OSS/J 의 분석 및 이를 이용한 OSS 파일럿 시스템 구축을 통한 OSS/J 의 효과 분석

안영회 [1], 이준걸 [1], 황성훈 [2], 김인규 [3]
1 국민대학교 BIT 전문대학원 비즈니스 컴퓨팅 전공
2 한국썬마이크로시스템즈, 국민대학교 BIT 전문대학원 겸임교수
3 국민대학교 BIT 전문대학원 교수

A Study on OSS/J and the Benefit of OSS/J-based Implementation with Pilot

Young-Hoe Ahn, Jun-Geol Lee[1], Seonghoon, Whang[2], In-Kyu Kim[3]
1. Dept. of Business Computing, BIT Graduate School of Kookmin University
2. Sun Microsystems, Inc., Prof. of BIT Graduate School of Kookmin University
3. Prof. of BIT Graduate School of Kookmin University

요 약

본 논문은 OSS/J 의 소개, 분석 및 OSS/J 호환 제품 및 OSS/J RI(Reference Implementation)을 사용하여 파일럿 시스템을 구축하고 이의 효과를 분석하여 OSS/J 가 관련 업계에 미치는 영향에 대한 조망을 제시한다.

1. OSS/J 소개

1.1 배경

현재의 OSS 기술은 빠르게 변화하는 네트워크의 규모, 다양한 통신 기술, 새로운 서비스의 더욱 짧아진 시장 진입 시간, 가용성과 신뢰성에 대한 한층 높아진 기대치 등을 감당할 수 없는 지경에 이르렀다. 이렇게 증가되어가는 요구에 부합하기위해, 서비스 제공자는 새로운 OSS 솔루션을 제공하기위한 접근법을 제공하고 있다. OSS/J 는 OSS 컴포넌트 시장을 육성하기위해 클라이언트가 밀접한(tightly) 혹은 느슨한(loosely) 연결 메커니즘으로 접근할 수 있도록 하는 API 를 정의한다. OSS/J 의 목적은 OSS 시스템 구축에 필필요한 API 스펙, RI(Reference Implementation), 그리고 TCK(Technology Compatibility Kits)을 개발하는 것인데, 이러한 과정은 모두 JCP(Java Community Process) 프로그램을 통해 이루어지게 된다.

OSS/J 는 Sun Microsystems, BEA, Ericsson, Motorola, Nokia, Nortel Networks, NEC, Telcordia Technologies 와 함께 JCP 에 JSR-144 OSS Common API 라는 이름으로 등록하면서부터 시작 되었다. 이들 회사는 OSS/J Common API 스펙을 만들고 RI 를 개발하는데 적극적으로 참여해 왔으며, 현재 Common API 이외에 5 개의 API 와 4 개의 RI 가 제공되어지고 있다.

1.2 장점 및 효과

1.2.1 OSS/J 표준으로써의 효과

현 비즈니스 환경에서 다양한 서비스 요구를 충족하는 솔루션을 적기에 제공하기 위해 OSS/J 라는 표준 API 가 만들어진 것처럼 이러한 OSS 시스템을 개발, 제공, 운영하는 데에는 많은 역할이 서로 다이나믹 하게 연관되어져 있으며, 이렇게 서로 다른 그룹들간의 관계 및 해당 그룹의 효과는 다음과 같다.

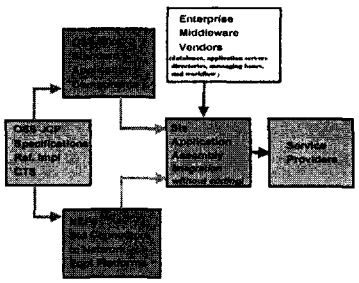

그림 1. OSS/J Value Chain

• Enterprise Middleware Vendors : OSS/J API 를 채택함으로써 시장을 확장 할 수 있다. 또한 벤더는 시스템 프로그래밍에 대한 노력을 기울이는 대신 업계가 요구하는 수준(carrier grade container)의 사용 및 특화 비즈니스 부분을 통해 확장성, 성능 그리고 가용성 있는 상품을 만들 수 있다.

• Network Equipment providers(NEPs) : EMS (Element Management System)와 NMS(Network Management System)에 대한 API 를 구현하였을 때, 그들은 자신들의 네트워크 요소에 대해 독립적인 소프트웨어 벤더(ISV)를 끌어 들일 수 있고, OSS 어플리케이션을 위한 폭 넓은 선택을 할 수 있게 된다. 또한 NEPs 네트워크 요소의 관리용이성을 향상시킬 수 있다. "어플리케이션 조립"으로써의 EJB 모델은 표준 OSS 어플리케이션은 ISV 가 표준 OSS 컨테이너를 제공함으로써 OSS 솔루션에 필요한 통합을 혁신적으로 감소시킬 수 있다. NEPs 는 자유롭게 RI(Reference Implementation)을 이용함으로써 개발과 통합의 노력을 상당부분 감소시킬 수 있다. 이렇게 절약된 노력은 핵심 비즈니스 로직에 대해 집중하고 부가가치 있는 기능을 개발할 수 있게 한다. 바로 이점이 경쟁업체와의 차별화를 이룰 수 있는 기반이 되어질 것이다.

• Independent Software Vendors(ISVs) : ISV 는 이미 OSS/J 의 산출물을 이용하여 더욱 빠른 개발을 하므로, 더 빠르게 시장의 요구에 부합할 수 있다. OSS/J 는 ISV 의 솔루션에 통합과 상호 운영성을 보장한다. 또한 OSS/J API 를 지원하므로 많은 수의 네트워크 요소 및 이들 네트워크 요소의 관리 시스템에 대해 접근할 수 있게 된다. 그리고 개별 제품에 대한 고유한 어댑터를 거의 필요로 하지 않기 상호 운영성이 더욱 증가하게 되는 효과를 이루게 된다.

• System Integrators(SIs) : OSS/J 라는 표준 인터페이스를 사용하기 때문에 어플리케이션 통합에 따른 비용이 감소하며, 비즈니스 이익에 더욱 집중할 수 있게 된다.

• Service Provider(SPs) : 쉽게 상호 운영될 수 있는 OSS 어플리케이션을 위한 시장은 서비스 제공자로 하여금 그들의 요구사항에 가장 잘 부합하는 어플리케이션을 선택할 수 있게 한다는 것이다. 또한, 시스템을 운영하는 사람의 입장에서는 기존의 검증된 컨테이너와 컴포넌트의 사용을 통해 상당 수준의 QOS 를 보장 받을 수 있다는 장점을 더불어 가지게 된다.

1.2.2 J2EE 플랫폼으로써의 장점

• 손쉬운 통합 모델: J2EE 플랫폼과 EJB 아키텍쳐는 엔터프라이즈 어플리케이션 통합을 지원한다. EJB 컨테이너 기술은 기존 데이터 처리와 데이터소스에 대한 지원을 통해 엔터프라이즈 어플리케이션의 통합을 쉽게 할 수 있는데 이는 J2EE 커넥터 기술의 제공을 통해 지원된다.

• 분산 객체 사용: EJB 컨테이너는 IIOP 를 통해 어떠한 시스템도 접근 할 수 있는 분산객체 접근을 제공한다.

• 최소한의 시스템 프로그래밍: J2EE 와 EJB 아키텍쳐는 시스템 수준에서 필수적으로 요구되는 것을 기본적으로 제공하므로 시스템 프로그램을 최소화 한다. 가령, 트랜잭션 관리, 라이프사이클 관리, 리소스 풀링등 이다.

• 통신사의 요구를 충족하는 수준 높은 지원: 통신 네트워크 관리는 높은 가용성과 신뢰성을 요구한다. EJB 기반의 네트워크 관리 프레임워크는 높은 가용성과 신뢰성을 통신-클래스 EJB 컨테이너를 통해 획득할 수 있다.

1.3 발전 방향

OSS/J 의 핵심 API 는 TeleManagement Forum 의 eTOM 프로세스에 매핑하는 것을 기본으로 하고 있으며 OSS/J 는 이를 근간으로 핵심적인 접근을 하고 있다. 이런 식의 매핑을 통해 현재 적용된 핵심 API 와 앞으로 개발 되어질 API 의 OSS/J API 로드맵은 아래 그림과 같다.

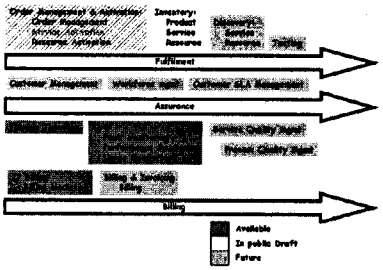

그림 2. OSS/J Core API Landscape

2. OSS/J 의 현재 구성 현황

2.1 전체 구성

OSS/J 는 OSS 시스템이 요구하는 기능들을 포괄하기 위해 340 여 회원사를 가지고 있는 국제적인 통신업계 표준화 단체인 TM Forum[1]에서 제시하는 OSS/BSS 를 위한 프레임워크인 NGOSS[2]에 참조하고 있다. NGOSS 는 비즈니스 프로세스의 자동화를 위한 eTOM(enhanced Telecom Operations Map)이라는 기능 모델을 제시하고 있는데 OSS/J 의 API 는 eTOM 의 영역을 포괄하는 방향으로 진행되고 있다. 그림 3 이 보여주는 eTOM 에 대한 OSS/J 매핑이 시사하는 바는 OSS/J 는 실현 가능한 최선의 방안을 제시한다는 점이다.

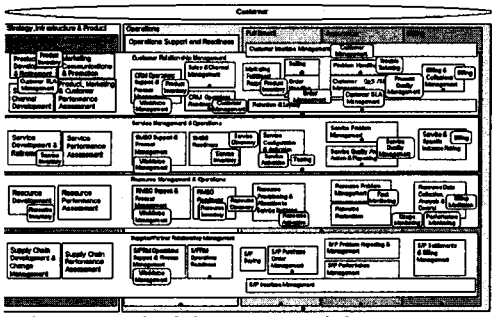

그림 3. eTOM 에 대한 OSS/J API 매핑

OSS/J API 는 JCP 에 의해 정의되며 각각의 API 는 표준 사양(Specification), 참조 구현(Reference Implementation) 및 TCK(Technology Compatibility Kit)의 세 가지 형태의 산출물로 구성되어 있다.

2.1.1 OSS/J 의 특성

OSS/J API 개발의 주도하고 있는 OSS/J Initiative 에서 제시하는 설계 가이드[3]에 따르면 OSS/J 설계의 근간이 되는 주요한 특성들은 다음과 같은 여섯 개 항목으로 분류할 수 있다. OSS 시스템의 기능들은 EJB 의 형태로 제공된다. 비즈니스적인 의미를 갖는 형태의 큰 인터페이스로 제공된다. 클러스터링, 확장성 및 장애 대책 등은 어플리케이션 서버의 기능을 이용하여 구축한다. 또한 메시징 방식을 이용하여 컴포넌트 사이의 의존성을 최소화하며, 프로세스의 통합과 컴포넌트의 협업에 의존한다. 기존 시스템의 통합은 JCA 를 활용한다.

2.1.2 OSS/J 빌딩 블록

OSS/J 빌딩 블록은 OSS/J 의 핵심 개념으로서, 하나 이상의 컴포넌트로, 하나 이상의 비즈니스 요구를 충족시키는 OSS 시스템을 구성한다. OSS/J 빌딩 블록을 구성하는 컴포넌트의 종류는 주로 JVT(Java Value Type)를 사용하는 EJB 세션빈 컴포넌트(Session Beans), EJB 엔터티빈 컴포넌트(Entity Beans) 및 MDB(Message Driven Beans) 등으로 구성되지만, 반드시 여기에 국한된 것은 아니다.

OSS/J 는 기본적으로 비즈니스 지향적인 인터페이스 구현을 위한 템플릿으로 Façade 패턴을 기본적으로 채용하여 세션빈 형태로 정의하고 있으며, OSS 시스템의 구성 요소들이 배치되는 서버에 국한되지 않고 사용하기 위하여 JNDI(Java Naming and Directory Interface) 표준 사양을 채택하고 있다.

2.1.3 OSS/J 의 상호작용 패턴

빌딩블록 사이의 상호작용 방식에 있어서는 OSS/J 는 동일한 기능을 제공하는 두 가지 서로 다른 상호작용 패턴을 제공하여 상황에 따른 선택적 구현이 가능하다. 그림 4 는 이를 도식화 한 것이다.

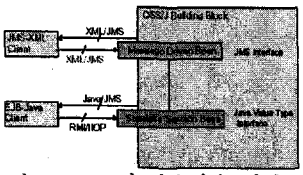

그림 4. OSS/J 의 상호작용 패턴

2.2 OSS/J API

2.2.1 TT(Trouble Ticket) API

TT API 는 Trouble Ticket 의 생성, 추적 및 삭제를 위한 인터페이스를 제공한다. TT API 는 서비스나 장비에 대한 장애에 대한 정보를 받아서 Trouble Ticket 을 생성하고, 문제가 해결될 때까지 Trouble Ticket 의 상태를 추적하며, 문제가 해결된 경우는 이를 고객에게 통보하고, Trouble Ticket 를 제거한다. 그림 5 는 TT API 와 관련 빌딩 블록 사이의 상호작용을 표현한 다이어그램이다.

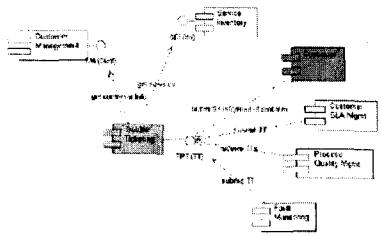

그림 5. TT API 와 관련 빌딩 블록 사이의 상호작용

2.2.2 QoS(Quality of Service) API

QoS API 는 크게 FM(Fault Management) API 와 PM(Performance Monitoring) API 로 이루어진다. FM API 를 사용하는 클라이언트는 장애에 대한 알람을 받아서 상태를 변경하고, 네트워크의 성능 수치에 대해 설정해놓은 임계치를 넘어서는 것에 대한 통지를 받으며, 현재 활성화 되어 있는 알람의 목록을 유지한다. 이는 궁극적으로 실시간에 가까운 네트워크 상태에 대한 지속적인 모니터링을 가능하게 한다.

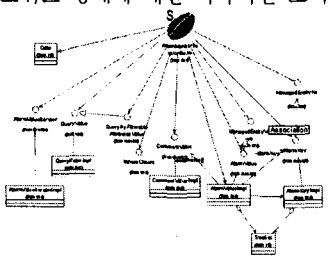

그림 6. QoS RI 의 클래스 및 인터페이스(비즈니스 관점)

2.2.3 SA(Service Activation) API

SA API 는 서비스 활성화를 지원하기 위한 API 이다. 그림 7 은 SA API 를 이용한 서비스를 정의에 관한 클래스 다이어그램이다.

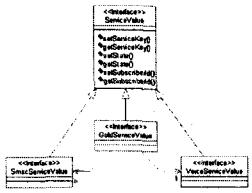

그림 7. SA API 에 정의된 서비스 인터페이스와 확장

2.2.4 Billing API

Billing API 는 서비스에 대한 과금과 청구내역 계산, 청구서 발송, 지불 처리 등을 가능하게 한다. 이외에도 고객의 청구서에 대한 문의를 처리하고, 요금 청구 과정에서의 문제를 처리하는 일을 지원한다. 그림 8 은 Billing API 와 관련 빌딩 블록의 상호작용을 보여주는 다이어그램이다.

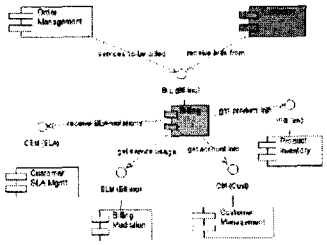

그림 8. Billing API 와 관련 빌딩 블록의 상호작용

3. OSS/J 를 활용한 OSS 시스템 구축

앞에서 기술한 OSS/J 의 효과 및 장점을 확인하기 위하여 OSS/J 를 지원하는 제반 제품과 OSS/J 의 RI 를 활용하여 간단한 OSS 시스템을 구축하였다. 본 논문에서 제시하는 OSS 시스템 구축 모델은 다음과 같다. OSS/J 의 TT, SA 를 사용하여 다음의 3 가지 Use Case 를 수행하는 OSS 시스템을 파일럿 수준에서 구축하고 이를 검증한다.

3.1 구축 시스템 아키텍쳐

파일럿 시스템의 아키텍쳐는 그림 9 와 같다.

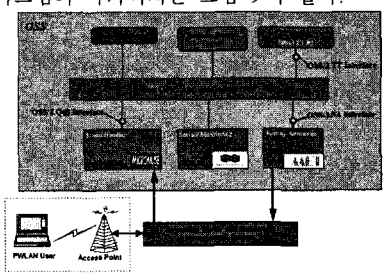

그림 9. OSS 파일럿 시스템 아키텍쳐

파일럿 시스템 구축에 사용한의 하드웨어 및 소프트웨어는 아래 표와 같다.

항목	세부사항
OSS 서버	Sun Ultra Server Solaris 8
Streaming Server 및 Bandwidth Management	Linux-2.4.17
사용자 시스템	Windows 2000
무선랜 Access Point	Linksys

항목	제품
Web Application Server, Message Bus	SunONE Application Server 7
Portal Server	SunONE Portal Server
Event handler	Micromuse TMS 5.2 RC 5
Service Monitoring	ILOG Jrules 4.0
Network namagement	SNMP++ V3.1
Video Streaming Client	QuickTime 6.0
Video Streaming Server	DarwinStreamingSrvr4.0-Linux
OSS/J TT RI	jsr091_oss_trouble_ticket-1_0-ri
Web Browser	Internet Explorer 또는 Netscape

표 3. 하드웨어, 소프트웨어 목록

3.2. Use Case

파일럿 시스템에서는 다음의 3 가지 use case(비즈니스 시나리오)를 구현하였다.

3.2.1 **Use Case 1: 무선 랜 서비스 사용자의 self-provisioning**

무선 랜 사용자가 서비스 사용 중에 현재 사용하고 있는 네트웍 속도(bandwidth)를 더 필요로 하는 경우, 서비스 사업자에게 전화 등의 방식으로 서비스 업그레이드 요청을 하고 이를 서비스 사업자가 처리하는 방식 대신에 필요한 서비스를 웹 환경에서 요청하면 이 부분을 OSS 시스템에서 가능한 범위에서 처리하고 이를 과금(billing) 부분까지 조정하는 기능을 보여준다. 해당 use case 의 동작 시나리오는 아래 그림 10 과 같다.

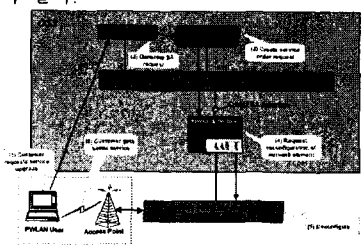

그림 10. Use case 1 동작 시나리오

(1) 사용자가 서비스 제공자의 포털에서 서비스 업그레이드를 요청한다.
(2) 포털에서 사용자가 요청한 부분에 대한 적합성을 점검하고 서비스 보증(Service Assurance)에 관한 요청을 생성한다.
(3) 관련 서비스 부분에 대한 요구를 생성한다
(4) 사용자의 요구에 해당하는 부분의 네트웍 환경에 대한 구성을 재 조정 요청을 한다.
(5) 해당 네트웍 장비의 환경 구성을 재 구성한다.
(6) 사용자는 업그레이드된 서비스를 사용한다.

3.2.2 **Use Case 2: 고객 관리 측면에서 사용자가 현재 사용하고 있는 서비스의 재 조정**

서비스 사용 중에 환경적인 요인으로 인하여 사용 서비스를 제대로 사용하지 못하는 경우, 사용자는 해당 서비스 사업자에게 불만을 표시하고 서비스 사업자는 사용자의 불만 사항을 확인하고 이에 대한 조치를 취한다. 해당 조치로 현재 사용자의 서비스 수준을 다음 단계로 업그레이드 함으로써(서비스 장애가 유지되는 동안만) 사용자는 보다 나은 서비스를 받을 수 있으며 이를 기반으로 서비스 사업자는 서비스 품질을 최적의 상태로 유지할 수 있다. 이의 동작 시나리오는 다음 그림 11 과 같다.

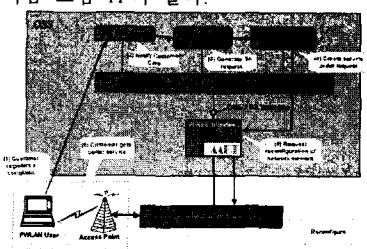

그림 12. Use case 2 동작 시나리오

(1) 사용자가 불만사항을 접수시킨다.
(2) 고객지원부에서 해당 불만 사항을 접수하고 검토한다
(3) 서비스 보증에 관한 요청(사용자의 서비스수준 업그레이드)을 생성한다.
(4) 사용자 서비스 수준을 다음 단계로 업그레이드 하라는 요청을 생성한다.
(5) 사용자의 서비스품질을 담당하고 있는 네트웍장비를 재 구성한다.
(6) 사용자는 더 나은 서비스를 받고 만족하며 서비스를 사용한다.

3.2.3 **Use case 3: 부분적인 네트웍 장애를 감지하고 이에 대한 조치로 무선 랜 사용자의 서비스를 재 구성하여 사용자의 서비스 품질을 보장**

일부 시스템 또는 네트웍 장비에 장애가 발생하여 해당 장애로 인하여 사용자가 계약된 서비스를 받지 못하는 경우, 서비스 사업자의 OSS 시스템에서는 해당 장애를 자동으로 감지하고 이를 조치하는 과정에서 해당 사용자의 서비스 수준을 다음 단계로 업그레이드하여 사용자의 서비스 환경을 개선, 서비스 품질을 보장하는 방식을 구현하고 있다.

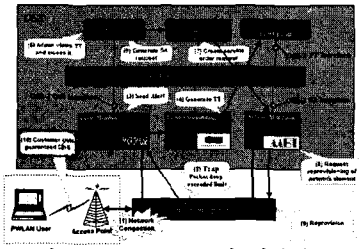

그림 13. Use case 3 동작 시나리오

(1) 네트웍에 장애 발생
(2) 관련 장비 관리시스템에서 해당 장애를 감지하고 이에 대한 trap 발생
(3) QoS 의 이벤트 핸들러 부분에서 경보를 발생한다.
(4) 서비스 모니터링 부분에서 관련 TT(Trouble Ticket)을 생성한다.
(5) 시스템 관리자는 Trouble Ticket 시스템에서 관련 TT 를 확인하고
(6) 관련 서비스 보증 요청을 보낸다.
(7) 관련 서비스 수준 업그레이드 요청을 보낸다.
(8) 관련 네트웍 장비의 구성 변경 요청을 보낸다.
(9) 해당 장비의 구성을 재 구성한다.
(10) 사용자는 계약된 서비스 품질로 서비스를 사용한다.

3.3 구현

관련 비즈니스는 SunONE Application Server 에서 OSS/J API 를 사용하여 관련 인터페이스를 개발하고 workflow 를 담당하는 Business Process Engine 을 개발하였다. 또한 request 및 response 의 처리는 메시징 방식으로 구현하였으며 이번 파일럿 구축에서는 별도의 메시지 버스 제품을 사용하여 구축하는 대신에 SunONE Application Server 에 내장된 JMS 기능을 이용하여 구축하였다.

3.4 구현 효과

본 파일럿은 OSS/J 를 활용하여 OSS 시스템 구축에 필요한 기능, 방법 및 아키텍처에 대한 검증 수준의 참조 구현이다. 이러한 참조 구현을 통하여 OSS/J 를 사용한 OSS 시스템 구축의 장점을 확인할 수 있었다.

4. 결론

본 논문에서는 현재 OSS 분야에서 활발하게 진행되고 있는 OSS/J 부분에 대한 배경, 의의 및 현재까지 진행된 관련 API 에 대한 분석 및 이를 활용한 파일럿 시스템을 구축함으로써 OSS/J 관련 부분에 대한 연구, 업계의 투자 및 선택이 어떠한 방향, 즉 장점 및 효과를 기대하면서 발전하고 있는 가에 대하여 살펴보았다.

OSS/J 는 OSS 시스템 구축 분야에서 관련 업체의 강한 지원을 받으면서 빠르게 발전하고 있는 부분으로 이 논문이 국내의 관련 업계의 종사자에게 OSS/J 의 의미와 효과를 제시할 수 있는 기회가 되어 이에 대한 연구, 개발 및 적용의 계기가 되었으면 한다.

참조 문헌

[1] TM Forum(TeleManagement Forum): *www.tmforum.org*
[2]Introduction to NGOSS: *http://www.tmforum.org/browse.asp?catID=1457&sNode=1457&Exp=Y&sync=False*
[3] OSS through Java™ J2EE Design Guidelines, OSS through Java™ Initiative, October 31, 2001
[4] OSS/J API Roadmap, OSS through Java™ Initiative, March 10, 2003
[5] Telecom Network Management with Enterprise JavaBeans™ Technology, Sun Microsystems, Inc., May 2001
[6] Business Demonstration Scenario Using OSS/J, Tom McSweeney et al., 2003
[7] Deployment Strategies focusing on Massive Scalability, Brian Naughton, 2003