 된다.

본 논문의 구성은 다음과 같다. 2장에서는 CoSim의 주요 구성 요소를 살펴본 후, 3장에서 간단한 예제를 통하여 실행과정을 소개한다. 4장에서는 향후 연구 방향을 토론하고 결론을 맺는다.

## 2. CoSim의 구조

CoSim은 <그림 1>과 같이 크게 Modeler, Translator, Engine, Scenario로 구성된다. 2.1~2.3은 각 구성요소를 설명한다.

### 2.1 Modeler

Modeler는 시스템을 모델링하고 시뮬레이션하는데 필요한 GUI(Graphic User Interface)를 제공한다. Modeler를 이용하여 사용자가 시뮬레이션을 수행하는 과정은 1) 시스템 모델 정의, 2) 시뮬레이션 모델 작성, 3) 시뮬레이션 실행, 4) 결과 리포트의 단계로 진행된다.

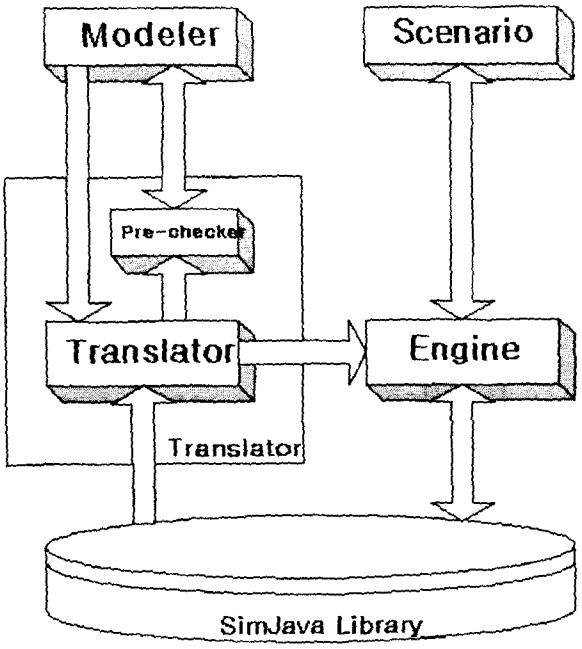


그림 1 CoSim의 구조

사용자는 UML의 배치 다이어그램을 통해 시스템의 아키텍처를 묘사한다. 배치 다이어그램은 시스템에서 소프트웨어와 하드웨어 사이의 물리적인 관계를 보여주는 다이어그램으로, 노드 • 링크 • 컴포넌트 • 의존관계로 구성된다.[7]

<그림2>는 UML의 Deployment diagram 예를 보여준다.

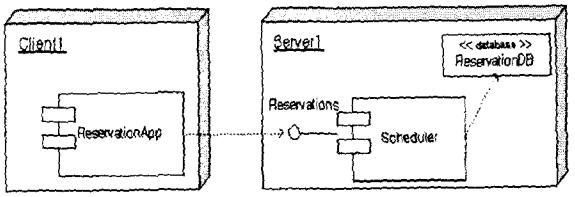


그림 2 UML의 Deployment diagram의 예

<그림 2>에서는, Client1 시스템에 포함된 컴포넌트 ReservationApp를 통해 서버에 요청하면 서버를 구성한 Scheduler 컴포넌트에 전달되어 DB에 요청하는 시스템을 묘사한 것이다.[7]

이렇게 묘사된 시스템 아키텍처를 실제 성능 평가기 위해서는 다음과 같은 정보가 필요하다.

- 노드 - 해당 노드 위에서 작동되는 소프트웨어 컴포넌트에 대한 실행 정보(복잡도, 수행예상 시간), 노드 자체의 하드웨어적 정보(processor type 등)
- 링크 - Network type, bandwidth 등의 하드웨어 정보
- 시뮬레이션 목적과 시간 - 사용자가 각 노드에서 얻고자하는 정보와 신뢰성 있는 계산을 위한 반복 회수 등
- 시뮬레이션 정보 - 노드간 데이터의 동적 흐름 순서

본 논문에서 제시하는 모델러는 위의 정보들을 자바빈즈 컴포넌트로 구현한다. 자바빈즈 컴포넌트는 자바의 컴포넌트 모델이다.[8] 자바빈은 다른 컴포넌트를 부분 수정하거나 조합해서 또 다른 컴포넌트를 만들 수 있는 재사용성 소프트웨어 컴포넌트이다. 자바빈즈 기반의 컴포넌트는 컴포넌트를 분석, 조작하는데 있어서 공통적으로 적용되는 표준을 정의할 수 있는 장점이 있는데, 이렇게 만들어진 컴포넌트는 표준 자바빈즈를 제공하는 어떤 개발 툴에서도 사용할 수가 있다. 본 논문의 모델러는 자바빈즈 기반의 컴포넌트의 장점을 이용해서 노드와 링크의 시각적인 배치와 재사용성을 높인다.

### 2.2 Translator

Translator는 <그림 1>에서와 같이 Pre-checker, Translator로 구성된다.

#### 2.2.1 Pre-checker

Pre-checker는 Translator로부터 시뮬레이션에 필요한 정보를 획득하여 사용자가 구성한 확장된 배치다이어그램이 시뮬레이션 될 수 있는지 검사한다. 즉, 시뮬레이션에 필요한 노드와 링크의 delay 시간, 스케줄 정보와 오브젝티브를 구할 때 필요한 파라미터가 노드 정보와 링크 정보에 있는지를 검사한 후 시뮬레이션에 필요한 모든 정보가 있으면 Modeler가 계속해서 진행하도록 하고, 필요한 정보가 없으면 해당 정보가 없음을 Modeler에게 알린다.

#### 2.2.2 Translator

Translator는 사용자가 구성한 Deployment 다이어그램으로부터 Modeler가 추출한 시뮬레이션 정보를 가지고 자바 기반의 시뮬레이션 코드를 자동 생성한다. Translator는 크게 모델 정보 분석, 시뮬레이션 알고리즘 작성, 오브젝티브 알고리즘 작성의 3 단계로 진행된다.

- 모델 정보 분석 - Modeler로부터 사용자가 작성한 모델 정보(in.smi 파일)를 읽어들여 이를 분석하고 분류하여 저장한다. 즉, 각 노드를 클래스로 만들기 위해 필요한 클래스 정보, 시뮬레이션의 스케줄 정보, 오브젝티브 정보를 읽어들인다. 클래스 정보는 노드의 이름이 되는 클래스 이름, 다른 노드와의 연결을 위한 포트 구성 • 링크 구조이고, 스케줄 정보는 스케줄되는 노드의 순서 • 스케줄 횟수 • 각 노드와 링크의 delay 시간이다. 오브젝티브 정보는 각 노드의 오브젝티브 이름과 오브젝티브를 구하기 위한 파라미터로 구성된다.
- 시뮬레이션 알고리즘 작성 - 분석된 모델 정보와 SimJava 라이브러리를 이용해서 시뮬레이션 구조를 구성한다.[6] 즉, 클래스 정보로 각 노드를 클래스로 만들어 각각을 쓰레드(thread)로 구성하고, 스케줄 정보를 이용해서 스케줄링 구조를 구성하며 스케줄 시간을 계산하는 알고리즘을 구성한다.
- 오브젝티브 알고리즘 작성 - 사용자가 알고자 하는 오브젝티브를 시뮬레이션 하는 동안 계산한다. 주로 사용되는 오브젝티브에는 Transmission time, Utilization, 부하량이 있다. 오브젝티브 계산에 대한 자세한 내용은 참고문헌 [1]에서 찾아볼 수 있다.

SimJava는 분산 이벤트 시뮬레이션 자바 패키지로 각 개체를 쓰레드로 구성해서 이벤트를 주고받도록 되어 있기 때문에 이를 이용할 때, 각 노드에 독립성과 동시성을 제공하여 보다 현실에 가까운 시뮬레이션을 할 수 있게 된다. [6]

### 2.3 Engine

Engine은 Translator가 생성한 자바 코드와 런타

입 라이브러리를 이용해서 실제로 시뮬레이션하고, 시뮬레이션 결과를 산출한다. 즉, Engine은 *.java 파일을 입력받아 SimJava 런타임 라이브러리를 이용해서 실행하고 그 결과로 Report와 Tracefile을 생성한다. Report 파일에는 사용자가 요구한 오브젝티브에 대한 결과와 시뮬레이션이 이루어진 시간 정보가 기록된다. Trancefile에는 시뮬레이션하는 동안 발생하는 모든 이벤트가 시뮬레이션 시간에 따라 기록된다.

### 2.4 Scenario

Scenario는 Translator를 통한 시뮬레이션 결과를 가시적으로 보여준다. 현재 Scenario는 Report 파일과 Tracefile 파일을 보여주는 형식이나, 향후 애니메이션을 통하여 결과를 직관적으로 이해할 수 있도록 할 계획이다.

## 3. 예제

본장에서는 CoSim의 진행 과정을 예를 통해 설명한다. 예에서 사용하는 시스템은 기본적인 3-tier 시스템으로, 사용자가 자신의 계좌에 있는 잔액 조회를 요청하면 해당 은행의 서버에 연결된 후, 서버의 DB에서 실제 트랜젝션을 처리하는 시스템이다.

### 3.1 시스템 모델 정의

<그림 3> 은 CoSim을 이용하여 예의 은행 시스템을 모델링 한 것이다.

작성된 모델을 시뮬레이션 하기 위해선 노드•링크•시뮬레이션 목적과 시간•스케줄 정보가 필요하다.

각 노드에 필요한 정보로는 delay time, frame size 등에 관련된 General Specification과 각 노드를 구성하는 component Specification, hardware Specification이다. Node Specification 다이얼로그는 이 과정에서 쓰여지는 모델러 부분이다. delay time과 frame size는 General 탭에서 명시되며 평균과 오차값을 갖는다. Component와 hardware는 각각 Component 탭과 Hardware 탭에서 리스트 방식으로 추가, 삭제, 수정된다.

링크는 자신이 연결하고 있는 두 노드의 정보 및 delay time과 hardware 정보를 갖는다. 노드와 마

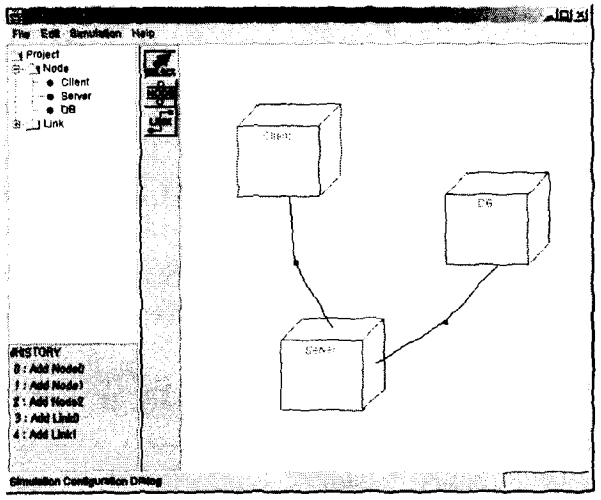

그림 3 CoSim을 이용한 은행 시스템 모델링

찬가지로 Link Specification 다이얼로그로 정의되고, delay time을 평균과 오차 범위로 명시한다. <그림 4>는 Node Specification 과 Link Specification 다이얼로그를 보여준다.

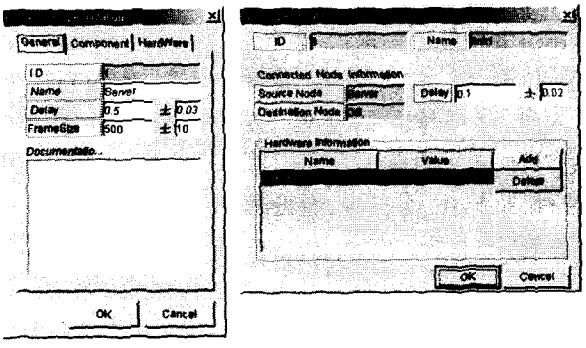

그림 4 Node(왼쪽), Link(오른쪽)의 Specification

주어진 은행 시스템의 시뮬레이션 목적은 각 노드별로 명시되어 질 수 있다. 주어진 예의 시뮬레이션 목적은 Client의 경우 transmission time, Server는 utilization, DB는 transmission time으로 결정된다.

<그림 5>는 CoSimimulation Configuration 다이얼로그의 Objective 탭을 보여준다.

또한, 작성된 모델의 동적인 수행정보를 위해 Schedule을 명시해야 한다. Schedule 탭은 링크

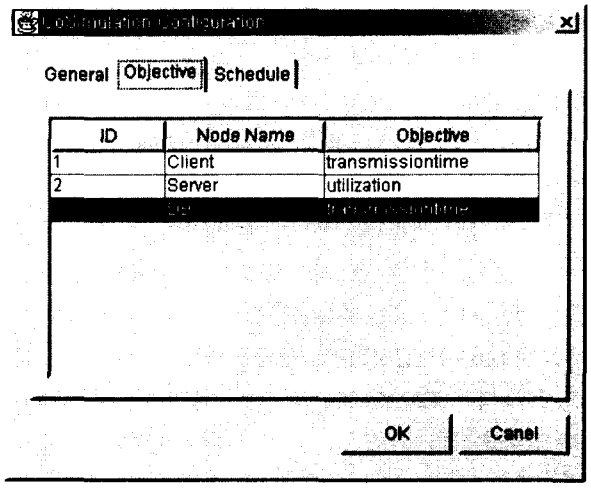


그림 5 Objective 구성

되어진 모든 노드의 리스트를 보여주고 사용자는 데이터의 흐름에 따라 노드의 순서를 결정한다. 주어진 은행 예의 경우 Client, Server, DB 순서로 진행된다.

<그림 6>은 Schedule 탭을 보여준다.

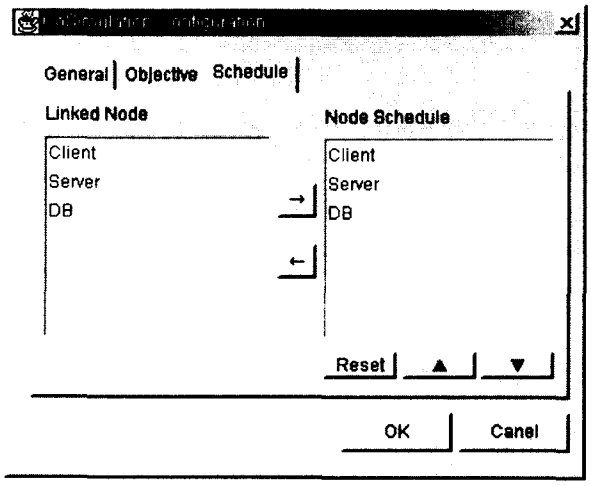


그림 6 Schedule 구성

### 3.2 Translate

3.1의 모델링 정보는 클래스 정보, 스케줄 정보, 오브젝티브 정보가 정의되어 있는 in.smi 파일의 형태로 Translator에 전달된다.

Pre-checker는 Translator가 자바코드를 생성하기 전에 모델 정보에서 시뮬레이션 스케줄에 따른 Client, Server, DB 노드와 Ethernet, TokenRing 링크의 delay 시간과 Server의 utilization, DB의 transmissiontime 오브젝티브에 따른 Server의 framesize, 링크의 bandwidth 파라미터가 노드와 링크 정보에 있는지를 검사한다. 모든 시뮬레이션 정보가 있으면 Translator는 SimJava 형태의 시뮬레이션 코드를 생성한다. <그림 7>은 생성된 자바 코드의 일부를 보여준다.

```
class Client extends Sim_entity {
  private Sim_port port0;

  public Client (String name) {
    super(name);
    port0 = new Sim_port("port0"); add_port(port0);
  }

  public void body() {
    Sim_event ev = new Sim_event();
    Printout Report = new Printout();
    Double totalSim;
    Sim_uniform_obj framesize, bandwidth;
    double transmissiontime, transmissiontimeMean=0.0;
    Sim_uniform_obj processDelay, transmissionDelay1;
    double processSim, transmissionSim1;

    for(int i=0; i<10; i++) {
      processDelay = new Sim_uniform_obj("processDelay",1.0,1.4,i*123);
      processSim = processDelay.sample();
      transmissionDelay1 = new Sim_uniform_obj("transmissionDelay1",
                           0.18, 0.22, i*123);
      transmissionSim1 = transmissionDelay1.sample();
      sim_hold(100);
      Report.report("Client -> " + port0.get_dest_ename()
                  + " Simulation-time: " + transmissionSim1);
      totalSim = new Double(transmissionSim1);
      sim_schedule(port0, transmissionSim1, 0, totalSim);

      framesize = new Sim_uniform_obj ("framesize", 425.0, 575.0, i*100);
      bandwidth = new Sim_uniform_obj ("bandwidth", 7, 13, i*100);
      transmissiontime = (framesize.sample()*8) /
                         (bandwidth.sample()*1000000);
      transmissiontimeMean += transmissiontime;

      Report.report("Client's Transmission-time: " + transmissiontime
                  + " sec");
    }

    sim_hold(100);
    transmissiontimeMean /= 10;
    Report.report("Client's Average Transmission-time: "
                + transmissiontimeMean + " sec");
  }
}
```

그림 7 생성된 자바 코드 일부

### 3.3 결과

생성된 자바 파일은 SimJava 라이브러리와 함께 Engine에서 실행되어 실행결과를 보여준다. <그림

8>는 이를 통해 생성된 Report 파일의 일부이다.

```
Client's Average Transmission-time: 3.761340509244737E-4 sec
Server's Average Utilization: 71.09472137704465 %
DB's Average Transmission-time: 2.460942126536497E-4 sec
Average Total Simulation-time: 0.10843212103282995
```

그림 8 Report 파일의 일부

주어진 은행 시스템에 대한 시뮬레이션 결과는 다음과 같다.

- Client의 평균 transmission time은
  ≒ 37.61 ms
- Server의 평균 utilization은
  ≒ 71.09 %
- DB의 평균 transmission time은
  ≒ 24.61 ms

이를 통해 Modeler는 작성한 시스템 아키텍처의 성능정보를 미리 예측해 볼 수 있다.

## 4. 결론 및 향후 연구방향

본 논문에서는 자바빈즈 컴포넌트 기반의 시뮬레이션 도구인 CoSim을 제작하였다. CoSim은 소프트웨어 및 시스템 아키텍쳐의 성능 정보를 설계 단계에서 예측하기 위해 시뮬레이션을 수행한다. CoSim은 Modeler, Tranlator, Engine, Scenario 별로 컴포넌트를 구성하여, 정의된 컨텍스트와 서로 약속된 인터페이스를 통해 작동된다. 따라서 향후 일반적인 성능 평가 도구 제작시 ASP(Application Service Provider)나 서드파티(Third Party)들의 필요에 따라 바이너리 형태의 조합 및 재사용 가능성을 높인다.

CoSim의 재사용성을 높이기 위한 방안으로 다음과 같은 것을 고려할 수 있다.

- Domain Wizard: Domain-Specific Software Architecture(DSSA) 분야는 특정 도메인 별로 공통적인 개념과 원리를 알아내고(도메인 모델), 참고 요구사항(Reference Requirements), 및 참고 아키텍쳐(Reference Architecture)를 정의하여 특정 도메인 별 재사용 가능성을 극대화한다[9]. CoSim에서는 향 후 특정 도메인을 선정하여 도메인 아키텍쳐를 작업을 수행한 후, CoSim에서의 아키텍쳐 기술 기본 단위인 노드와 링크별로 컴포넌트를 제작할 계획이다. 이를 통해 모델 작성자는 자신이 속한 도메인에서 이미 작성된 노드와 링크 컴포넌트를 재사용하여 작업을 진행 할 수 있으므로 개발 속도의 증가 및 시간 절감을 가져올 수 있다.
- Web-based CoSim: CoSim의 기능은 크게 Server와 Client별로 나누어 재정의 할 수 있다. 예를 들어, Client는 자신의 아키텍쳐 모델을 작성한 후 Server로부터 자신이 작성한 모델에 대한 시뮬레이션 결과를 요청할 수 있다. 또한, Client가 작성한 모델을 웹상에서 다른 사람들이 사용할 수 있도록 하기 위해 Server의 Database에 저장 요청 할 수 있으며, 동시에 다른 사람들이 작성한 아키텍쳐 모델을 Server로부터 조회 한 후 자신의 Modeler에서 사용할 수 있다. Client와 Server 기능의 분리는 각 기능별로 재사용 기회를 증진시킬 수 있으며, Client쪽의 어플리케이션을 가볍게 가져갈 수 있다는 장점이 있다. 현재 자바빈즈 형태로 작성된 CoSim은 EJB(Enterprise Java Beans)[10][11] 형태로 쉽게 확장될 수 있어, Web-based CoSim을 위한 분산 서버 제작을 용이하게 할 수 있다.

## 참고문헌

[1] Daniel A Menasce and Virgilio A. F. Almeda, Scaling for E-Business, Prentice Hall, 2000

[2] GVU, "GVU's WWW User Surveys", http://www.gvu.gatech.edu

[3] Svend Frolind and Pankaj Garg, Design-Time Simulation of a Large-Scale, Distributed Object System, ACM Transactions on Modeling and Computer Simulation, vol 8., no 4. October 1998, pp 374-400

[4] James Runbaugh, Ivar Jacobson & Grady Booch, "The Unified Modeling Language Reference Manual", Addison-Wesley, 1999

[5] Jim Conallen, "Building Web Applications with UML", Addison-Wesley, 1999

[6] SimJava Library, Writing simulation using th e library, http://www.dcs.ed.ac.uk/home/hase/simj ava

[7] Craig Larman, "Applying UML and Patterns: An Introducation to Object-Oriented Analysis and Design"

[8] Sun Microsystems: JavaBeans 1.01 API Speci fication, 2001, http://java.sun.com/products/javabe ans/glasgow

[9] R. Taylor, W. Tracz, and L. Coglianese, "Sof tware Development Using Domain-Specific Soft ware Architectures: CDRL A011A curriculum Mo dule in the SEI style", SIGSOFT Software Engi neering Notes, vol. 20, pp 27-37, December, 1995

[10] Enterprise JavaBeans™ technology, http://ja va.sun.com/products/ejb

[11] Vlada Matena and Beth Stearns, "APPLYIN G ENTERPRISE JAVABEANS: Component-Base d Development For The J2EE Platform", Addiso n-Wesley Pub Co (Sd), December, 2000