

## 0. WebLogic Tuxedo Connector의 Introduction

### 0.1 WebLogic Tuxedo Connector Overview

WebLogic Tuxedo Connector (이하 WTC)는 WebLogic Server 애플리케이션과 Tuxedo 서비스 사이에 상호운용성(interoperability)을 제공하여 준다. WebLogic Server client가 Tuxedo 서비스를 호출할 수 있고 서비스 요청에 대한 응답으로 Tuxedo client 또한 WebLogic Server Enterprise Java Beans (EJBs)를 호출할 수게 한다.

다음과 같은 특징을 갖는다.

- 기존의 Client-to-Server 관계가 아닌 Peer-to-peer 방식이다.
- Jolt와 WLEC의 기능이 WTC에 추가되었다.
  - 양방향 상호 운영성 (Bi-directional interoperation)
  - tpenqueue와 tpdequeue 지원
  - 양방향 transaction 지원
  - 도메인 레벨의 failover / failback
- Tdomain 프로토콜을 사용한다.
  - 관리할 connection pool을 갖지 않는다.
  - JSL, JSH, Repository를 갖지 않는다.
- 버전 호환성
  - WLS 6.0, 6.1 - Tuxedo 6.5, .1, 8.0 (ATMI)
  - WLS 6.1 - Tuxedo 8.0 (CORBA)
- WTC는 WLS에 packaged 되어있다.
  - WLS 6.1부터 packaged
  - 더 이상 개별적으로 download할 필요가 없다.
  - 개별적으로 license가 필요 없다.
  - 구매를 위한 추가비용이 필요 없다.

### 0.2 주요 기능과 운영상의 특징

WTC는 Java Tuxedo ATMI와 유사 개념인 Application-to-Transaction Monitor Interface (JATMI)를 사용하여 WebLogic Server와 Tuxedo 통합 애플리케이션을 개발하고 지원할 수 있게 도와준다.

## Tuxedo + WTC + WLS = Flexibility

다음과 같은 면에서 유연성 (flexibility)을 제공한다.

- WLS로부터 Tuxedo Services (Transaction, conversations, security, queueing, CORBA) 호출
- WLP와 WLI 통합 (Integration)
- Tuxedo와 WLS 통합을 위해서 설계되어졌기에 WLS는 견고 (robust)하다.
- 추가적인 비용없이 WLS에 bundle 되어져 있다.
- WLS console을 통해서 WTC를 관리할 수 있다.

WTC는 다음과 같은 양방향 상호운용성 (bi-directional interoperability)을 제공한다.

- Tuxedo 애플리케이션에서 WebLogic Server 애플리케이션을 호출하거나 EJBs를 호출하게 하는 기능
- 기존 Tuxedo 환경에 WebLogic Server 애플리케이션을 통합하게 하는 기능
- Transaction 지원
- CORBA Java와 CORBA C++ Server 애플리케이션 사이에 상호 운영성을 제공하게 하는 기능
- eLink 1.2 adapter를 포함한 Tuxedo ATMI 서비스를 통해서 workflow를 관리하는 WebLogic Procee Integrator를 사용하게 하는 기능
- WebLogic Server와 Tuxedo 사이의 다수의 커넥션 (connection)을 정의하게 하는 기능

WTC는 다음과 같은 중요한 운영상의 특징을 포함한다.

- WTC는 기존 Tuxedo 애플리케이션 코드를 수정하는 것을 요구하지 않으므로 간단히 구현할 수 있다.
- 기존 Tuxedo clients는 WebLogic Server EJB를 WTC를 통해서 호출한다.
- 새롭게나 수정된 WebLogic Server clients는 WTC를 통해서 Tuxedo 서비스를 호출한다.

- Domain과 ACL 보안 (security)을 포함하는 양방향 보안 전파 (Bi-directional security propagation)
- Domain-level failover와 failback
- 향상된 메시징 서비스 (Advanced messaging 서비스)가 Tuxedo /Q와 JMS에 의해 제공된다.
- eLink를 사용하여 mainframe과 legacy 애플리케이션 사이의 상호 운영성 (interoperability) 제공

### 0.3 WTC와 Jolt와의 차이점

WTC는 Jolt를 대체 (replacement)하는 기능은 아니다. 다음에서 WTC와 Jolt의 차이점을 기술한다.

- WTC는 Jolt와 유사하지만 다른 API를 제공한다.
- Jolt는 WTC가 하지 못하는 Java client와 다른 Web Server 애플리케이션의 개발을 지원한다.
- Jolt는 통합된 WebLogic Server와 Tuxedo 트랜잭션 메커니즘을 제공하지는 못한다.

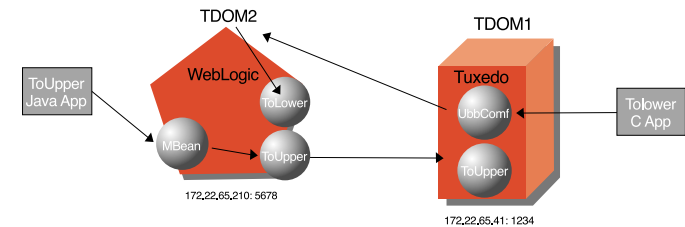
사용자들은 Java client나 다른 Web Server 애플리케이션이 요구되어지는 상황이나 WebLogic Server가 해결책이 되지 못하는 경우에 대신하여 사용할 수 있다.

### 0.4 관련문서

WTC에 관련문서는 다음 위치에서 얻을 수 있다.

- <http://www.bea.com> BEA 웹 사이트상에서 제품 문서 (Product Documentation) 부분을 클릭
- <http://e-docs.bea.com/index.html>로 직접 가서 WebLogic Server 7.0 documentation center의 링크를 따라가면 된다.

## 1. WebLogic 구성



▲ 그림 1 WTC 예제 시스템 구성

### 1.1 Weblogic TraceLevel 설정

SAMPLES\_HOME/server/config/examples/startExamplesServer.cmd 파일에 다음과 같은 TraceLevel 설정을 위한 옵션을 지정한다.

JAVA\_OPTIONS = -Dweblogic.wtc.TraceLevel=100000

### 1.2 Simpapp 예제 작성

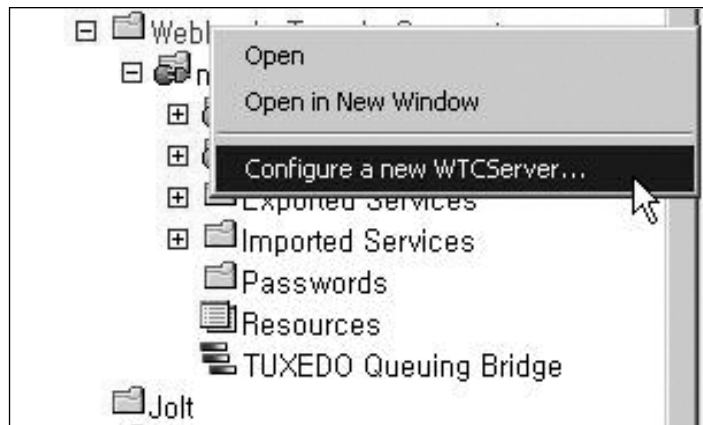
- 웹로직 examplesServer를 실행한다. (WEBLOGIC\_HOME/samples/server/config/examples/start ExamplesServer.sh)
- 새로운 shell을 열고 Shell에 환경 변수를 셋팅한다. (SAMPLES\_HOME/server/config/examples/setExamplesEnv.sh) .setExamplesEnv.sh
- 다음의 디렉토리로 이동한다. (SAMPLES\_HOME/server/src/examples/wtc/atmi/simpapp)
- ant를 실행시켜 wtc\_toupper.jar 파일을 만든다. wtc\_toupper.jar는 SAMPLES\_HOME/server/config/examples/applications 디렉토리에 생성된다.
- 디렉토리를 simserv로 이동한다. (SAMPLES\_HOME/server/src/examples/wtc/atmi/simserv)
- ant를 실행시켜 wtc\_tolower.jar 파일을 생성한다.
- WebLogic console

```
[BEAAPP2#/app/WLP7.0.1/weblogic700/samples/server/config/example/applications] ls
DefaultwebApp      ejb20_homemethods.ear  readme.txt
ejb20_basic_beanManaged.ear  ejb20_message.ear      webservices_trader_ear
ejb20_basic_containerManaged.ear  jdbc_oracle_ext.ear      wtc_tolower.jar
ejb20_basic_statefulSession.ear  jms_messageformat.jar    wtc_toupper.jar
ejb20_basic_statelessSession.ear  jta_ejb_jmsjdbc.jar      xml_xsit_content.ear
```

▲ 그림 2 ant에 의해서 생성된 ejb



### ◀ 그림 3 배포된 EJB 확인



### 1.3 WTCServer MBean 생성하기

- Services Node를 선택한다.
- WebLogic Tuxedo Connector Node의 오른쪽을 클릭한다.
- WTCServer의 이름필드에 “mySimapp”를 입력한다.
- “Create”를 클릭한다.

#### ◀ 그림 4 새로운 WTCServer 생성



#### 1.4 Local WLS Domain 생성

- Local WLS Domain의 오른쪽 클릭
- “Configure a New WTCLocalTuxDom” 선택
- “General” 탭에서 다음과 같이 입력한다.  
 Access Point : TDOM2  
 AccessPoint ID : TDOM2  
 Network Address : 〈WebLogic Server에 대한 IP와 Port〉  
 예) //123.123.123.123:5678
- “Create” 클릭
- Tuxedo 6.5 도메인과 연결하려고 한다면 다음과 같은 절차를 추가해야 한다.
  - a. “Connections” 탭 선택
  - b. Interoperate 필드 값을 “Yes”로 설정한다.
  - c. Apply를 클릭한다.

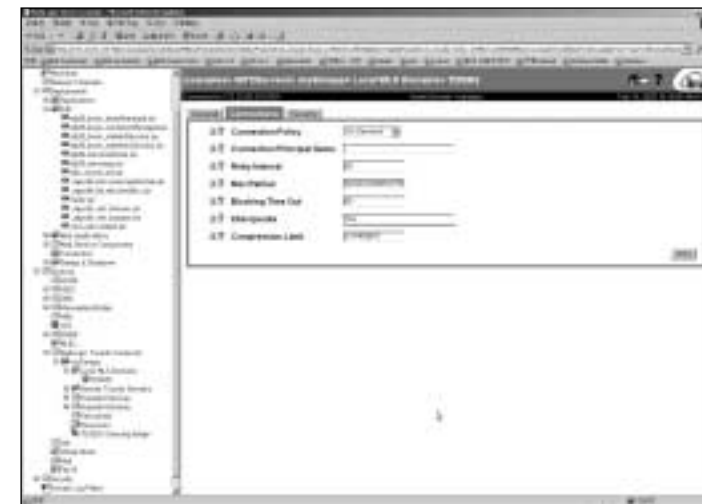
◀ 그림 5 *Simpapp WTCServer MBean* 생성



## 1.5 Remote Tuxedo Domain 생성

- "Remote Tuxedo Domain" 선택
- "Configure a New WTCRemoteTuxDom" 선택
- "General" 탭에서 다음의 필드 값들을 입력한다.  
Access Point: TDOM1  
AccessPoint ID: TDOM1  
Local Address Point: TDOM2  
Listen Address: 〈Tuxedo 서버의 IP와 포트〉  
예) //123.123.123.123:1234
- "Create" 클릭

### ◀ 그림 6 Local WLS Domain 설정



## 1.6 Exported Services 생성

- "Exported Services" 선택
- "Configure a New WTCExport" 선택
- "General" 탭에서 다음의 값들을 입력  
 Resource Name : TOLOWER  
 Local Access Point : TDOM2  
 EJB Name : tuxedo.services.TOLOWERHome
- "Create" 를 클릭한다.

#### ◀ 그림 7 Tuxedo 6.5 설정



### 1.7 Imported Services 생성

- "Imported Services" 선택
- "Configure a New WTCImport" 선택
- "General" 탭에서 다음과 같은 값들을 입력  
Resource Name : TOUPPER  
Local Access Point : TDOM2  
Remote Access Point List : TDOM1
- "Create" 생성

◀ 그림 8 WTC Exported Service 구성



◀ 그림 9 WTC 중 Imported services 구성



- 1.8 WTCServer MBean 반영
- 새로이 구성한 "mySimap" 선택
  - "Targets" 선택
  - "examplesServer" 선택
  - "오른쪽 화살표" 선택
  - "Apply" 클릭

◀ 그림 10 WTCServer MBean 배포



- 1.9 TDOM1 Weblogic 사용자 등록
- "Security" 선택
  - "Realms" 선택
  - "default security Realm" 선택
  - "Users" 선택
  - "Configure a new user text" Link 선택
  - "DefaultAuthenticator" 선택
  - "General" 탭 선택
    - a. 이름 필드에 "TDOM1"
    - b. password 입력
    - c. "Apply" 선택

◀ 그림 11 TDOM WebLogic 사용자 등록

2. Tuxedo 구성

2.1 복사본을 만든다.

\$TUX\_HOME/apps에 있는 simpapp 디렉토리를 복사하여 임의의 디렉토리를 생성한다.

2.2 Tuxedo 환경 파일을 만든다.

다음과 같은 내용으로 setenv 파일을 생성한다.

Tuxedo 환경 구성을 위한 Shell 설정	
<pre>export APPDIR = /home1/wkhan/tux65/apps/simpapp_yjkim export TUXCONFIG = /home1/wkhan/tux65/apps/simpapp_yjkim/tuxconfig export BDMCONFIG = /home1/wkhan/tux65/apps/simpapp_yjkim/bdmconfig export TUXDIR = /home1/wkhan/tux65  export LD_LIBRARY_PATH=\$TUXDIR/lib:\$LD_LIBRARY_PATH export PATH=\$TUXDIR/bin:\$PATH</pre>	

표 1 Tuxedo 구성을 환경 설정 Shell

Shell 환경 적용은

"[chaeju# /home1/wkhan/tux65/apps/simpapp\_yjkim]. Setenv" 같이 한다.

2.3 컴파일을 한다.

서버 모듈 컴파일

buildserver -o simpserv -f simpserv.c -s TOUPPER

클라이언트 모듈 컴파일

buildclient -o simpcl -f simpcl.c

2.4 Ubbconfig 을 수정한다.

Ubbdomain이라는 파일명으로 ubbconfig 파일을 작성한다.

Ubbconfig 파일

#Skeleton UBBCONFIG file for the TUXEDO Simple Application.  
#Replace the <bracketed> items with the appropriate values.

```
*RESOURCES
IPCKEY          55577
#Example:
#IPCKEY          123456
DOMAINID        simpapp
```

```
MASTER          simple
MAXACCESSERS    50
MAXSERVERS      25
MAXSERVICES     50
MODEL           SHM
LDBAL           N

*MACHINES

DEFAULT:
    APPDIR="/home1/wkhan/tux65/apps/simpapp_yjkim"
    TUXCONFIG="/home1/wkhan/tux65/apps/simpapp_yjkim/tuxconfig"
    TUXDIR="/home1/wkhan/tux65"

chaeju          LMID=simple

*GROUPS
GROUP1
    LMID=simple    GRPNO=1 OPENINFO=NONE

GROUP2
    LMID=simple    GRPNO=2 OPENINFO=NONE

*SERVERS
DEFAULT:
    CLOPT="-A"

simpserv        SRVGRP = GROUP1 SRVID = 1

DMADM           SRVGRP = GROUP2 SRVID = 1
GWADM           SRVGRP = GROUP2 SRVID = 2
GWTDOMAIN       SRVGRP = GROUP2 SRVID = 3

*SERVICES
TOUPPER
```

표 2 WTC 를 사용하기 위한 Ubbconfig 파일 예제

2.5 tmloadcf

텍시도를 실행한다.

tmloadcf -y ubbdomain

참고) 텍시도 다운 커맨드: tmshutdown -y

2.6 tuxedo 서비스 테스트

[chaeju# /home1/wkhan/tux65/apps/simpapp\_yjkim]simpcl dkdkdkd  
Returned string is : DKDKDKD

2.7 domain Config

dom1config이라는 이름으로 domain config 파일을 작성한다.

Domain Config
<pre>*DM_RESOURCES VERSION=U22 *DM_LOCAL_DOMAINS TDOM1  GWGRP=GROUP2         TYPE=TDOMAIN         DOMAINID="TDOM1"         BLOCKTIME=20         MAXDATALEN=56         MAXRDOM=89         DMTLOGDEV="/home1/wkhan/tux65/apps/simpapp_yjkim/tlog"         AUDITLOG="/home1/wkhan/tux65/apps/simpapp_yjkim/aud"         DMTLOGNAME="DMTLOG_TDOM1  *DM_REMOTE_DOMAINS TDOM2  TYPE=TDOMAIN         DOMAINID="TDOM2"  *DM_TDOMAIN TDOM1  NWADDR="//172.22.65.41:1234" TDOM2  NWADDR="//172.22.65.210:5678"  *DM_REMOTE_SERVICES TOWER RDOM="TDOM2"</pre>

표 3 WTC 를 사용하기 위한 Domain Config

2.8 dmloadcf

dmloadcf -y dom1config

2.9 도메인 환경 하에서의 테스트 클라이언트 작성

WEBLOGIC WTC 예제에 있는 것을 작성

Tolower.c를 컴파일할 때 에러가 발생할 수 있으며 혹시 에러가 발생하면  
에러 메시지를 잘 살펴보고 소스 컴파일한다.

Buildclient -v -f tolower.c -o tolower

2.10 텍시도 서버 실행

tmboot -y

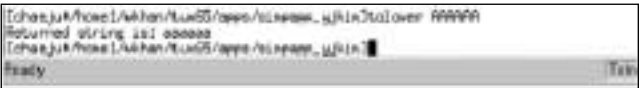
2.11 통합 테스트

2.11.1 WebLogic 에서 Tuxedo 호출



▲ 그림 12 WebLogic에서 Tuxedo 호출

2.11.2 Tuxedo에서 WebLogic 호출



BEA WebLogic Server 7.0에서의 Session Persistence Performance

Application server와 상호작용하면서 user session을 유지하는 방식이 hidden HTML field와 URL을 통한 방식에서 안정적인 J2EE 기반의 HTTP session 방식으로 발전해 오고 있다.

Session은 application architecture에서 다양한 계층에 위치할 수 있다. 이러한 다양한 계층은 client, presentation, object layer 또는 database layer가 될 수 있다.

Session data가 client layer에 위치하는 방법에는 두 가지가 있는데 hidden HTML field나 cookie를 통한 저장 방법이다. 이러한 방법은 session data 양이 많아지면 관련 Network 부하를 증가시키는 문제와 Cookie의 경우 String값 만을 사용할 수 밖에 없는 단점들이 있다.

성능, 신뢰성, 사용성 차원에서는 Presentation 또는 Web Application layer에서 session을 유지하는 것이 좀더 나은 방법이다. 이 방법은 session data를 유지하는데 추가적인 option을 제공하며 network overhead를 줄이며 session data type과 object size에 대한 제약이 없다. 또한 server나 cluster가 crash났을 때에 failover mechanism을 제공한다. 게다가 이러한 접근방법은 application server-specific 기능들(예를 들어 cluster)을 이용할 수 있으며 이에 따라 신뢰성, 확장성, 성능면에 향상을 가져온다.

Session data는 또한 stateful EJB를 통해 object layer에 저장될 수 있다. 이 방법은 stateful EJB의 Handle을 저장하기 위해 Web application layer를 사용한다. 이 방식은 application data가 주로 object layer상에 있을 때, Web application layer가 system에 대한 주 interface로 사용될 때 좋다. 이러한 조합은 결과적으로 가벼운 HTTP session을 통한 application 성능 향상을 가져올 수 있다. 그러나 EJB activation과 passivation에 의한 부하를 신중히 고려해 보아야 한다.

WLS7.0은 다양한 application적 요구에 따른 적절한 persistence mechanism을 선택할 수 있도록 여러 방법을 제공한다. Session 관리는 사용자에게 완전히 투명하며 deployment descriptor file을 통해 쉽게 구성될 수 있다. 이 문서는 다양한 HTTP session persistence mechanism과 각각의 방법들의 session data size에 따른 성능적 측면에 중점을 두고 있다. 5가지의 session persistence 방식은 다음과 같다.

- 1. Memory(single – server, nonreplicated) – based
- 2. Cookie – based
- 3. File system persistence
- 4. JDBC persistence
- 5. In-memory replication(across a cluster)

Memory-Based Session Persistence

Session 관리의 default mechanism이다. 여기서 memory는 Web application이 배치되고 수행되는 server(JVM process)상의 memory를 지칭한다. Session data는 session object의 life cycle동안 JVM process memory상에서 유지된다. Memory-based, single-server, nonreplicated persistent storage를 사용하기 위해서는 WLS-specific deployment descriptor상의 <session-descriptor> element에 PersistentStoreType property를 memory로 설정하면 된다. 모든 session 정보는 memory에 저장되므로 WLS가 stop되고 restart되면 소실 된다.

이 방법은 성능면에서는 뛰어나나 failover를 지원하지 못한다. Benchmark 결과에 의하면 session data의 크기에 상관없이 안정적인 performance를 낸다 (see Figure1).

