Integrasjon mellom Java 2™ Platform Enterprise Edition og Microsoft DNA i Objectnets driftsmiljø.

Hovedoppgave

ved

sivilingeniørutdanningen i

informasjons- og kommunikasjonsteknologi

av

Per Ivar Pedersen

Arendal, mai 2001

1 Forord

Etter mange lange dager, kvelder og setninger over flere måneder så har jeg endelig nådd målet med å sluttføre den avsluttende hovedoppgaven ved sivilingeniørstudiet på HiA. Denne oppgaven har vært inspirerende for meg fordi den har gitt meg muligheten til fordypning innen et område jeg er spesiellt interessert i, nemlig komponentbasert utvikling og integrasjon mellom komponenter i heterogene miljøer. Det har samtidig vært en utfordring for selvdisiplin og disponering av tid ettersom jeg parallelt med skriving av oppgaven har arbeidet ved Objectnets avdeling i Arendal. Jeg vil derfor benytte anledningen til å takke Objectnet som har gjort det mulig for meg å kombinere jobb med oppgaveskriving og vist stor fleksibilitet rundt dette.

En spesiell takk til Knut-Håvard Aksnes ved Orion Systems for bidrag til konfigurasjon av TAO som er brukt til eksperimentering med CORBA. Mine veiledere Jan Pettersen Nytun, Øyvind Hansen og Mikael Snaprud må også takkes for positiv innstilling, konstruktiv kritikk og interesse for problemstillinger underveis.

Sist men ikke minst vil jeg takke min samboer, Solveig, som har holdt ut med en overarbeidet sjel som det siste halve året har hatt vel mye fokus på hovedoppgave fremfor hjemlige sysler.

2 Sammendrag

Denne rapporten foretar en vurdering av hvilke mekanismer for integrasjon av de to EIS (Enterprise Information Systems) J2EE og Microsoft DNA som kan brukes for å realisere integrasjon av business komponenter. Det er lagt fokus på å skjerme klientapplikasjoner for kompleksiteten i integrasjonen. Prosessen er avgrenset til å involvere EJB (Enterprise Java Beans) og COM+ komponenter der EJB er proxy for COM+ og omvendt.

I hovedsak er de tre hovedområdene bridging, mekanismer for RPC over HTTP og IIOP transport vurdert hver for seg og mot hverandre. Det er foretatt testing innenfor hvert hovedområde som danner grunnlag for drøfting. Det vurderes styrker og svakheter innen områder som åpenhet, sikkerhet og transaksjoner.

Rapporten starter innledningsvis med å generalisere rundt EIS systemer for så å tilnærme seg avgrenset problemområde. Deretter følger inndeling i de tre nevnte hoveddelene. Innenfor hver hoveddel utdypes teknologien samt at det gjennomføres eksperimenter som ender i refleksjoner og drøfting. Verktøy og metoder til eksperimentene tar utgangspunkt i lisensiert og gratis software i regi av Objectnet as.

Hovedområdet med bridging vurderes og testes med utgangspunkt i IIOP og DCOM som protokoller for transport. Det eksperimenteres her med bridger fra Microsoft og JACOB. Hovedområdet med RPC over HTTP tar for seg SOAP-RPC som mekamisme for å kalle metoder fra J2EE mot Windows DNA. Det eksperimenteres her med Apache SOAP og Microsoft SOAP Toolkit. Hovedområdet med IIOP vurderer TAO implementasjonen av CORBA som en mulig kandidat for å integrere COM+ komponenter mot EJB.

3 Innholdsfortegnelse

1 FORORD II

2 SAMMENDRAG III

3 INNHOLDSFORTEGNELSE IV

4 LISTE OVER FIGURER OG TABELLER VII

5 INNLEDNING 1

5.1 Enterprise Informasjonssystemer (EIS) 1

5.2 Java 2™ Platform, Enterprise Edition (J2EE) 3

5.2.1 Kilder 4

5.3 Microsoft™ DNA 5

5.3.1 Kilder 5

5.4 Microsoft™ DNA vs. J2EE. 6

5.4.1 Kilder 7

5.5 Oppdragsgiver 8

5.6 Problemområde 8

5.7 Avgrensninger 8

6 MATERIALE OG METODER 10

6.1 Materiale 10

6.2 Metoder 10

6.3 UML 11

6.4 Kilder 11

7 TESTMILJØ 12

7.1 Verktøy 12

7.1.1 Together 4.1 12

7.1.2 Visual Studio 6.0 12

7.1.3 Microsoft SQL Server 2000 Developer Edition 12

7.1.4 Windows 2000 Professional 13

7.1.5 WebLogic Server 5.1 13

7.1.6 Apache SOAP 13

7.1.7 Microsoft SDK for Java 4.0 13

7.1.8 TAO 13

7.2 Oppsett av software 14

7.3 Repository 14

7.4 Grunnleggende Java miljø 15

7.5 Grunnleggende COM miljø ... 16
7.6 Kilder ... 17

8 BRIDGING ... 19
8.1 Innledning ... 19
8.2 Bridging mellom J2EE og COM+ ... 19
8.3 Valg til eksperiment ... 20
8.3.1 Microsoft SDK for Java 4.0 ... 20
8.3.2 Java-COM Bridge (JACOB) ... 21
8.4 Gjennomføring av eksperiment ... 24
8.4.1 COM mot Java – Microsoft SDK for Java 4.0 ... 24
8.4.2 Java mot COM - JACOB ... 30
8.5 Konklusjon ... 35
8.6 Kilder ... 36

9 SIMPLE OBJECT ACCESS PROTOCOL (SOAP) ... 38
9.1 Innledning ... 38
9.1.1 Brannmurer og SOAP ... 39
9.1.2 Sikkerhet i SOAP ... 39
9.2 Valg til eksperiment ... 39
9.2.1 MS SOAP Toolkit ... 39
9.2.2 Apache SOAP ... 41
9.3 Gjennomføring av eksperiment ... 43
9.3.1 Java mot COM - SOAP ... 43
9.4 Konklusjon ... 53
9.5 Kilder ... 53

10 CORBA ... 56
10.1 Innledning ... 56
10.1.1 CORBA ... 56
10.1.2 J2EE og CORBA ... 56
10.2 Valg til eksperiment ... 57
10.2.1 WebLogic Server 5.1 ... 57
10.2.2 TAO ... 57
10.2.3 Oversikt ... 59
10.3 Gjennomføring av eksperiment ... 59
10.3.1 IIOP som transport mellom COM+ og EJB ... 59
10.4 Konklusjon ... 65
10.5 Kilder ... 65

11 GENERELL DRØFTING ... 67

12 KONKLUSJON 69

VEDLEGG I. VERKTØY GRENSESNITT 71

A. Together 4.1 IDE Grensesnitt 71

B. Visual Studio 6.0 IDE Grensesnitt 72

VEDLEGG II. PROGRAMKODE 73

A. setEnvMS.cmd 73

VEDLEGG III. PROGRAMKODE GRUNNL. J2EE TESTMILJØ 75

A. Build.cmd 75

B. ejb-jar.xml 75

C. weblogic-ejb-jar.xml 76

D. EmployeeHome.java 76

E. Employee.java 77

F. EmployeeBean.java 77

G. HourHome.java 81

H. Hour.java 81

I. HourBean.java 82

VEDLEGG IV. PROGRAMKODE FOR GRUNNL. COM+ TESTMILJØ 87

A. Diplom.Project class 87

B. Diplom.vbp 87

VEDLEGG V. PROGRAMKODE JAVA-COM JACOB TEST 88

A. Build.cmd 88

B. ejb-jar.xml 88

C. weblogic-ejb-jar.xml 88

D. ProjectHome.java 89

E. Project.java 89

F. ProjectBean.java 89

G. Client.java 90

H. go.cmd 92

VEDLEGG VI. SQL SKRIPT FOR Å LAGE TABELLER TIL TESTMILJØ 93

A. Diplom1 database tabeller 93

B. Diplom2 database tabeller 93

4 Liste over figurer og tabeller

Figur 1: To-lags applikasjonsmodell 2
Figur 2: Flerlags applikasjonsmodell 2
Figur 3: J2EE arkitektur 3
Figur 4: Grafisk presentasjon av problemområde 9
Figur 5: Repository for testmiljøet 15
Figur 6: Grunnleggende J2EE testmiljø konsept (Employee EJB) 16
Figur 7: Konsept for bridging mellom COM+ og J2EE. 19
Figur 8: COM-Java via Microsoft SDK for Java eksperimentell prosedyre 25
Figur 9: Grensesnitt for klient applikasjon mot COM+ Employee proxy 29
Figur 10: Java-COM via JACOB eksperimentell prosedyre 31
Figur 11: Struktur skisse for Java til COM via SOAP 44
Figur 12: Java-COM via SOAP eksperimentell prosedyre 47
Figur 13: Sporing av SOAP-RPC medlinger 50
Figur 14: Oversikt over ACE+TAO ORB 58
Figur 15: COM+ via CORBA mot EJB eksperimentell prosedyre 59

Tabell 1: Java API som kreves av J2EE 4
Tabell 2: Sammenlikning mellom Windows DNA og J2EE 7
Tabell 3: WebLogic konfigurasjon for tilgang mot repository 15
Tabell 4: WebLogic konfigurasjon for grunnleggende EJB komponenter 16
Tabell 5: Grunnleggende COM+ testmiljø konsept (COM+ Project) 17
Tabell 6: Eksponere Java klient klasser som trusted for Microsoft SDK for Java 26
Tabell 7: Programkode for COM+ Employee proxy komponent 27
Tabell 8: Kode for å opprette instans av COM+ Employee med VBA 27
Tabell 9: Klient applikasjon mot COM+ Employee proxy 28
Tabell 10:Avhengigheter for EJB Project proxy med JACOB 32
Tabell 11: Implementasjon av EJB Project proxy metode med JACOB 32
Tabell 12: JAVA_CLASSPATH for WebLogic med JACOB støtte 32
Tabell 13: Deploy konfigurasjon for EJB Project med JACOB 32

Tabell 14: Policy konfigurasjon for WebLogic med JACOB støtte 33
Tabell 15: Implementasjon av klient for JACOB test 33
Tabell 16: Resultater fra Java mot COM timing med JACOB 34
Tabell 17: SQL spørring for å hente prosjektliste 46
Tabell 18: ASP skript kode for å linke IIS mot ProjectSrv 47
Tabell 19: Avhengigheter for EJB Project proxy med SOAP 48
Tabell 20: Implementasjon av EJB Project proxy metode med SOAP 49
Tabell 21: Implementasjon av klient mot EJB Project proxy 49
Tabell 22: WebLogic classpath for Apache SOAP integrasjon 50
Tabell 23: Deploy properties for EJB Project SOAP proxy 50
Tabell 24: Resultater fra Java mot COM timing med SOAP 51
Tabell 25: RMI over IIOP konfigurasjon for WebLogic 60
Tabell 26: Generert og revidert IDL for Hello klient 61
Tabell 27: IOR referanse til navnetjener i WebLogic Server 5.1 61
Tabell 28: CORBA klient med TAO mot WebLogic Server 62
Tabell 29: Implementasjon av Hello COM+ proxy med TAO 63

5 Innledning

5.1 Enterprise Informasjonssystemer (EIS)

Internett har tilført betydelig kompleksitet til EIS, som nå må supportere forretningsoperasjoner som inkluderer å integrere eksterne forretningspartnere og kunder. Enterprise systemer i dag og for fremtiden søker hele tiden å utvide sine horisonter i form av kostnadsreduksjoner og redusert responstid ved å tilby kunder, partnere, ansatte og leverandører enkel aksess til systemene. Karakteristikken for det fremtidsrettede EIS er typisk at flere EIS kombineres ved å tilføre funksjonalitet i form av forretningstjenester, samt at tjenestene enkelt kan gjøres tilgjengelig for mange brukere. EIS applikasjoner består typisk av navnetjenere for lokalisering av ressurser, enhetlig styring av sikkerhet og transaksjoner, meldingstjenester for asynkron integrasjon av systemkomponenter, samt kopling mot registre som databaser. Det stilles krav til sikkerhet, tilgjengelighet, robusthet, skalerbarhet, transaksjonell integritet og distibusjon.

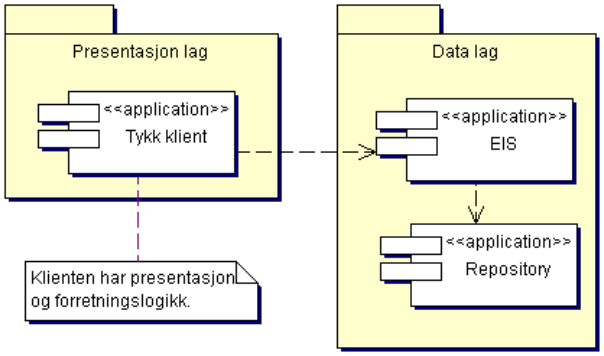
Internett er stort. Tidligere skulle EIS supportere opp til flere tusen samtidige brukere. Idag og mot fremtiden, ettersom elektroniske forretningsområder skrider frem, må EIS i sin ytterste koneskvens kunne støtte opp til millioner av samtidige brukere. Det stilles derfor store krav til skalerbarhet i dagens og fremtidens EIS.

Internett stopper aldri opp. Tilgjengelighet er et viktig begrep i dagens og fremtidens EIS. Et EIS må ofte være kontinuerlig tilgjengelig. Det stilles krav til at EIS kan vedlikeholdes og utvides uten at driften må stanses.

Internett har ingen vegger. Et EIS må ofte være tilgjengelig over Internett. Man kan ikke lenger plassere EIS bak fysiske sperrer for å beskytte system og informasjon. Hvem som helst kan potensielt infiltrere et EIS som er knyttet til Internett. EIS mot Internett krever forskjellige metoder for å identifisere og autentisere brukere samt beskytte EIS mot uautorisert tilgang. EIS må også gi mulighet for å skjerme identiteten til brukere og kunder.

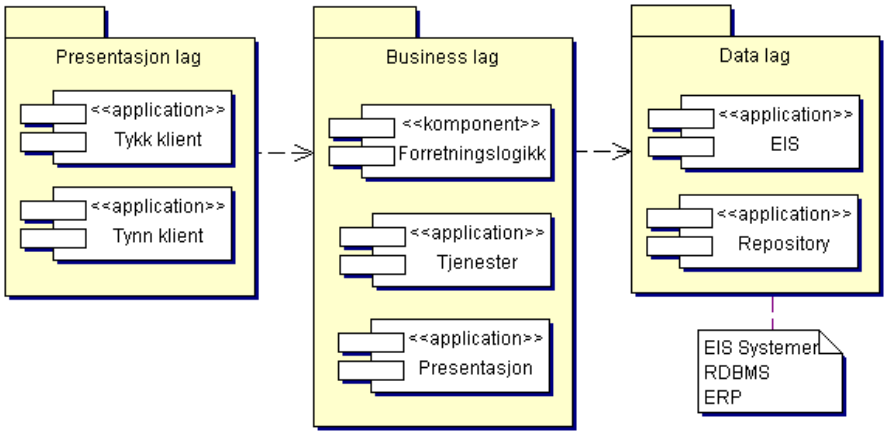
Tradisjonelle EIS har blitt basert på to-lags klient-tjener applikasjonsmodell (Figur 1). Denne modellen lovte forbedret skalerbarhet og funkjonalitet. Forretningslogikk implementeres i applikasjonen som plasseres ute hos klientene. To-lags modellen har vist en rekke svakheter i forhold til dagens krav til EIS. Installering og versjonering av applikasjoner ute hos klienten er ressurskrevende. EIS har i nyere tid tatt

et sceneskifte med fokus og vekt på å samle forretningslogikk i et business lag f.eks. på en delt serverløsning.



Figur 1: To-lags applikasjonsmodell

Et av dagen heteste temaer innen EIS er distribuerte, lagdelte løsninger for applikasjoner. Distribuerte applikasjoner deles ofte i flere logiske eller fysiske lag (Figur 2). Klientapplikasjonen er lokalisert i presentasjonslaget, repository er lokalisert i data laget, mens forretningslogikk og andre felles tjenester er lokalisert i business laget. Hoveddelen av software utvikling i EIS er konsentrert i business laget. Her integreres ofte flere eksisterende EIS ved implementasjon av forretningstjenester. På denne måten blir klienter skjermet fra kompleksiteten i Enterprise systemets forretningslogikk. Business laget kan supplementere tykke klienter som selv styrer presentasjon og interaktivitet samt tynne klienter som nettlesere der en serverløsning genererer presentasjonen.



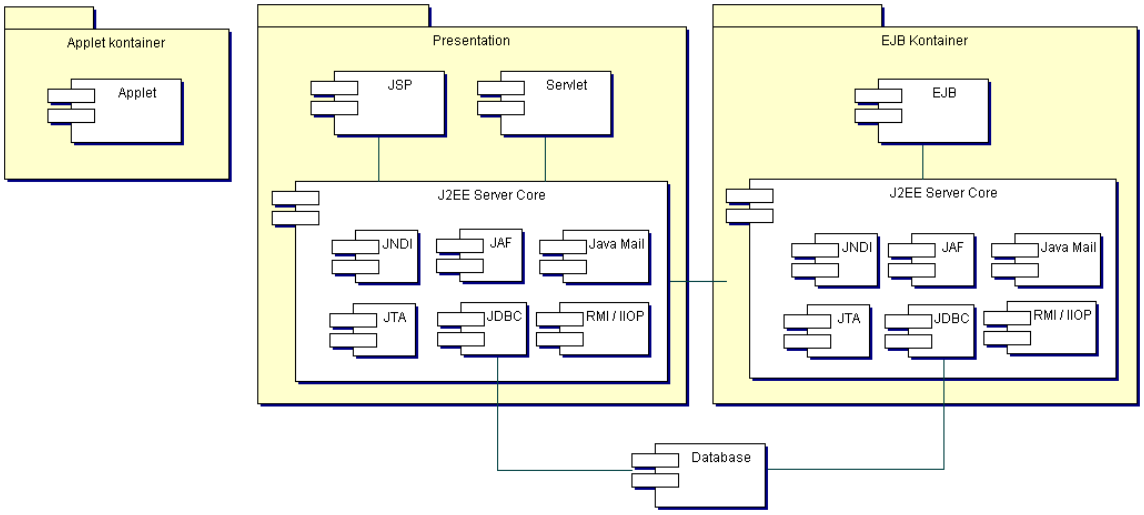
Figur 2: Flerlags applikasjonsmodell

En av de mest utfordrende oppgavene knyttet til den nye flerlags EIS modellen er å kople sammen flere ulike EIS systemer til et enhetlig forretningsområde. Det kan f.eks. være eksterne systemer for bestilling av varer hos leverandører eller annen type kommunikasjon. Det er da ofte ønskelig at kommunikasjon mellom flere EIS

implementeres i forretningslaget. Dette medfører at flere EIS fremstår for klient applikasjoner som ett enhetlig EIS. På dette området er det enda ikke fremskredet en felles standard som alle EIS støtter. Det er derfor nødvendig med eksterne applikasjoner og protokoller for å lime forskjellige EIS sammen til enhetlige forretningstjenester.

5.2 Java 2™ Platform, Enterprise Edition (J2EE)

12. april 1997 annonserte Sun Microsystems, Inc at de ønsket å lede arbeidet med å utvikle spesifikasjon for Java Plattform for Enterprise. Sun koordinerte utviklingen av standard Java utvidelse som i dag er kjent som Enterprise Java™ API. Disse API-ene definerer grensesnitt for mellomvare som er uavhengige av leverandør. Det viktigste bidraget til Enterprise Java™ API er Enterprise JavaBeans™, som definerer en serverside modell for utvikling av software komponenter og et grensesnitt for implementasjoner av Java applikasjonstjenere uavhengig av leverandør. I dag er spesifikasjonen for Enterprise Java API-ene versjonert til Java 2™ Platform, Enterprise Edition og implementasjoner er tilgjengelige fra leverandører som blant annet BEA Systems, IBM, Inprise med flere. Enterprise JavaBean (EJB) har blitt en foretrukket komponentmodell for Java applikasjonstjenere. Det er per idag mer enn tyve EJB implemetasjoner tilgjengelig fra forskjellige leverandører.



Figur 3: J2EE arkitektur

Figur 3 viser konseptuell oversikt over J2EE arkitekturen. Applet er applikasjonskomponenter i presentasjonslaget. JSP og Servlet er programkomponenter som genererer presentasjon i form av HTML og XML og kjører på server-siden og gir tilgang til EIS back-end systemer. J2EE Server Core er grunnkomponentene som tilbyr

tjenestene som er definert for J2EE plattformen. EJB er applikasjonskomponenter som skjermer klient fra EIS back-end og implementerer forretningslogikk.

Tabell 2 viser hvilke grunnkomponenter i J2EE arkitekturen som er påkrevd for de tre grunnkomponenter.

Standard utvidelse	Applet	Servlet	EJB
J2SE 1.2	J	J	J
JDBC 2.0	N	J	J
RMI / IIOP 1.0	N	J	J
EJB 1.1	N	N	J
Servlets 2.2	N	J	N
JavaServer Pages 1.1	N	J	N
JNDI 1.1	J	J	J
JTA 1.0	N	J	J
JavaMail 1.1	N	J	J
JavaBeans Activation Framework 1.0	N	J	J

Tabell 1: Java API som kreves av J2EE

5.2.1 Kilder

Enterprise JavaBeans™ Specification, 1.1 - Final Release

© Sun Microsystems, Inc.

http://e-docs.bea.com/wle/pdf/ejb1_1-spec.pdf

Java™ 2 Platform Enterprise Edition Specification, v1.2

© Sun Microsystems, Inc. Final release. Bill Shannon

http://www.orionserver.com/docs/j2ee/j2ee1_2-spec.pdf

JavaServer Pages™ Specification, Version 1.1

© 1999 Sun Microsystems, Inc. Eduardo Pelegri-Llopart and Larry Cable

http://ioctl.org/doc/jsp-spec.pdf

Java™ Servlet Specification, v2.2

© 1999 Sun Microsystems, Inc.James Duncan Davidson and Danny Coward

http://atlassw1.phy.bnl.gov/doc/servlet-2_2/servlet2_2-spec.pdf

JDBC™ 2.1 API

© 1999 Sun Microsystems, Inc. Seht White and Mark Hapner

http://nenya.ms.mff.cuni.cz/~prochazk/papers/tp/jdbc2_1-spec.pdf

JavaMailTM API Design Specification Version 1.1
© Sun Microsystems, Inc.
http://java.sun.com/products/javamail/JavaMail-1.1.pdf

Java™ Transaction Service (JTS) 1.0
© Sun Microsystems, Inc.
http://java.sun.com/products/jts/

5.3 Microsoft™ DNA

Microsoft utviklet Windows Distributed interNet Application Architecture (Windows DNA) som en platform for å integrere Web med flerlags modellen for utvikling av applikasjoner. Windows DNA definerer et rammeverk for å levere løsninger som kan møte kravene til redusert utviklingstid og reduserte kostnader ved integrasjon mot Internett, Intranett og elektronisk handel.

Windows DNA arkitektur gir standard Windows baserte tjenester mulighet til å kunne implementeres i hvert lag i en flerlags appklikasjonsløsning. Dette omfatter grensesnitt mot bruker, forretningslogikk og datalagring. Tjenestene som tilbys gjennom Windows DNA innebefatter Dynamisk HTML (DHTML), Active Server Pages (ASP) COM komponenter, COM+ komponent server, Active Directory navnetjenester, Microsoft Security, Microsoft Message Queuing og Microsoft Data Access Components (MDAC).

Windows DNA bruker åpne protokoller og publiserte grensesnitt som gjør det relativt enkelt å integrere produkter for Windows DNA fra flere leverandører. I tillegg støttes industristandarder for Internett, noe som skal gjøre det enkelt for utviklere å følge nye trender og endringer i teknologi.

5.3.1 Kilder

Microsoft Business, Web Site
© Microsoft Corporation, Inc.
http://microsoft.com/business/products/webplatform/default.asp

Microsoft COM, Component Object Model, Web Site
© Microsoft Corporation, Inc.
http://www.microsoft.com/com/default.asp

Microsoft Universal Data Access, Web Site
Microsoft Corporation, Inc.
http://www.microsoft.com/data/

Active Server Pages, MSDN Web Site
© Microsoft Corporation, Inc.
http://msdn.microsoft.com/library/psdk/iisref/iiwawelc.htm

Message Queuing Overview and Resources, Windows NT Server Web Site
Microsoft Corporation, Inc.
http://www.microsoft.com/ntserver/appservice/exec/overview/MSMQ_Overview.asp

5.4 Microsoft™ DNA vs. J2EE.

Det er få plattformer som hevder seg blant EIS. De eneste reelle konkurentene er for tiden J2EE, CORBA og Microsoft DNA. J2EE implementeres over CORBA teknologi. De fleste leverandører som leverer plattformer for enterprise løsninger velger å implementere applikasjonsservere for CORBA og J2EE i tillegg til egne propritære løsninger istendenfor å konkurrere. J2EE bruker IIOP, som derved sikrer interoperabilitet med CORBA implementasjoner (BEA Systems, IBM, Inprise og Iona m.f.). Det åpner for at EIS som støtter CORBA kan kommunisere med J2EE. CORBA representerer mere en realisering av infrastruktur i J2EE enn en konkurrent.

Windows DNA representerer tre modeller for komponenter i distribuerte applikasjoner. Disse er ActiveX, Active Server Pages (ASP) og COM+. Modellene i Windows DNA kan koples en-til-en mot tilsvarende modeller i J2EE. ActiveX korresponderer med Applets. ActiveX komponenter kan flyttes over Internett og kjøre som en komponent i en nettleser som støtter ActiveX, som f.eks. Internet Explorer (IE). ASP korresponderer med servlet og JSP. ASP er skript som kjøres under Microsoft Information Server (IIS) som igjen kopler nettleser-applikasjoner til forretningslogikk og repositorier. En ASP applikasjon prosesserer HTTP meldinger og genererer resultatet i HTML eller XML. COM+ korresponderer med EJB komponenter. En COM+ komponent kjører i en COM+ applikasjonsserver og implementerer forretningstjenester mot datalaget.

Selv om disse to enterprise applikasjonstjenerene har konseptuelle likheter, så er det flere fundamentale forskjeller. Hovedtrekkene i forskjellene kan summeres i uavhengighet mot leverandør versus uavhengighet i utviklingsspråk.

J2EE er uavhengig av leverandør, men baserer seg på utviklingsspråket Java. J2EE implementasjoner er tilgjengelig fra flere leverandører, hardware og operativsystemer. Selv etter implementasjon av forretningslogikk er det mulig å flytte til

J2EE fra en annen leverandør. J2EE applikasjoner er flyttbare mellom forskjellige leverandørers implementasjoner.

Windows DNA er uavhengig av utviklingsspråk, men er bundet til Microsoft Windows operativsystem og dertil støttede mikroprosessorer. Utviklere har stort utvalg i verktøy og utviklingsspråk til å implementere applikasjoner under Windows DNA. Windows DNA støtter utviklingsspråk som Java, C++, Visual Basic, Delphi og PowerBuilder. Applikasjoner må derimot kjøres på IIS, COM+, Distributed Transaction Coordinator (DTC) og OLE DB. Windows DNA server applikasjoner må kjøres på Windows NT eller Windows 2000. Nettleser-applikasjoner er knyttet til IE hvis de benytter ActiveX komponenter. Klienter som ikke bruker nettleser må implementeres på Win32 plattformer som Windows CE, 95, 98, NT eller 2000.

Tjeneste	Windows DNA	J2EE
Operativsystemer	Windows CE/95/98/NT/2000	Mange
Nettleser	Mange (best med IE)	Mange
Nettleser komponenter	ActiveX	Applet
Web server	IIS	Mange
Web server komponenter	ASP	Servlet og JSP
Applikasjonsserver	MTS og COM+	Mange EJB Applikasjonsservere
Server komponenter	MTS og COM+ komponenter	EJB
Kommunikasjon	DCOM	IIOP
Database aksess	ADO og OLE DB	JDBC og SQLJ
Transaksjoner	MTC	Mange via JTS
Sikkerhet	Microsoft OS	Java sikkerhet
Katalogtjenester	Microsoft registry, Active Directory	Mange via JNDI

Tabell 2: Sammenlikning mellom Windows DNA og J2EE

5.4.1 Kilder

The Business Benefits of EJB and J2EE™ Technologies Over COM+ and Windows DNA

Av Ed Roman og Rickard Øberg, ©1999 The Middleware Company.

http://java.sun.com/products/ejb/pdf/j2ee_dnabwp.pdf

The Technical Benefits of EJB and J2EE Technologies Over COM+ and Windows DNA

Av Ed Roman og Rickard Øberg, ©1999 The Middleware Company.

http://java.sun.com/products/ejb/pdf/j2ee_dnatwp.pdf

Comparing MTS and EJB by Anne Thomas, Patricia Seybold Group

Av Anne Thomas, ©1999 Copyright Distributed Computing

http://java.sun.com/products/ejb/pdf/9812MTSEJB.pdf

Enterprise JavaBeans Technology — Server Component Model for the Java™ Platform
Av Anne Thomas, Patricia Seybold Group, Revised December 1998
http://java.sun.com/products/ejb/pdf/white_paper.pdf

Comparing Microsoft Transaction Server to Enterprise JavaBeans
©2000 Microsoft Corporation
http://www.microsoft.com/com/wpaper/mts-ejb.asp

5.5 Oppdragsgiver

Objectnet er et selskap som primært driver utvikling av software. Konsulentene fra Objectnet jobber ute hos kunder på langtidskontrakter. Det er derfor viktig at konsulentene har flere måter å kople seg mot Objectnets administrative systemer. Konsulenter fra Objectnet utvikler enkelte av objectnets egne interne administrative systemer. Objectnet har idag følgende server applikasjoner i sitt test- og driftsmiljø. Web Logic Server, COM+, MS SQL Server, MySQL, Active Directory, Apache Web Server og Internet Information Server. Miljøet er delt i to grupperinger. Det ene miljøet er basert på Java teknologi og det andre er basert på Microsoft DNA teknologi.

5.6 Problemområde

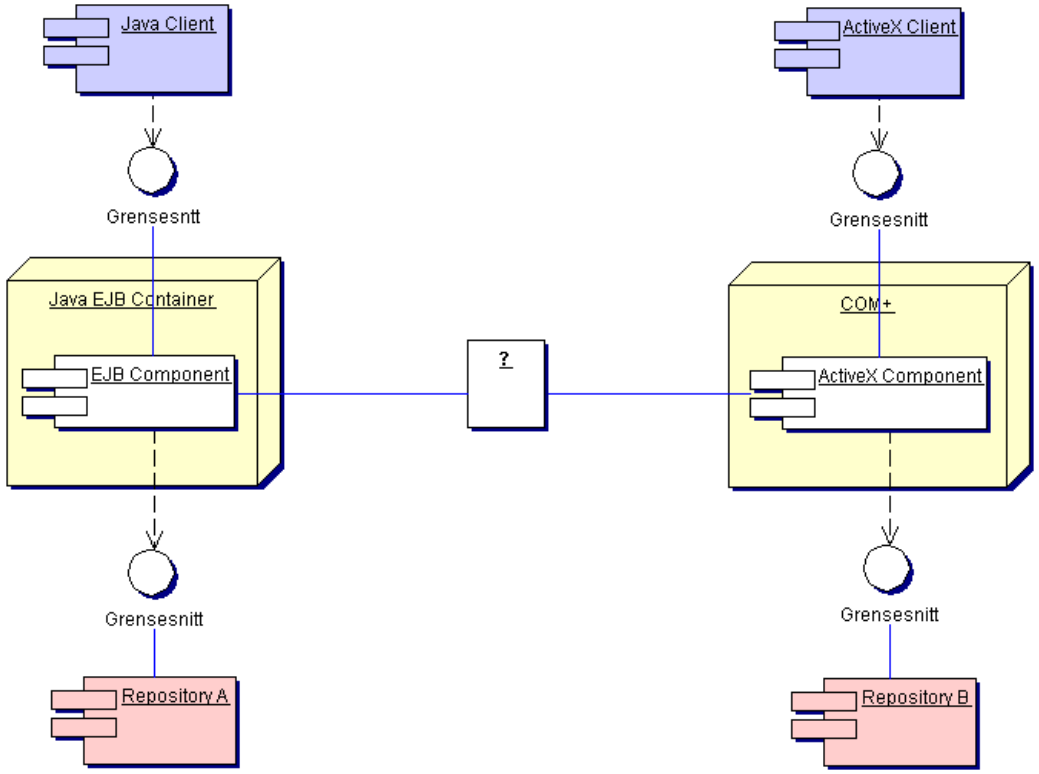
Det skal undersøkes hvordan applikasjonsserverene COM+ og Web Logic Server kan kommunisere i et serveroppsett, samt hvordan disse kan bruke felles repository for datalagring. Med repository i objectnets regi menes det registre over ansatte, kontakter, prosjekter og timeføring. Teknologier som er interessante er Java Enterprice Beans, COM+, SOAP, XML, JDBC, ODBC og JAVA2COM Bridge.

Objectnet ønsker å kartlegge måter å integrere administrative applikasjonsløsninger utviklet for henholdsvis Microsoft plattform og Java plattform i Objectnets test- og driftsmiljø. Det skal utvikles prototyper for integrasjonsmodell med fokus rundt registre med ansatte, kontakter og timer og disse skal vurderes mot hverandre ut fra driftssikkerhet, stabilitet og kompleksitet i implementasjon.

5.7 Avgrensninger

Det har vært nødvendig å gjøre avgrensninger i oppgaven, da kommunikasjon mellom J2EE og Windows DNA er et bredt område å forske på, og kan skje på flere plan. Fokus i oppgaven settes på hvordan kommunikasjon mellom J2EE og Windows DNA kan løses ved bruk av Enterprise Java Beans på J2EE siden og COM+ komponenter på Windows DNA siden, slik at man kan oppnå kommunikasjon mellom enterprise

komponenter på business laget. Dette er en interessant vinkling fordi det skjermer klientapplikasjoner fra å kjenne til hvordan komponentene i business laget er implementert. Det skal forskes på hvordan man kan eksponere en komponent, henholdsvis som EJB og COM komponenter, i begge miljøene gjennom å implementere server komponenten i det ene miljøet og en proxy komponent som eksponerer server komponenten i det andre miljøet.



Figur 4: Grafisk presentasjon av problemområde

Oppgaven vil i stor grad gå ut på å kartlegge og teste metoder for å oppnå kommunikasjon mellom EJB komponenter og COM+ komponenter samt å reflektere over disse i henhold til oppgaveteksten. Figur 4 illustrerer grafisk hvordan settingene i forskningen vinkles. Hovedområdet i forskningen identifiseres i Figur 4 med et ”?”.

Det har i samarbeid med veileder blitt gjort et tillegg til de teknologier som er nevnt i oppgaveteksten der CORBA og IIOP har kommet frem som teknologi det i tillegg skal legges vekt på i denne oppgaven.

6 Materiale og metoder

6.1 Materiale

Av relatert forskningsmateriale som har vært nyttig lesning i innledningsfasen til denne oppgaven kan spesiellt nevnes følgende arbeider:

A Detailed Comparison of
Enterprise JavaBeans (EJB) & The Microsoft Transaction Server (MTS) Models
Av Gopalan Suresh Raj
http://members.tripod.com/gsraj/misc/ejbmts/ejbmtscomp.html

A Detailed Comparison of CORBA, DCOM and Java/RMI
Av Gopalan Suresh Raj
http://www.execpc.com/~gopalan/misc/compare.html

Disse arbeidene har gitt meg en oversikt med fokus på håndtering av instanser gjennom en komponents livssyklus, “Database Connection Pooling”, arkitektur, support for distribuerte transaksjoner, typer av komponenter, portabilitet, interoperabilitet, håndtering av tilstander, håndtering av persistens, håndtering av ressurser, sikkerhet og implementasjon.

6.2 Metoder

Det ble i starten av prosjektet gjort research for å finne frem til overordnede metoder som kan lede til tilfredstillende løsninger i arbeidet med å implementere kommunikasjon mellom komponenter implementert henholdsvis som EJB eller COM+. Kriterier som var viktige var at metodene som ble valgt skulle ha forskjellige grader av åpenhet på forskjellige plan for å gi en god spennvidde i teknologi. Etter research ble det identifiert overordnede metoder som bridging, RPC/HTTP og CORBA/IIOP. Disse danner hovedområdene for eksperimenter og diskusjon. Hvert av hovedområdene har fått tildelt sitt eget hovedkapittel.

Forskjellig grad av eksperimentering er utført innen hvert hovedområde. Eksperimentene er med på å identifisere generelle problemstillinger til grunnlag for drøfting.

6.3 UML

For modellering av testscenarier og konseptuelle skisser for eksperimenter er det valgt å bruke Unified Modelling Language (UML). UML er en standard som egner seg til modellering i hele spekteret fra konseptuelle skisser til detaljert spesifisering av komponenter. Det finnes idag tre klasser av verktøy for å modellere med UML standarden. Den første klassen verktøy kan kun brukes til å modellere visuelle, grafiske skisser av systemer. Den andre klassen verktøy kan i tillegg generere et kodeskall i form av programkode i valgt utviklingsspråk ut fra UML-modellen. Den tredje typen verktøy kan i tillegg synkronisere tilbake i UML-modellen endringer som er gjort i kildekode utenfor verktøyet. Den siste klassen verktøy er i større grad knyttet til utviklingspråket enn den andre klassen.

6.4 Kilder

UML Resource Page
© Object Management Group (OMG)
http://www.omg.org/technology/uml/

7 Testmiljø

7.1 Verktøy

7.1.1 Together 4.1

Til modellering av grafiske fremstillinger i UML er det valgt å bruke Together 4.1. Together er et verktøy som kan generere og synkronisere kildekode for Java, C++ og CORBA IDL i tilknytning til UML-modeller. Together er brukt til å lage alle UML diagrammer. Together er også brukt som IDE til koding og testing av Java-klienter og J2EE EJB komponenter. Vedlegg I.A viser hvordan grensesnittet i Together 4.1 arter seg på Windows plattform. Together 4.1 er utviklet fullstendig i Java og er avhengig av Java SDK 1.2 eller nyere. Hvis Java SDK ikke er installert på arbeidsstasjonen fra før, så vi installasjonsprogrammet installere denne sammen med Together.

Objectnet as har tegnet partnetavtale med Togethersoft som utvikler Together. Til internt bruk og opplæring er da Together fritt tilgjenelig. Det er derfor et naturlig valg å dra nytte av dette verktøyet i denne oppgaven.

7.1.2 Visual Studio 6.0

For å utvikle COM+ testkomponenter er det brukt Visual Studio 6.0. Visual Studio er Microsofts integrerte utviklingspakke som innkapsulerer alle de utviklingsverktøy som er av vesentlig visuell karakter. Blant disse er Visual Basic, Visual C++ og Visual Interdev. Visual Basic er brukt i denne oppgaven til å utvikle COM+ objekter fordi det er et språk med enkel syntaks og det skjermer komplekse strukturer som utviklere av komponenter til applikasjonsservere ofte ikke skal trenge å forholde seg til. Visual Interdev er brukt i forbindelse med testing av komponenter som kommuniserer med SOAP som omtalt i kapittel 9. Vedlegg I.B viser Visual Studio IDE grensesnitt. Objectnet as har tegnet en partneravtale (MCSP) med Microsoft og har derfor tilgang til lisenser for intern testing og utvikling med Visual Studio. Det er da naturlig å velge Visual Studio som et verktøy til denne oppgaven.

7.1.3 Microsoft SQL Server 2000 Developer Edition

For å simulere et repository er det brukt Microsoft SQL Server 2000 Developer Edition. MS SQL Server er Microsofts satsingsområde på arenaen for stordatabaser. MS SQL Server skalerer fra arbeidsstasjon med en bruker til flerprosessorsystemer med

tusener av samtidige oppkoplinger og er laget kun med støtte for Win32 plattformen. Objectnet as har gjennom partneravtale med Microsoft tilgang til lisenser for intern testing og utvikling mot Microsoft SQL Server 2000 plattformen.

7.1.4 Windows 2000 Professional

All testing for denne oppgaven gjøres på Microsoft Windows 2000 Professional plattform. De andre verktøyene fungerer tilfredsstillende over denne plattformen. Denne plattformen gir i tillegg tilgang til COM+ tjenesten som kreves for å eksekvere COM+ komponenter.

7.1.5 WebLogic Server 5.1

WebLogic Server 5.1 fra BEA Systems er brukt som J2EE plattform for EJB komponenter og JDBC connection pooling mot repository. Objectnet as har gjennom partneravtale med BEA Systems tilgang til WebLogic Server for testing og internt bruk.

7.1.6 Apache SOAP

Apache SOAP er brukt som transport for SOAP meldinger fra J2EE til COM+. Dette er et gratis produkt som vedlikeholdes og lisensieres av The Apache Software Foundation. Apache SOAP utvikles i Java.

7.1.7 Microsoft SDK for Java 4.0

Microsoft SDK for Java 4.0 er det som er igjen av Microsofts satsing på Java etter rettsaken med Sun Microsystems, Inc. Microsoft utgir gjevnlig oppdateringer. SDK'en inneholder Microsofts JVM samt implementasjon av JDK med eksempler. Dette er en gratis SDK.

7.1.8 TAO

TAO er en OpenSource C++ ORB som implementerer de fleste egenskaper og komponenter i CORBA 2.4 spesifikasjonen. TAO kan lastes ned fra Internett uten utvikler- eller kjøretidslisensiering. TAO bygger på ACE (ADAPTIVE Communication Environment™) som er en objekt orientert verktøypakke for å utvikle programkomponenter mot nettverk.

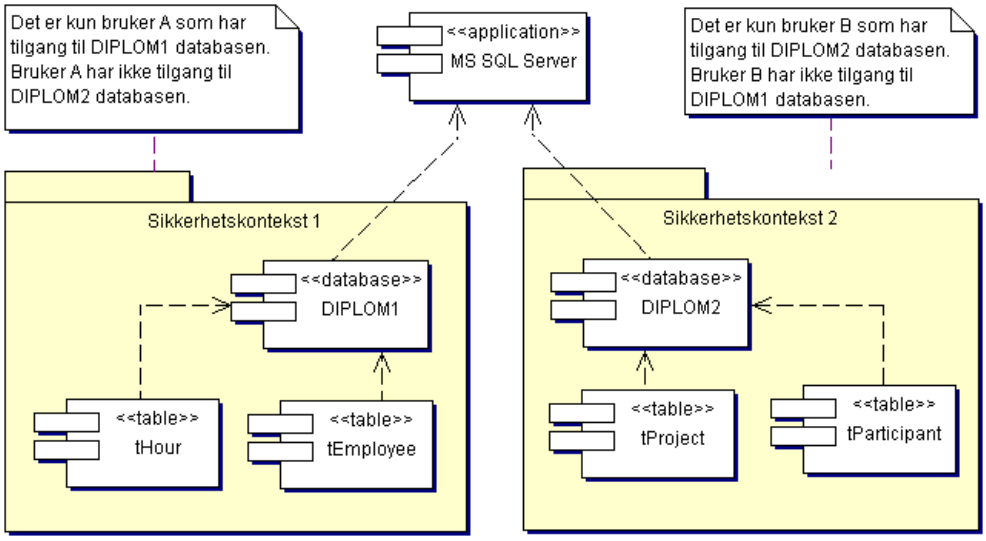
7.2 Oppsett av software

Testmiljøene er satt opp på en enkel maskin med konfigurasjon Intel 500 MHz prosessor, 128MB RAM, 6 GB harddisk. Denne maskinen har Windows 2000 Preofessional edition som operativsystem og er base for all testing som er gjort i denne oppgaven. På testmaskinen er det installert WebLogic Server 5.1, Sun JDK 1.3, Microsoft SQL Server 2000 Developer Edition, Microsoft Visual Interdev 6.0, Microsoft Visual Basic 6.0, Java-COM Bridge (JACOB), Microsoft SDK for Java 4.0, Microsoft SOAP Toolkit 2.0, Internet Information Server 5.0, Together Control Station 4.1 og Apache SOAP. Det er ikke gjort endringer i forhold til default installasjon for noen av produktene med unntak av Microsoft SDK for Java som ble installert under katalogen MSSDK4. Dette fordi oppsett av miljøvariable i samspill med WebLogic Server 5.1 ikke fungerte med default installasjon. WebLogic Server 5.1 er konfigurert til å bruke Java 2 Platform Standard Editition 1.3 og Java 2 Platform Enterprise Edition 1.2.1 som kjøremiljø. I tillegg er det konfigurert inn i oppstartsfilen til WebLogic Server 5.1 referanser til Apache SOAP og JACOB etter behov for testingen. Together Control Station 4.1 er konfigurert til å bruke Java 2 Platform Standard Edition 1.3.

7.3 Repository

For å simulere to repository som er adskilt i forhold til sikkerhetskontekst så er det satt opp to databaser på Microsoft SQL Server 2000 Developer Edition. Databasene er navngitt DIPLOM1 og DIPLOM2 som vist i Figur 5. DIPLOM1 fungerer som repository for J2EE-miljøet mens DIPLOM2 fungerer som repository for COM+ miljøet. Hver base har sin respektive bruker. diplom1 brukeren har kun tilgang til DIPLOM1 databasen, mens diplom2 brukeren kun har tilgang til DIPLOM2 databasen. J2EE-miljøet skal kun få tilgang til repository gjennom diplom1 brukeren mens COM+ miljøet kun skal få tilgang til repository gjennom diplom2 brukeren.

Databasene har tabeller som til sammen simulerer et enkelt system for å kontrollere timeforbruk fra ansatte mot prosjekter. DIPLOM1 databasen har tabellene tHour og tEmployee. DIPLOM2 databasen har tabellene tProject og tParticipant. På denne måten får en klientapplikasjon som skal tilby et system for å registrere timer ha indirekte tilgang til repository gjennom begge miljøer samtidig. SQL skript for å lage tabeller i databasene er listet i Vedlegg VI.



Figur 5: Repository for testmiljøet

7.4 Grunnleggende Java miljø

Grunnkomponentene i J2EE testmiljøet er implementert som Java Enterprise Beans. Figur 6 viser grafisk modell av hvordan Employee-grensesnittet gjøres tilgjengelig fra WebLogic Server 5.1 som en Java Enterprise Bean. tEmployee tabellen i repository gjøres tilgjengelig mot omverdenen via Microsofts OLEDB grensesnitt for Microsoft SQL Server 2000. WebLogic settes opp med en pool av databasekoplinger som holdes åpne mot repository gjennom OLEDB. Det er i *weblogic.properties* konfigurasjonsfila konfigurert en databasekopling mot Microsoft SQL Server 2000 som vist i Tabell 3.

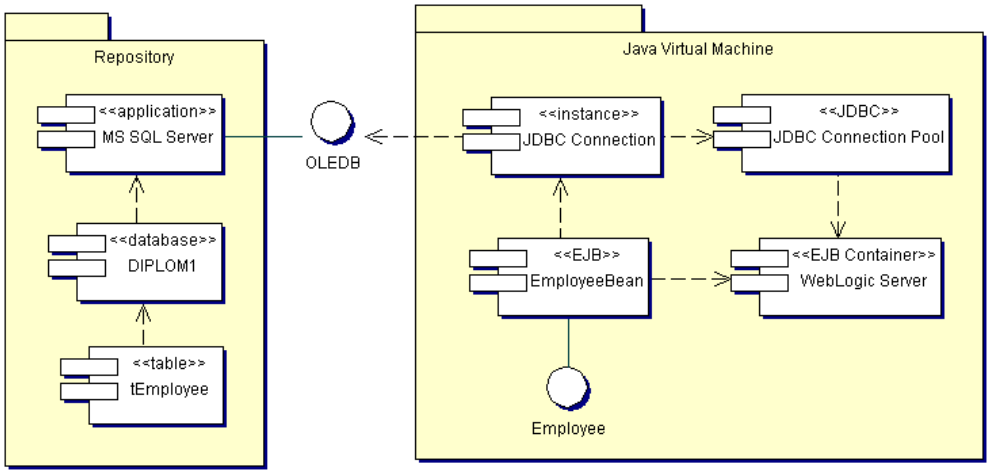
```
# ----------------------------------------------------------------
# Connection for DIPLOM1
# ----------------------------------------------------------------
 weblogic.jdbc.DataSource.diplom1DataSource=diplom1Pool

 weblogic.jdbc.connectionPool.diplom1Pool=\
     url=jdbc:weblogic:mssqlserver4:localhost:1433,\
     driver=weblogic.jdbc.mssqlserver4.Driver,\
     loginDelaySecs=1,\
     initialCapacity=1,\
     maxCapacity=1,\
     capacityIncrement=1,\
     allowShrinking=true,\
     shrinkPeriodMins=15,\
     refreshTestMinutes=1,\
     testTable=tEmployee,\
     props=user=diplom1;password=diplom1;server=localhost;\

# Add a TXDataSource for the connection pool:
weblogic.jdbc.TXDataSource.weblogic.jdbc.jts.diplom1Pool=diplom1Pool
#
# Add an ACL for the connection pool:
weblogic.allow.reserve.weblogic.jdbc.connectionPool.diplom1Pool=everyone
```

Tabell 3: WebLogic konfigurasjon for tilgang mot repository

Grensesnittet Employee mot EmployeeBean er tilgjengelig direkte for applikasjoner i J2EE-miljø og for miljøer utenfor J2EE gjennom IIOP. Gangen i bruk av EJB komponenten er som følger. EmployeeBean spør WebLogic Server om å få bruke en databaseoppkopling fra poolen, utfører oppdrag fra klientapplikasjoner gjennom denne og frigjør den tilbake til WebLogic Server.



Figur 6: Grunnleggende J2EE testmiljø konsept (Employee EJB)

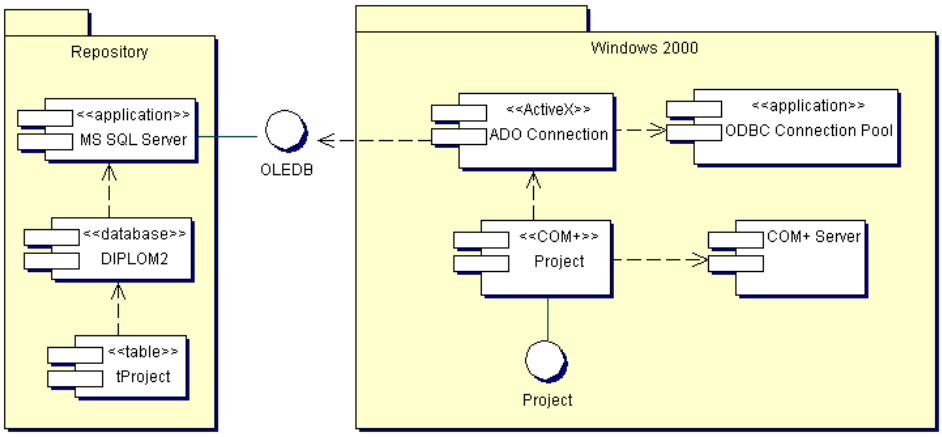
EmployeeBean og HourBean er pakket i jar-fila ejb_diplom.jar. WebLogic er via konfigurasjonen vist i Tabell 4 satt opp til å laste Home-grensesnittene til disse ved oppstart. Programkode for Employee og Hour komponentene samt byggeskript er listet i Vedlegg III.

weblogic.ejb.deploy=C:/weblogic/myserver/ejb_diplom.jar

Tabell 4: WebLogic konfigurasjon for grunnleggende EJB komponenter

7.5 Grunnleggende COM miljø

Det ble ikke utviklet en COM+ Participant komponent da denne ble overflødig i forhold til å gjennomføre testing. Grunnkomponenten Project i COM+ miljøet er implementert som en ActiveX komponent som igjen er registrert mot COM+ serveren og eksponeres derfra som en COM+ komponent. Tabell 5 viser grafisk modell av hvordan Project-grensesnittet gjøres tilgjengelig fra COM+ serveren som en COM+ komponent. tProject tabellen i repository gjøres tilgjengelig mot omverdenen via Microsofts OLEDB grensesnitt for Microsoft SQL Server 2000. Windows 2000 settes opp med en Open DataBase Connector (ODBC) som navngis DIPLOM2. Denne påner for en datakanal mot Microsoft SQL Server 2000. COM+ Project komponenten bruker Active Data Object (ADO) ActiveX komponent som kanal for å snakke SQL mot repository.



Tabell 5: Grunnleggende COM+ testmiljø konsept (COM+ Project)

Programkode for implementasjon av COM+ Project komponenten er vist i Vedlegg IV.

7.6 Kilder

Together Web Site

© TogetherSoft

http://www.togethersoft.com/

Visual Studio Web Site

© Microsoft Corporation

http://msdn.microsoft.com/vstudio/

Microsoft SQL Server Web Site

© Microsoft Corporation

http://www.microsoft.com/sql/

Microsoft Windows 2000 Professional

© Microsoft Corporation, Inc.

http://www.microsoft.com/windows2000/guide/professional/overview/

BEA WebLogic Server 5.1 Documentation Center Web Site

© BEA Systems, Inc.

http://www.weblogic.com/docs51/resources.html

The Apache Software Foundation Web Site

© Apache Software Foundation

http://www.apache.org/

Microsoft Technologies for Java, Web Site

© Microsoft Corporation, Inc.

http://www.microsoft.com/java/

Realtime CORBA with TAO™ (The ACE™ ORB)

By Douglas C. Schmidt

http://www.cs.wustl.edu/~schmidt/TAO.html

The ADAPTIVE Communication Environment (ACE™)

By Douglas C. Schmidt

http://www.cs.wustl.edu/~schmidt/ACE.html

8 Bridging

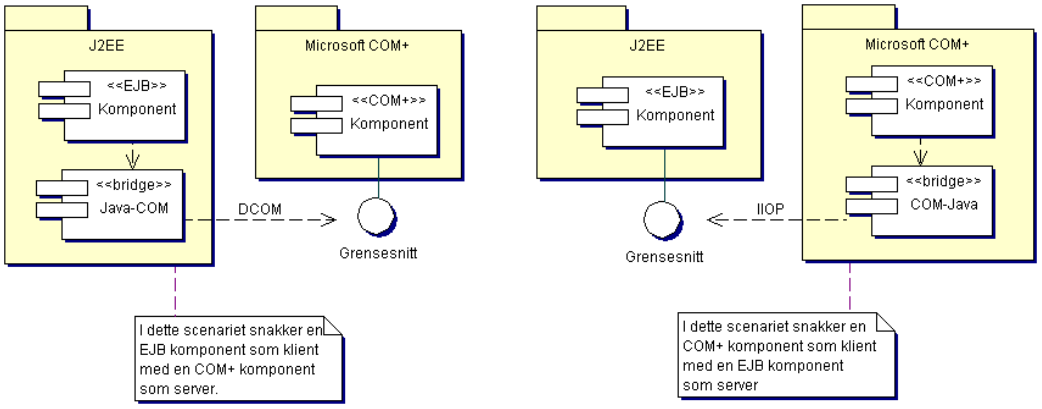
8.1 Innledning

For å dekke behovet for at enterprise systemer med utspring i forskjellig teknologi kan samkjøres så er en metode å anvende en softwarebridge. En softwarebridge kan implementeres for å kople sammen bestemte objekter mellom EIS systemer og dermed være låst til disse, eller den kan være av mere generalisert art slik at den kopler sammen EIS systemene på et lavere nivå. Den første metoden kan gi utviklere av EIS komponenter kontrollen på alle lag i implementasjonen, mens den siste metoden kan frita utviklere for å implementere nærmere transportlaget.

Softwarebridgen implementeres oftest i tilknytning til klientsoftwaren eller serversoftwaren. Det er ikke så vanlig å implementere den som en frittstående, mellomliggende proxy. Dog skal det nevnes at CORBA 3.0 spesifikasjonen beskriver ”CORBA/DCOM Interoperability” som er en generalisert, mellomliggende bridge mellom CORBA og DCOM. Det finnes mange kommersielle softwarebridger å få kjøpt for å integrere Java og COM.

8.2 Bridging mellom J2EE og COM+.

I kontekst av oppgaven så er det to hovedsaklige metoder for bridging som er interessante. Den ene metoden er å plassere bridgen på Java-siden, mens den andre metoden er å plassere bridgen på COM+ siden (se Figur 7).



Figur 7: Konsept for bridging mellom COM+ og J2EE.

Med bridge på Java-siden kan EJB komponenter som klienter bruke COM+ komponenter som servere. Kommunikasjon skjer da over DCOM protokoll og via Java

Native Interface (JNI). Hvis bridgen plasseres på COM+ siden så kan COM+ komponenter som klienter bruke Java-komponenter som servere. Kommunikasjonen skjer da over IIOP protokoll.

En rekke kommersielle og gratis bridger er tilgjengelige fra tredjeparts leverandører. De bridger som har vært interessante for denne oppgaven er blant følgende; J-Integra Java-COM Bridge, The JACOB Project, Weblogic COM, IBM Bridge2Java, jacoZoom og Microsoft SDK for Java 4.0.

Til implementasjon i testscenarier er det valgt ut to bridger. Microsoft SDK for Java 4.0 brukes som bridge på COM+ siden, mens Java-COM Bridge (JACOB) brukes som bridge på Java-siden. Bridgene til testing er valgt ut fra kriteriet at de er gratis tilgjengelig.

8.3 Valg til eksperiment

8.3.1 Microsoft SDK for Java 4.0

8.3.1.1 Introduksjon

Microsoft SDK for Java står for Microsoft Software Development Kit for Java. Denne kan lastes ned fra Microsofts nettsted *www.microsoft.com/java*. SDK'en inneholder en samling med Java-verktøy for å kompilere kildekode og eksekvere kommandoer. Det følger med eksempler som demonstrerer bruk av Java med andre teknologier, og java pakker, som er samlinger av klasser som tilhører samme "namespace", samt dokumentasjon av de viktigste egenskaper SDK'en tilbyr.

Microsoft SDK for Java er ikke et verktøy for å utvikle grafiske grensesnitt med menyer, knapper, verktøylinjer og liknende. De fleste verktøy i SDK'en kjører fra MS-DOS kommando grensesnitt. Enkelte verktøy er applikasjoner. Det antas at utviklere som skal benytte Microsoft SDK for Java har kunnskap om hvordan kalle opp kommandolinje kompilatorer og og tilsvarende andre verktøy. Utvikling og bruk av Javaskript er ikke dekket av Microsoft SDK for Java.

8.3.1.2 Java objekter eksponert som COM komponenter

For å eksponere et Java objekt som en COM component så kreves det at Java objektet innkapsuleres i en COM-kompatibel wrapper. Microsoft VM gjør dette ved å bruke en COM-kallbar wrapper (CCW). Den presenterer Java objektet nøytralt til alle

andre COM-komponenter og utviklingsspråk som kan benytte COM. Microsoft VM genererer dynamisk en CCW for Java-objektet når det kreves at det eksponeres som et COM objekt. CCW er essensiellt et generisk COM objekt som har sin virtuelle tabell (vtable) over pekere til funksjoner dynamisk generert for å supportere standard COM grensesnitt samt alle COM grensesnitt som er implementert i Java-objektet. Standard COM-grensesnitt som eksporteres for alle Java/COM objekter av CCW tillater Java-objekter å bli brukt som funksjonelle COM-objekter. Et Java objekt som f.eks. registreres som Control vil bli eksportert som en ActiveX komponent uten ekstra utvikling fra uviklerens side. CCW har egenskap av å kunne bli aggregert av et annet COM-objekt, men kan ikke selv aggregere andre COM-objekter. Microsoft VM lagrer pekeren til det ytre COM-objektets IUnknown grensesnitt i CCW.

8.3.1.3 Avhengigheter

Microsoft SDK for Java er i skrivende stund i versjon 4.0. Denne SDK'en har utviklere på Windows 2000 plattform som målgruppe, men den er bakoverkompatibel med Microsoft Windows 95, Microsoft Windows 98, Microsoft Windows Millennium Edition (Me) og Microsoft Windows NT 4.0 utviklingsmiljøer.

8.3.1.4 Status

JDK 1.1 supporteres, med unntak fra RMI. RMI støtte kan lastes ned gratis fra *microsoft.com*.

Informasjon om fikser og oppdateringer av Microsoft VM etter releasen av SDK'en for Java 4.0 vil finnes på Microsofts java-site *www.microsoft.com/java*.

Informasjon om Microsoft VM Y2K støtte finnes på Microsoft Internett site *http://www.microsoft.com/technet/year2k/produkt/produkt.asp* under gruppen Technologi.

8.3.2 Java-COM Bridge (JACOB)

8.3.2.1 Introduksjon

I enkelte situasjoner kan det være hensiktsmessig å kalle COM objekter fra et Java-object uten å være bundet til en spesiell VM. JACOB er utviklet initiellt som et privat prosjekt av Dan Adler for å dekke dette behovet. Grunnen til dette er at Java 2 Plattform øker i popularitet samtidig med at Microsofts satsing på Java er noe usikker. Tidligere har det vært nødvendig for utviklere å bruke Microsoft VM for å lett få tilgang

til slik funksjonalitet. JACOB er uavhengig av hvilken VM som kjøres men er bundet til Microsoft Win32 plattform.

Intensjonen bak JACOB er at den skal være en kompatibel implementasjon til Microsoft Java SDK. Dette er enda ikke helt i mål. Porting av kode som kaller COM objekter fra Microsoft Java SDK til JACOB skal etter intensjonen være så enkelt som å bytte import direktiv i Java-kode fra *com.ms.com* til *com.jacob.com*. Det er derfor ikke laget en egen dokumentasjon for JACOB. Man skal kunne bruke dokumentasjonen til Microsoft Java SDK direkte men dog med enkelte unntak. JACOB implementerer enkelte av *com.ms.com* grensesnitt, som Variant, Dispatch og SafeArrey ved å bruke Java Native Interface (JNI) teknologi.

8.3.2.2 Distribusjon

JACOB binær distribusjon inkluderer:

jacob.jar	En jar fil med Java-klasser som må legges til CLASSPATH. Denne pakken erstatter *com.ms* med *com.jacob* (f.eks. *com.ms.com.Variant* erstattes med *com.jacob.com.Variant*.)
jacob.dll	En liten relativt liten Win32 DLL som må legges til PATH.
Eksempler	Leveres i Java kildekode og kompilert form for å demonstrere egenskaper i JACOB. Et sett med wrapper klasser for Microsoft ADO følger som eksempel.

Kildekode er tilgjengelig i JACOB kildekode distribusjonen, som også ikluderer Java og C++ kode.

8.3.2.3 Implementasjon

JACOB implementasjonen av en COM Automasjon kontroller er grunnleggende anderledes enn implementasjonen til Microsoft JVM. I JACOB er *com.jacob.com.Dispatch* den grunnleggende byggesteinen. Denne klassen brukes til å opprette en instans av en Automasjonsserver. *com.jacob.activeX.ActiveXComponent* klassen utvider Dispatch grensesnittet for å sikre kompabilitet med måten Automasjonsservere opprettes på i MS JVM. Alle metodene i Dispatch-klassen er statiske for å sikre kompatibilitet med Microsofts Dispatch.

Konstruktøren for *com.jacob.com.Dispatch* tar en *String* som parameter. Hvis strengen inneholder et kolon (':') så oversettes den som en Moniker, ellers oversettes den

som en ProgID. Dette betyr at hvis man ønsker å opprette en instans ved hjelp av CLSID istedenfor ProgID, så må man bruke *clsid:* Moniker som parameter. På denne måten unngås nødvendigheten for å implementere klasser for å håndtere GUID i Java.

Implementasjonen av SafeArray i JACOB gjenspeiler det meste av funksjonaliteten som finnes i Microsoft implementasjonen. Dette innebærer bl.a. at man må sette størrelse på arrayet i konstruktøren før det kan fylles med data.

8.3.2.4 Begrensninger og avvik fra MS JVM

- JACOB supporterer for øyeblikket ikke custom grensesnitt.
- JACOB har ikke egenskaper av å kunne generere kildekode, som f.eks. JACTIVEX fra Microsoft implementasjonen har. Derfor er det ikke gjort forsøk på å mappe COM-grensesnitt direkte til ekte Java-grensesnitt.
- I denne versjonen initialiseres COM via CoInitializeEx() med parameter CO_INITMULITTHREADED en gang per tråd når den første instansen av Dispatch i en tråd.
- Event modellen i denne utgaven er ikke kompatibel med hverken Beans modellen eller WFC modellen. Nåværende modell tillater å levere Java objekter som en COM event lytter, og metodene til Java objektet kalles ved refleksjon.
- Denne versjonen inneholder ikke en omvendt bridge. Derfor kan man ikke bruke Java objekter som parametre til COM objekters metoder. Alle parametre må pakkes i Variant objekter. Å utvikle en omvendt bridge er ikke prioritert da Sun tilbyr en slik som en komponent i Java PLUG-IN.
- Denne versjonen kan ikke innkapsulere grafiske ActiveX kontroller inn i Java objekter.

8.3.2.5 Stabilitet

- Plattform og VM testing er kun utført mot NT4SP5 med Sun’s JDK 1.1.6 og JDK 1.2.2 og Microsoft’s Java VM (Build 3182).
- Det er mulige problemer med lekasje av minne både på COM siden og i JNI siden.

8.3.2.6 Levetiden til COM Objekter

Levetiden til JACOB objekter korresponderer med levetiden til Java wrapper objektet. IUnknown::Release() kalles i finalizer på Java wrapper objektet. Siden COM-objektets pekere holdes som Java int's, så har ikke den underliggende VM kunnskap om denne sammenhengen. Hvis man ønsker å frigi COM-objekter tidligere, så er det best å forårsake Garbage Collection (GC) for å samle opp Java wrapper objektet.

8.3.2.7 Avhengigheter

JACOB vil kun fungere i JVM på Microsoft Win32 plattform.

8.3.2.8 Status

JACOB kjøres nå som et Open Source proskjekt. Det er derfor fritt å melde seg som deltaker til utvikling av delkomponenter. I skrivende stund er JACOB i versjon 1.6. Denne ble sluppet 9. desember 2000. Det er usikkert om og når en ny versjon vil slippes.

8.4 Gjennomføring av eksperiment

8.4.1 COM mot Java – Microsoft SDK for Java 4.0

8.4.1.1 Innledning og hypotese

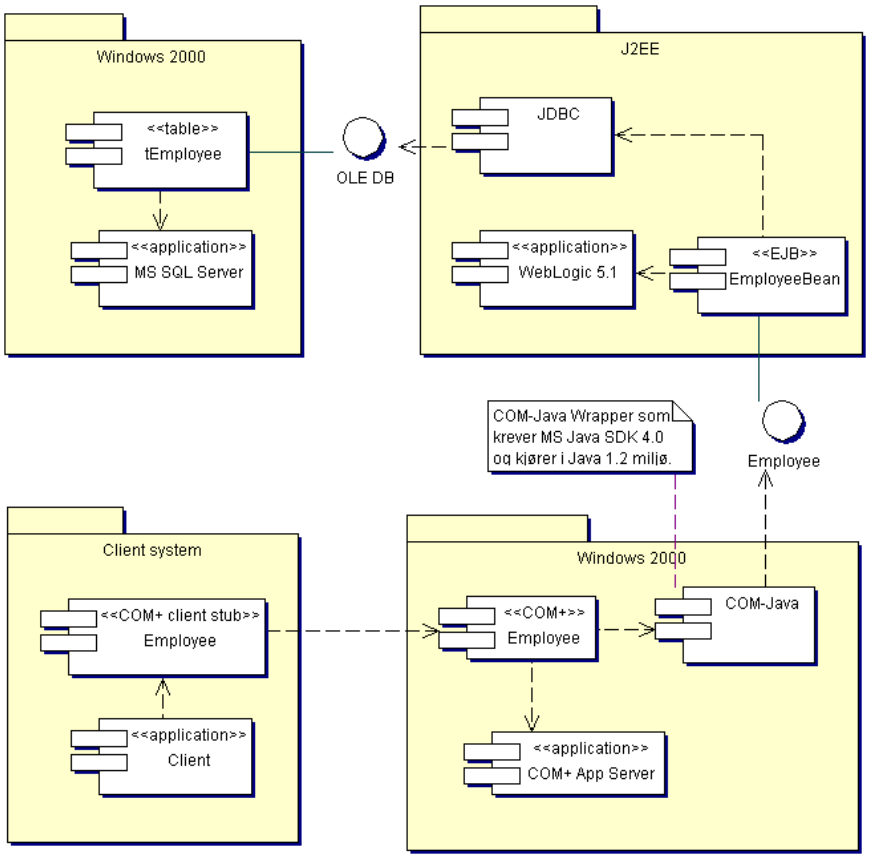
En mulig metode for å integrere J2EE og Windows DNA på business-laget kan være å benytte Microsoft SDK for Java. Microsoft SDK for Java vil da fungere som en bridge for kommunikasjon fra Windows DNA miljø mot J2EE miljø.

I denne delen skal det testes om det kan løses at Microsoft DNA (COM+) kan snakke med J2EE EJB serveren (WebLogic) på businesslaget ved hjelp av Microsoft SDK for Java. I Figur 8 vises en strukturell skisse over en mulig vei å gå for å nå en EJB komponent fra en COM+ komponent via Microsoft SDK for Java. COM+ Serveren skal tilby EmployeeEJB komponenten indirekte gjennom COM+ Employee komponenten som er en proxy for EJB komponenten. Klienter som kopler seg mot COM+ Serveren Employee komponent skal ikke trenge å supportere Java direkte, og skal tro at Employee komponenten er en ren COM+ komponent. Microsoft SDK for Java skal brukes på laget der meldingsutveksling mellom enterprise komponentene, henholdsvis EJB komponent som server og COM+ som proxy foregår. COM+ proxy komponenten skal benytte Microsoft SDK for Java for å snakke med EJB komponenten.

8.4.1.2 Eksperimentell prosedyre

8.4.1.2.1 Oversikt

For å teste Microsoft SDK for Java som en mulig bridge på COM-siden mellom J2EE og Microsoft COM+ så er det utviklet og konfigurert et testmiljø som vist i Figur 8. Den øverste halvdelen av figuren er utdrag fra det initielle scenariet for EJB+ miljøet som beskrevet i kapittel 7.4. Grensesnittet Employee tilbys som en EJB komponent i Java-miljøet. Nedre halvdel av figuren viser hvilke komponenter som utvikles og benyttes spesiellt for testen. Microsoft SDK for Java installeres og konfigureres. En COM+ komponent som tilbyr COM+ grensesnittet Employee i Microsoft-miljøet implementeres slik at den bruker Microsoft SDK for Java som proxy for kommunikasjon mot EmployeeEJB komponenten. COM+ Employee komponenten vil for testklienten fremstå som en ren COM+ komponent, men vil implementeres som en proxy mot EmployeEJB komponenten.



Figur 8: COM-Java via Microsoft SDK for Java eksperimentell prosedyre

8.4.1.2.2 Oppsett av WebLogic Server

J2EE Serveren skal kjøre under Sun JDK og kan derfor startes på vanlig måte, mens klientklassene skal kjøre i kontekts av Microsoft SDK for Java 4.0. Det er derfor laget et eget skript (se Vedlegg II.A) for å opprette de korrekte miljøvariablene som trengs for at Microsoft SDK for Java skal fungere som miljø for klienten.

8.4.1.2.3 Konfigurere Microsoft SDK for Java

Klientklassene må registreres som trusted for Microsoft SDK for Java for at de skal kunne eksponeres til COM miljøet. Dette gjøres fra kommandolinje som vist i Tabell 6. Kommandoene kjøres i kontekst av skriptet setEnvMS.cmd (se Vedlegg II.A) som setter miljøvariablene for Microsoft SDK for Java. EJB jar fila må legges til Java CLASSPATH. Deretter kjøres jview weblogic.com.Export for å gjøre alle klassene i CLASSPATH trusted for Microsoft SDK for Java bridgen. Dette gjør at bridgen får rettigheter til å eksponere disse mot Microsoft DNA gjennom COM.

```
$ setEnvMS.cmd
$ clspack –auto
$ set CLASSPATH=%CLASSPATH%;weblogic\myserver\ejb_diplom.jar
$ jview weblogic.com.Export %CLASSPATH%
```

Tabell 6: Eksponere Java klient klasser som trusted for Microsoft SDK for Java

8.4.1.2.4 Lage COM+ proxy komponent

Lage COM+ proxy komponent. Må referere til MS JDK programbiblioteker som trengs for å mappe opp COM proxy J2EE komponenten.

COM+ Employee proxy komponenten er laget som en ActiveX DLL ved hjelp av Visual Basic 6.0. Det er opprettet et prosjekt av type ActiveX DLL som er navngitt DiplomEmployee. En klasse som er navngitt Employee er implementert i dette prosjektet med programkode som vist i Tabell 7. Det lages en DLL av Visual Basic prosjektet. Denne DLL'en registreres i COM+ Applikasjonsserveren som en COM+ komponent. COM+ Employee er nå eksponert i Windows DNA til alle utviklingsspråk og applikasjoner som kan benytte COM komponenter.

```
Dim ctx As Object
Dim broker As Object
Dim Employee As Object

Private Sub Connect()
   Set WebLogic = GetObject("java:weblogic.com.Tengah")
   Set ctx = WebLogic.getInitialContext("t3://localhost:7001")
   Set broker = ctx.lookup("diplom.EmployeeHome")
End Sub

Public Sub LookupEmployee(sEmpNo As String, sEmpName As String)
```

```
    Connect

    On Error GoTo lookupfailed
    Set Employee = broker.findByPrimaryKey(sEmpNo)
    GoTo success

lookupfailed:
    On Error GoTo createfailed
    Set Employee = broker.Create(sEmpNo, sEmpName)
    GoTo success

createfailed:
    Exit Sub

success:

End Sub

Public Property Get FullName() As Variant
  FullName = Employee.getName
End Property
```

Tabell 7: Programkode for COM+ Employee proxy komponent

For å kople seg mot COM+ Employee komponenten så bruker man i Microsoft DNA miljø, vist for Visual Basic for Applications (VBA), metodikk som vist i Tabell 8.

```
Set Emp = CreateObject("DiplomEmployee.Employee")
Emp.LookupEmployee "A11", "Odd Oddsen"
```

Tabell 8: Kode for å opprette instans av COM+ Employee med VBA

8.4.1.2.5 Lage testklient

Til denne testen er det laget en testklient i Visual Basic 6.0 for å teste om oppkopling er mulig. Testklienten kjører 3 runder med 10000 kall mot testklienten og bruker metoder slik at COM+ Employee proxy komponenten må kontakte EmployeEJB for at den skal gjøre jobben. På denne måten testes stabilitet samtidig som man får en timing av hastigheten i den teknologiske løsningen.

Testklienten er implementert som en applikasjon som refererer til og bruker COM+ Employee komponenten via COM+ serveren. Kildekode for klientapplikasjonen er vist i Tabell 9.

```
Dim Emp As DiplomEmployee.Employee

Private Sub Command1_Click()
  Set Emp = New DiplomEmployee.Employee
  Dim sName As String

  Emp.LookupEmployee "A11", "Odd Oddsen"

  ' Test 1
  StartTime = Time

  For i = 1 To 10000
   sName = Emp.FullName
  Next

  EndTime = Time

  Label1.Caption = Abs(DateDiff("s", StartTime, EndTime)) & " sek"
```

```
' Test 2
StartTime = Time

For i = 1 To 10000
 sName = Emp.FullName
Next

EndTime = Time

Label2.Caption = Abs(DateDiff("s", StartTime, EndTime)) & " sek"

' Test 3
StartTime = Time

For i = 1 To 10000
 sName = Emp.FullName
Next

EndTime = Time

Label3.Caption = Abs(DateDiff("s", StartTime, EndTime)) & " sek"

End Sub
```

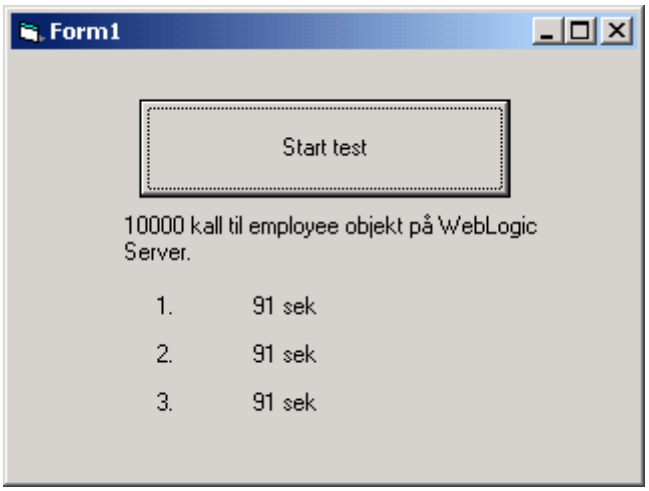
Tabell 9: Klient applikasjon mot COM+ Employee proxy

8.4.1.3 Resultater

Implementasjonen gikk uten de helt store problemer. Det var litt problematisk å få Microsoft SDK for Java 4.0 til å aksepteres som oppstartsmiljø for COM+ Employee proxy komponenten. Det viste seg at det var problem med mellomrom i navn på stien som SDK'en installeres på som default. Da Microsoft SDK for Java 4.0 ble installert med en sti som ikke inneholdt mellomromstegn så gikk registreringen av miljøvariabler og trusted EJB Java klient klasser uten problemer.

Det var også litt problemer med at *java.lang.Long* klassen ikke var trusted av ukjent grunn. Det var derfor i testen nødvendig å skrive opp EmployeeEJB komponenten så den brukte *java.lang.String* som parameter i metoder der *java.lang.Long* tidligere ble brukt. Da fungerte alt etter hypotesen.

Klientapplikasjonen, vist i Figur 9, kjørte 3 serier med 10000 kall mot COM+ Employee komponenten. Denne ble kjørt flere gange for å se etter svakheter i teknologien. Det ble ikke påvist noen svakheter hverken når en klient kjørte alene eller da flere klienter kjørte samtidig. Timing vises i Figur 9.



Figur 9: Grensesnitt for klient applikasjon mot COM+ Employee proxy

8.4.1.4 Drøfting

Etter at utviklingen av dette testmiljøet der Microsoft SDK for Java 4.0 er brukt som bridge mellom J2EE og Microsoft DNA så sitter man igjen med intrykket av at dette er en praktisk og ryddig måte å løse kommunikasjon i retningen J2EE mot Windows DNA. Det er imidlertid endel betraktninger som er viktige å gjøre.

Hver gang det gjøres en revisjon av EJB komponenten så må den registreres for Microsoft SDK for Java bridgen. Dette gjør at det nok kan være en fordel å utvikle komponentene i EJB miljøet til en ferdig revisjon for så å implementere COM+ proxy komponenter mot disse. Det sees ut fra testingen at COM+ proxy komponenten ikke trenger å implementeres som en fullt funksjonabel transparent kanal mot EJB komponenten. Det er dermed mulig å la COM+ proxy komponenten avgrense hvilken funksjonalitet som skal eksponeres fra EJB komponenten.

COM+ proxy komponenten kan gjennom sikkerhetkontekst styrt fra COM+ serveren gi forskjellige brukergrupper tilgang til forskjellige implementasjoner av proxy grensesnitt mot EJB komponenten. Det er da for COM+ komponenten mulig å implementere integrert sikkerhet mot Windows 2000 brukerdatabasen lokalt eller fra domene. Dette gjør at de nettverksgrupper i Windows 2000 som en bruker er medlem av kan styre tilgangskontroll. På den måten er det driftspersonell for nettverk som indirekte styrer tilgang til COM+ komponenten. EJB serverkomponenten må tilbys utad i kontekst av anonym bruker. D.v.s at alle brukere direkte mot EJB server komponenten har fulle rettigheter. Dette kan være en sikkerhetsrisiko for implementasjoner.

Levetiden for et system utviklet etter prinsippene i testsystemet er uviss. I og med at det er to forskjellige Java miljøer som skal fungere sammen så er man i stor grad

avhengig av at versjoneringen av disse er i fase. Sun JDK er nå i versjon 1.3 mens Microsoft implementert forholdsvis kompatibel med versjon 1.2. Disse versjonene avviker ikke fra hverandre på essensielle områder som kan få følge for testsystemet. Dette kan i fremtiden endre seg.

Det er ikke system via Microsoft SDK for Java 4.0 for å takle transaksjoner på kryss av J2EE og Windows DNA. Windows DNA bruker Distributed Transaction Coordinator (DTC) for støtte distribuerte transaksjoner. I kontekst av denne oppgaven er det ikke funnet støtte for DTC i J2EE. J2EE bruker JTS for distribuerte transaksjoner som ikke støttes av Windows DNA. Man kan allikevel oppnå en grad av transaksjoner ved å la atomiserte operasjoner starte og avslutte i et av miljøene. Transaksjonens kontekst er da kun i J2EE eller Windows DNA.

8.4.2 Java mot COM - JACOB

8.4.2.1 Innledning og hypotese

En mulig metode for å integrere J2EE og Windows DNA på business-laget kan være å benytte JACOB. JACOB vil da fungere som en bridge for å åpne for kommunikasjon fra J2EE miljø mot Windows DNA miljø.

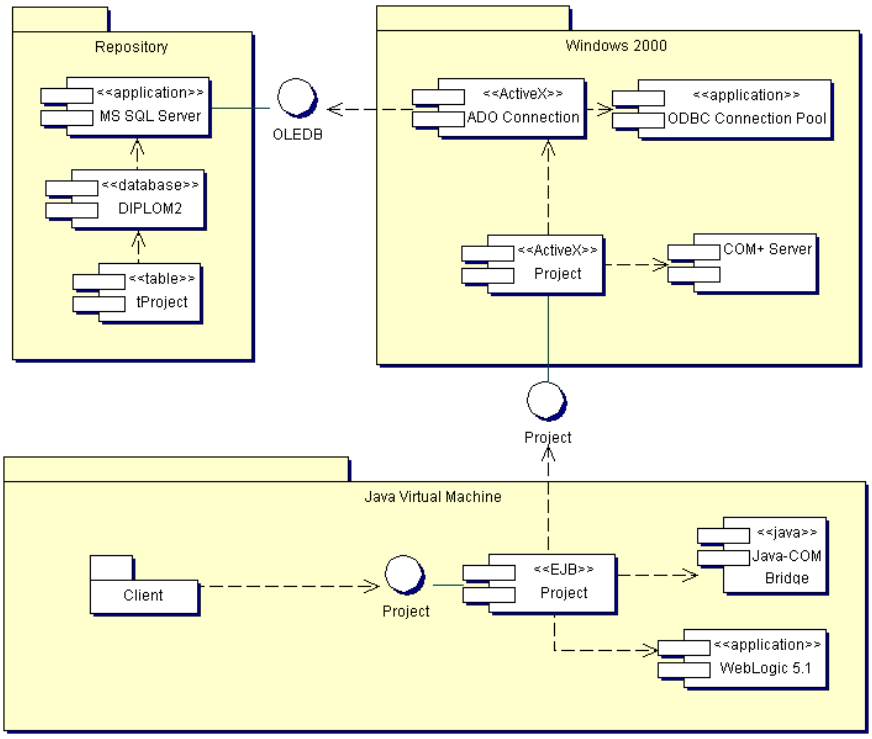
I denne delen skal det testes om det kan løses at J2EE EJB serveren (WebLogic) kan snakke med Microsoft DNA (COM+) på businesslaget ved hjelp av JACOB. I Figur 10 vises en strukturell skisse over en mulig vei å gå for å nå en COM+ komponent fra en EJB komponent via Java-COM bridge.

EJB Serveren skal tilby COM+ Project komponenten indirekte gjennom EJB komponenten Project som egentlig er en proxy for COM+ komponenten. Klienter som kopler seg mot EJB serverens Project komponent skal ikke trenge å supportere JACOB direkte, hverken direkte eller ved JACOB exception klasser, og skal tro at Project komponenten er en ren EJB komponent. JACOB skal kun brukes på laget der meldingsutveksling mellom enterprise komponentene, henholdsvis EJB komponent som proxy og COM+ som server, foregår. EJB proxy komponenten skal benytte JACOB for å snakke med COM+ komponenten.

8.4.2.2 Eksperimentell prosedyre

8.4.2.2.1 Oversikt

For å teste JACOB som en mulig bridge på Java-siden mellom J2EE og Microsoft COM+ så er det utviklet og konfigurert et testmiljø som vist i Figur 10. Den overste halvdelen av figuren er det initielle scenariet for COM+ miljøet som beskrevet i kapittel 7.5. Grensesnittet Project tilbys som en COM+ komponent i Microsoft-miljøet. Nedre halvdel av figuren viser hvilke komponenter som utvikles og benyttes spesiellt for testen. JACOB installeres og WebLogic konfigureres til å kunne tilby JACOB's komponenter til sine EJB komponenter. En EJB komponent som tilbyr EJB grensesnittet Project i J2EE-miljøet implementeres slik at den bruker JACOB som proxy for kommunikasjon mot COM+ Project grensesnittet. EJB Project komponenten vil for testklienten fremstå som en ren EJB komponent, men vil implementeres som en proxy mot COM+ Project grensesnittet.



Figur 10: Java-COM via JACOB eksperimentell prosedyre

8.4.2.2.2 Lage EJB Project proxy komponent

Det er for denne testen laget et build.cmd skript for å kompilere og deploye Project proxy komponenten samt å bygge en enkel klientapplikasjon mot denne. Se Figur 10 for konseptuell figur. Det er i tillegg laget et go.cmd skript som starter klientapplikasjonen.

ProjectBean.java er implementasjonen av proxy funksjonaliteten. Project.java er grensesnittet for business metoder utad. ProjectHome er fabrikkeringsklassen for å

generere nye instanser av ProjectBean objekter. EJB Project proxy komponenten er avhengig av Java-biblioteker listet i Tabell 10.

```
import java.net.*;
import java.util.*;
import javax.ejb.CreateException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.jacob.com.*;
import com.jacob.activeX.*;
```

Tabell 10:Avhengigheter for EJB Project proxy med JACOB

Business metoden som gir proxy funksjonaliteten i EJB Project komponenten er implementert som listet i Tabell 11. Det er forholdsmessig lite programkode som skal til for å lage en proxy-funksjonalitet mot COM+ Project komponenten. ActiveXComponent lager en instans av COM+ Project komponenten. getObject() gir håndtak til instansen. Dispatch.call er en statisk metode som kalles med instansen av COM+ Project komponenten, metodenavn med tilhørende parameter som parametre. Resultatet av å kalle getProjects(1) på COM+ Project komponenten returneres i en instans av Variant klassen.

```
public String getProjects (int emp_id)
{
  Object sControl = null;
  String expr = new String(emp_id);

  ActiveXComponent sC = new ActiveXComponent("Diplom.Project");
  sControl = sC.getObject();
  Variant result = Dispatch.call(sControl, "GetProjects", expr);

  return new String( result.toString() );
}
```

Tabell 11: Implementasjon av EJB Project proxy metode med JACOB

8.4.2.2.3 Oppsett av WebLogic Server

For at EJB komponenter skal kunne bruke JACOB som proxy mot COM+ komponenter så må WebLogic settes opp med referanser til implementasjonen av JACOB. Dette gjøres ved å oppdatere oppstartsfilen startweblogic.cmd med JAVA_CLASSPATH som vist i Tabell 12.

```
set JAVA_CLASSPATH=c:\java\jacob\jacob.jar;.\classes\boot;.\eval\cloudscape\lib\cloudscape.jar
```

Tabell 12: JAVA_CLASSPATH for WebLogic med JACOB støtte

WebLogic må i tillegg konfigureres til å laste EJB ProjectHome grensesnittet slik at det kan opprettes nye instanser av EJB Project. Dette gjøres ved å oppgi i konfigurasjonsfilen weblogic.properties stien til filen ejb_standard_project_jacob.jar, som inneholder implementasjonen av EJB klassene.

```
weblogic.ejb.deploy=C:/weblogic/myserver/ejb_basic_project_jacob.jar
```

Tabell 13: Deploy konfigurasjon for EJB Project med JACOB

JACOB bruker Java Native Interface (JNI) som gir utviklere tilgang til implementasjon av programkode utenfor Java Virtual Machine (JVM). WebLogic gir i utgangspunktet ikke komponenter tilgang til å kalle funksjonalitet i underliggende OS under JVM da dette er en sikkehetsrisiko samt mulig ustabilitet for J2EE serveren. Det er imidlertid mulig å gi komponenter rettigheter til allikevel å få tilgang til underliggende OS. Det må da eksplisitt settes rettigheter i konfigurasjonsfilen weblogic.policy. For at JACOB skal fungere tilfredstillende må konfigurasjonsfilen oppdateres som vist i Tabell 14.

```
grant codeBase "file:d:/weblogic/-" {
  permission java.security.AllPermission;
};

grant codeBase "file:/c:/classes/-" {
  permission java.security.AllPermission;
};


grant codeBase "file:${java.home}/lib/ext/-" {
  permission java.security.AllPermission;
};

grant {
  permission java.security.AllPermission;
};
```

Tabell 14: Policy konfigurasjon for WebLogic med JACOB støtte

8.4.2.2.4 Lage testklient applikasjon for JACOB

Essensen i klientapplikasjonen er implementert som vist i Tabell 15. En instans av EJB Project klassen opprettes. Denne implementerer proxy funksjonaliteten mot COM+ Project komponenten via JACOB. Metoden getProjects(1) kalles 1000 ganger med timing for å få en profilering.

```
public void example()
    throws CreateException, RemoteException, RemoveException
  {
    Project project = (Project) narrow(home.create(), Project.class);

    Calendar timingStart = Calendar.getInstance();

    for (int i=0; i<1000; i++) {
        project.getProjects( 1 );
    }

    Calendar timingEnd = Calendar.getInstance();

    long timing;
    timing = timingEnd.getTime().getTime() - timingStart.getTime().getTime();

    log("Timing: " + timing );

    project.remove();
  }
```

Tabell 15: Implementasjon av klient for JACOB test

8.4.2.3 Resultater

Implementering og testing fungerte etter beskrivelsen ovenfor uten uforutsette problemer under utviklingen. Detter viser at det er mulig å oppnå integrasjon mellom EJB komponenter og COM+ komponenter på business layer mellom J2EE og Microsoft COM+ med JACOB som proxy implementert på Java-siden av systemene.

8.4.2.4 Timing av Java-COM med JACOB

Timing ble gjort fra klientapplikasjonen som benytter EJB Project proxy komponenten. Timingen ble utført med all server og klientsoftware installert fysisk på samme maskin. Maskinen som ble brukt var en Pentium 500 med 128 MB RAM. Det ble gjort 1000 kall mot EJB Project proxy komponentens metode getProjects med ansattid som parameter. 3 gjennomkjøringer ble gjennomført med resultater som listet i Tabell 16.

Antall kall	Kjøring 1	Kjøring 2	Kjøring 3
1000 Kall	9844 ms	9584 ms	9113 ms

Tabell 16: Resultater fra Java mot COM timing med JACOB

8.4.2.5 Drøfting

JACOB er implementert ved å benytte Java Native Interface (JNI). Under WebLogic kjører EJB komponenter i en avgrenset kontekst. En EJB komponent kan f.eks. ikke direkte bruke *java.io* klasser eller JNI. Det er imidlertid mulig å lette på sikkerhetskonteksten for EJB mot omverdenen. Dette må gjøres for å få JACOB til å fungere som en bridge mellom J2EE og Microsoft DNA. Å åpne på sikkerhetskonteksten kan være risikabelt med tanke på stabilitet. En EJB komponent som kaller JNI direkte har da også muligheten til å påvirke stabilitetsfaktorer i WebLogic Server implementasjonen.

Detaljert tilgangskontroll kan kun oppnås mot EJB proxy komponenten. Det er her mulig å gi brukere i J2EE miljø, med brukernavn og passord, tilgang til EJB proxy komponenten. Det er ikke mulig å ha detaljert sikkerhet mot COM+ komponenten, da denne kun kan styres ved integrert sikkerhet fra brukerdatabasen til Windows 2000. All tilgang må for COM+ komponenten derfor styres gjennom en proxy bruker mot COM+ miljøet som er felles innfallspunkt for alle brukere i J2EE miljøet. Windows 2000 proxy brukeren er samme bruker som J2EE miljøet kjører i sikkerhetskontekst av. For WebLogic Server installert som en Windows NT Service så vil proxybrukeren derfor

være den Windows 2000 brukeren som servicen kjører i kontekst av. Dette tilfører aksess mot COM+ komponenten en risikofaktor i og med at en Windows 2000 bruker, som proxy sikkerhetskontekst for EJB proxy komponenten, må ha tilgang til all funksjonalitet i COM+ komponenten. Å gi en proxy bruker superuser tilgang åpner for mulig uønsket aksess mot COM+ komponenten.

Det er ikke direkte mulig å gjennomføre native distribuerte transaksjoner mellom WebLogic Serveren og COM+ Serveren. COM+ Serveren er avhengig av Windows Distributed Transaction Coordinator (DTC) for å kunne delta i rollback funksjonalitet mellom heterogene systemer. Det er ikke støtte for DTC i WebLogic. Fra BEA er det tilgjengelig et produkt, Tuexido, som er et system for å håndtere distribuerte transaksjoner. Tuexido kan benyttes som en ekstern koordinator for transaksjoner mellom COM+ og J2EE generellt ved at den tilbyr et API for dette. Dette krever imidlertid eksplisitt implementasjon av programkode for å håndtere dette. Dette er ikke testet i kontekst av denne oppgaven, men står som en åpen mulighet.

Levetiden til JACOB er uviss da denne er et Open Source prosjekt hvor enkeltpersoner eller grupper tar på seg klart definerte oppgaver for videre utvikling av komponenten. Det er derfor ikke et kommersielt produkt som har en definert fremgang til enhver tid. En ny utgave kommer når den kommer. Det er utviklere, og ikke markedet, som avgjør når en ny versjon blir tilgjengelig. Det er per idag ikke fremkommet informasjon som tilsier om en ny versjon vil komme.

JACOB er ikke videre kompleks å implementere. Det må imidlertid eksplisitt skrives kode for å dra nytte av JACOB som en Java til COM wrapper. Utviklere må kalle en *call* funksjon som fungerer som en inngangsport (proxy) mot COM+ komponenten. Alle kall til metoder og entiteter gjøres gjennom denne. Det er derfor mulig å avgrense og styre implementasjonen av EJB proxy komponenten til ikke å omfatte all funksjonalitet som COM+ komponenten kan tilby.

JACOB har i henhold til testscenariet vist seg å være vært stabil. Det er imidlertid rapportert fra JACOB's offisielle nettsted at det skal være oppdaget minnelekasje. Å reparere dette er i følge nettstedet en oppgave som er åpen for frivillige deltakere å ta tak i.

8.5 Konklusjon

Testingen har vist at Microsoft SDK for Java 4.0 er en kandidat til å implementere kommunikasjon fra COM+ proxy komponent mot EJB server komponent. Det er mulig å

oppnå detaljert kontroll i en COM+ proxy komponent av hvilke grensesnitt som tilbys fra en EJB komponent som server. Dette er mulig ved å implementere et kodeskall i COM+ proxy komponenten som indirekte implementerer ønsket funksjonalitet fra EJB server komponenten. Brukerkontroll er mulig å implementere detaljert og integrert med Windows 2000 brukerdatabasen for COM+ proxy komponenten, mens COM+ proxy komponentens kommunikasjon mot EJB komponenten må skje med anonym brukerkontroll for EJB komponenten. Distribuerte transaksjoner kan oppnås gjennom å implementere tredjeparts software, som Tuexido, for dette.

JACOB har vist seg som kandidat til å løse problemet med kommunikasjon mellom EJB komponenter og COM+ komponenter. Det er mulig å oppnå detaljert kontroll i en EJB proxy komponent av hvilke grensesnitt som tilbys fra en COM+ komponent som server. Dette ved å implementere et kodeskall i EJB proxy komponenten som indirekte implementerer ønsket funksjonalitet fra COM+ komponenten. Brukerkontroll er mulig å implementere detaljert for EJB proxy komponenten, mens EJB proxy komponentens kommunikasjon mot COM+ komponenten må ksje gjennom en proxy bruker for COM+ komponenten.

Et videre arbeid i forhold til bridging kan være å forske på hvordan man kan oppnå distribuerte transaksjoner gjennom å implementere tredjeparts software.

8.6 Kilder

The JACOB Project: A Java-COM bridge
By Dan Adler
http://users.rcn.com/danadler/jacob/

Microsoft Technologies for Java
© Microsoft Corporation, Inc.
http://www.microsoft.com/java/

Linear Ltd – Pure Java-COM bridge
© Linear Ltd.
http://www.linear.com/

Using WebLogic COM
© BEA Systems, Inc.
http://www.weblogic.com/docs45/classdocs/API_com.html

jacoZoom, Web Site
© infoZoom
http://www.infozoom.de/NS/jacozoom.html

Bridge2Java, Web Site

© IBM

http://www.alphaworks.ibm.com/tech/bridge2java/

Integrating Java and COM

© 1999 Microsoft Corporation, Av Chad Verbowski, Software Design Engineer

http://www.microsoft.com/java/resource/java_com.htm

FTP directory /developr/MSDN/UnSup-ed/ at ftp.microsoft.com

© Microsoft Coorporation, Inc.

http://www.microsoft.com/isapi/goftp.asp?TARGET=/developr/MSDN/UnSup-ed/

Using COM Objects from Java

© Microsoft Corporation, Inc.

http://www.microsoft.com/java/resource/java_com2.htm

Java™ Plug-in 1.3

© Sun Microsystems, Inc.

http://java.sun.com/products/plugin/

9 Simple Object Access Protocol (SOAP)

9.1 Innledning

Remote objekter kan gi et program nesten uavgensede krefter gjennom Internet men de fleste brannmurer blokkerer ikke-HTTP forespørsler. SOAP er en XML-basert protokoll som omgår denne problematikken og muliggjør kommunikasjon mellom prosesser på kryss av maskiner i nettverk.

Utviklere av software har ofte vanskelig for å bli enige om noenting. De har stor loyalitet til den teknologien de bruker. Kompromier ved å tilby koplingspunkter for andres teknologier er derfor ofte ikke aktuellt, noe som fører til at interoperabilitet mellom systemer kan bli en mangelvare.

Det hadde vært fint å kunne enes om en standard teknologi for interoperabilitet som alle kunne være enige om. Løsningen kan være å spesifisere et subsett av en minimalistisk, etablert teknologi. XML og HTTP er to slike etablerte teknologier. Simple Object Access Protocol (SOAP) definerer hvordan bruk av XML og HTTP kan nyttes til å aksessere tjenester og objekter på en måte som er uavhengig av plattform. SOAP fungerer som bro mellom heterogene programvare komponenter. Hovedmålet for SOAP er å fremme interaktivitet.

XML er et enkelt og utvidbart språk for koding av tekst. Siden XML er tekst så kan hvilken som helst applikasjon forstå kodingen så lenge en felles forståelse av kodingsregler i tegnsett er mulig å oppnå. Alle karakterer som er definert i ISO/IEC 10646 (Universal Character Set - UCS) gjennkjennes som standard i XML. XML spesifikasjonen krever at alle XML prosessorer uansett platform må gjenkjenne karakterer kodet med USC Transformasjonene UTF-8 og UTF-16. Siden de 256 første karakterene i UTF-8 er direkte knyttet til ASCII, så vil en XML prosessor kunne lese ASCII tekst filer.

Kombinasjon av HTTP og XML i en løsning tilbyr et helt nytt nivå av interoperabilitet. Kombinert med SOAP koding så vil f.eks. klienter skrevet i Microsoft Visual Basic lett kunne kalle tjenester implementert i CORBA som kjører på UNIX plattform. Der heterogene systemer idag knyttes sammen med propritære kryss-platform broer så vil SOAP direkte kunne lime sammen systemene.

9.1.1 Brannmurer og SOAP

Siden de fleste brannmurer konfigureres til å blokkere alt bortsett fra noen få porter, som f.eks. standard HTTP port 80, så vil protokoller for distribuerte objekter som DCOM og CORBA ha problemer fordi de baserer seg på dynamisk tildening av porter ved remote funksjonskall. SOAP baserer seg på HTTP som transportkanal og de fleste brannmurer kan lett konfigureres til å slippe gjennom HTTP. Det er derfor ingen problemer å bruke SOAP som endepunkter i klient og tjener.

Brannmurer kan sette filtre for HTTP forespørsler med Content-Type: text/xml-SOAP. De kan også tvinge klienter til å bruke M-POST istedenfor vanlig POST, filtrering kan skje på grensesnitt og metodedefinisjon i SOAP kodingen. Etter SOAP spesifikasjonen så er MethodName hodet påkrevd i kodingen. SOAP tjeneren er ansvarlig for å nekte tilgang hvis den ikke kan dekode forespørselen forsvarlig. Uten SOAP hodene i forespørselen så ville brannmuren måtte parse all XML koden for å lete etter tegn som krever restriksjoner. Dette ville krevd at brannmurer kunne gjenkjenne SOAP.

9.1.2 Sikkerhet i SOAP

SOAP definerer ikke protokoll spesifikke entiteter for sikkerhet. Siden SOAP bruker HTTP som transport så vil hvilken som helst standard HTTP sikkerhetsentitet eller endepunkt spesifikk sikkerhet kunne nyttes. SOAP kan fritt ta i bruk mekanismer for HTTP autentisering som f.eks. Secure Channel Communication (SSL) kjent som HTTPS.

9.2 Valg til eksperiment

9.2.1 MS SOAP Toolkit

9.2.1.1 Introduksjon

Microsoft® .NET Framework og Microsoft® Visual Studio .NET har som strategi å tilby XML og SOAP teknologi til utviklere. Visual Studio .NET er foreløpig i beta utgave, men Microsoft har gitt ut MS SOAP Toolkit 1.0 for utviklere som ønsker å benytte denne teknologien. MS SOAP Toolkit er i versjon 2.0 Release Candidate 0. Det anbefales derfor av Microsoft at denne versjonen ikke installeres på servere dedikert til drift.

Det er flere nøkkelteknologier i .NET. Disse er .NET Remoting, ASP .NET Web Services og ATL Web Services. Disse deler en rekke karaketerisitikker.

- XML for koding og dekoding av meldinger.
- SOAP 1.1 kompatibilitet. Inkuldert Section 5 SOAP koding. Dette tilrettelegger for interaksjon med andre tjenester som er implementert med SOAP grensesnitt.
- XML-fidelity (ikke-Seksjon 5 SOAP koding) for en rekke koplingsløse modeller.
- WSDL for beskrivelse. Dette er et XML skjema.
- Skalering ved bruk av stateless programmeringsmodell.
- Tett knytting til utviklingsverkøy i Visual Studio .NET.

ASP .NET Web Services og .NET Remoting deler i tillegg følgende:

- XCOPY deployment.
- System.NET, som fungerer både i server og klient miljø for nettverkskommunikasjon.
- Felles språk runtime for tread pool og managed kode.
- Sterkt knyttet SOAP support for egenskaper som SOAP Header og enveis meldinger.
- Potensiellt for å brukes med C#, Visual Basic .NET, eller andre språk for å skrive applikasjoner (Cobol, Phyton, ComopnentPascal o.s.v.).

9.2.1.2 Avhengigheter i MS SOAP Toolkit

For å installere MS SOAP Tookit må man bruke Microsoft Installer 1.1. Microsoft Installer er Microsofts installasjonsverktøy for å installere og deinstallere komponenter i Microsoft Windows OS miljøer. Denne installeres sammen med Windows 2000, Office 2000, Windows Me, SQL Server 2000 etc.

- Det kreves Visual Basic runtime. Denne må installeres før installasjonsprogrammet til SOAP Toolkit kjøres.
- Det er avhengigheter mot Microsoft Internet Explorer versjon 5.0.
- MSXML 3.0 vil installeres med denne utgaven hvis det er en eldre versjon av MSXML installert.
- SOAP Toolkit 2.0 inkluderer komponenter som er skrevet i VB, så VB runtime er påkrevd.

9.2.1.3 Status for MS SOAP Toolkit

MS SOAP Toolkit 2.0 kommer i release versjon i Juni 2001. Det anbefalse av Microsoft å vente med å sette i drift software utviklet med SOAP Toolkit inntil release.

9.2.2 Apache SOAP

9.2.2.1 Introduksjon

Apache SOAP er en open-source implemetation av SOAP v1.1 og SOAP Messages with Attachments spesifikasjonene. Apache utvikles og vedlikeholdes av Apache SOAP community. Apache SOAP er i skrivende stund i utgave 2.1.

Apache SOAP supporterer de fleste av SOAP v1.1 spesifikasjonene. Implemetasjonen tilbyr tjenere infrastruktur for å deploye, vedlikeholde og kjøre tjenester som krever SOAP. Den tilbyr også klient API for å kalle tjenester på tjenere som implementerer SOAP. Full kildekode er tilgjengelig under Apache Software License. Meldingsflyt og RPC-kall supporteres over HTTP (og HTTPS) og SMTP. Det er i tillegg mulig å skrive SOAP tjenester i skriptspråk.

Tre kodingsregler for SOAP supporteres direkte: SOAP v1.1 Encoding, Literal XML og XMI. XMI koding (tilgjengelig under Java 1.2.2) supporterer automatisk koding og dekoding av objekter. SOAP koding tilbyr innebygget support for koding og dekoding av primitive typer, Strings, JavaBeans (ved bruk av refleksjon) og Vectors, Enumerations eller 1-dimensjonale arrayer av disse nevnte typene. For andre typer kan der kodes koder/dekoder som kan registreres i Apache-SOAP runtime. Literal XML koding tillater å sende XML elementer (DOM org.w3c.dom.Element objekter) som parametre ved å innkapsle literal XML serialization av elementtreet i DOM modellen. Ingen kode trengs å strives for å støtte dette.

9.2.2.2 Avhengigheter i apache SOAP

Apache SOAP avhenger av enkelte andre open-source prosjekter samt elementer av spesifikke J2EE API. Nedenfor følger en liste over de elementer som må være til stede for at Apache SOAP skal fungere tilfredstillende.

- Java 1.1 eller nyere, og en servlet motor som supporterer versjon 2.1 ellere nyere av Java Servlet API kreves.
- Apache Xerces-Java 1.2.3 (xerces.jar) kreves. Dette er en DOM parser som må inkluderes I classpath før alle andre eventuelle DOM parsere.

- JavaMail (mail.jar) og JavaBean Activation Framework (activation.jar) er pårkevd.
- XML koding krever bruk av Java 1.2.2 og XML4J 2.0.15. Classpath må ha xerces.jar først og deretter xml4j.jar i den rekkefølgen.
- For å implementere tjenester i skriptspråk så kreves Bean Scripting Framework.
- SSL (HTTPS) support krever Java 1.2.1 eller nyere og Java Secure Socket Extension.
- SMTP transport av SOAP meldionger krever SMTP og POP3 Bean-pakkene installert.

9.2.2.3 Begrensninger i Apache SOAP.

Følgende egenskaper definert i SOAP v1.1 spesifikasjonen er ikke implementert i versjon 2.1 av Apache SOAP.

- EncodingStyle atributten må ha kun en koding stil (se seksjon 4.1.1 i spesifikasjon).
- MustUnderstand atributten er ikke implementert.
- Root atributten er ikke implementert.
- ID/href linker og mulit-referanse aksessorer.
- Alle typer unntatt de XML Schema enkle typene string, boolean, double, float, long, int, short og byte mangler.

Følgende er avgrensninger på SOAP Messages with Attachments:

- Dokumentbase URI hentes ikke fra multipart Content-Location header.
- Support for relative URI i Content-Location headers er bergenset til å legge til dokumentbase URI til den relative URI.
- Tjenesten for transport av SOAP meldinger supporterer ikke multipart meldinger.
- RPC forespørsler på tjener-siden kan ikke lokalisere attachments som ikke er referert fra SOAP-delen av forespørselen eller legge til attachment til respons utenom objektet som returneres.

9.2.2.4 Status for Apache SOAP

Apache SOAP er under kontinuerlig utvikling styrt etter open-source prinsippet. Det er fritt å melde seg som deltaker hvis man har noe å tilby prosjektet. Det er opprettet to mailing lister tilpasset rollene man kan ha i forhold til Apache SOAP. Disse er SOAP User mailing lista *soap-user@xml.apache.org* og SOAP Developer mailing lista *soap-dev@xml.apache.org*.

9.3 Gjennomføring av eksperiment

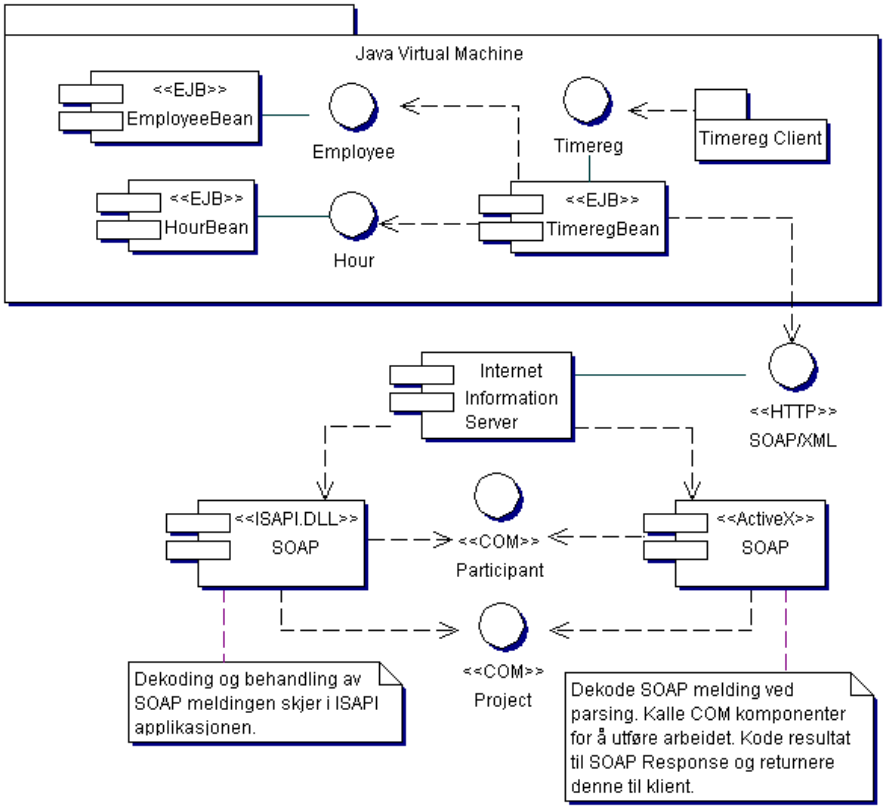
9.3.1 Java mot COM - SOAP

9.3.1.1 Innledning og hypotese

I denne delen skal det testes om det kan løses at J2EE EJB serveren (WebLogic) kan snakke med Microsoft DNA (COM+) på businesslaget ved hjelp av SOAP meldinger. I Figur 11 vises en strukturell skisse over mulige veier å nå en COM+ komponent fra en EJB komponent via SOAP.

COM+ Project komponenten skal tilbys ut fra Microsoft DNA miljø med et SOAP grensesnitt. SOAP server komponenten implementeres ved hjelp av MS SOAP Toolkit under Internet Information Server 5.0 (IIS). IIS 5.0 vil fungere som transportkanal for HTTP pakker som inneholder SOAP meldinger innkapsulert i XML struktur.

EJB Serveren skal tilby COM+ Project komponenten indirekte gjennom EJB komponenten Project som egentlig er en proxy for COM+ komponenten. Klienter som kopler seg mot EJB serverens Project komponent skal ikke trenge å supportere SOAP, og skal tro at Project komponenten er en ren EJB komponent. SOAP skal kun brukes på laget der meldingsutveksling mellom enterprise komponentene, henholdsvis proxy og server, foregår. EJB proxy komponenten skal benytte Apache SOAP for å snakke SOAP utad. EJB proxy komponenten får en forespørsel fra en klient, formulerer en SOAP melding og bruker Apache SOAP til å sende meldingen samt å få tilbake eventuell respons melding.



Figur 11: Struktur skisse for Java til COM via SOAP

9.3.1.1.1 Avgrensninger

Det er valgt å kun se på hvordan man kan nytte SOAP-RPC kall i integreringen. Avgrensninger i funksjonalitet er gjort i Project komponenten. Komponenten kan kun motta forespørsel om å returnere alle prosjekter der en ansatt, ut fra ansatt nummer, er deltaker.

9.3.1.2 Eksperimentell prosedyre

9.3.1.2.1 Oppsett av ODBC

For å nå frem til data som er lagret i repository, så må det settes opp et oppkoplingspunkt. I denne testen er det valgt å bruke oppkopling gjennom ODBC (Open DataBase Connector). På datamaskinen som kjører IIS 5.0 settes det opp en ODBC kopling i administrasjonsprogrammet for ODBC under ”Start | Control Panel | Administrative Tools”. Koplingen er av type ”SQL Server” og er navngitt ”DIPLOM”.

9.3.1.2.2 Lage COM+ SOAP server komponent

For å lage COM+ SOAP server komponenten er det valgt å bruke Visual Basic 6.0. Denne er valgt på grunn av enkel syntax. Dette gjør at det er enkelt og raskt å endre forretningslogikk etter behov uten komplekse oppsett i utviklingsverktøy.

Det er valgt å lage COM+ SOAP server komponenten i form av en ActiveX DLL. Visual Basic prosjektet er navngitt DiplomSOAP og komponentklassen er navngitt ProjectSrv. Det er da mulig å opprette en instans av ProjectSrv i Microsoft DNA miljø ved å bruke navnebetegnelsen ”DiplomSOAP.ProjectSrv”. Komponenten innkapsuleres i DiplomSOAP.dll filen.

For at komponenten skal kunne kode og dekode SOAP meldinger, kople seg mot repository, og kunne samarbeide med IIS 5.0 så må Visual Basic prosjektet referere til følgende kode biblioteker:

- Microsoft ActiveX Data Objects 2.5 Library
- Microsoft SOAP Type Library
- Microsoft Active Server Pages Type Library

Interaksjon mellom ActiveX komponenten og IIS 5.0 krever et løst definert grensesnitt som lett kan utvides. Det som er essensen med grensesnittet er at referanser til predefinerte instanser av Request og Response objektene overføres som innparametre. I testen er grensesnittet definert følgende:

```
Public Sub Process( ByVal Request As ASPTypeLibrary.Request, ByVal Response As
ASPTypeLibrary.Response )
```

I ActiveX komponenten er det definert en konseptuell struktur for å håndtere mottak og sending av SOAP-meldinger. Denne strukturen kan gjenbrukes generellt for implementasjon av ActiveX SOAP server komponenter. Strukturen er som følger:

1	Set Reader = New MSSOAPLib.SoapReader	Opprett en instans av Microsofts ActiveX SOAP-RPC koder og dekoder.
2	Reader.Load Request	Last SOAP/XML meldingen inn i SOAP-RPC dekoderen.
3	MethodName = Reader.RPCStruct.baseName	Hent identifikator for ønsket metode fra SOAP-RPC dekoderen.
4	Param = Reader.RPCParameter(“emp_id”).Text	Hent parameter for metoden.
5	Response.ContentType = “text/xml”	Sett responsmelding til å kodes og sendes som en XML melding av IIS 5.0.

6	Response.Write [XML respons melding]	Send SOAP-RPC respons til IIS 5.0 som videresender denne til klienten.

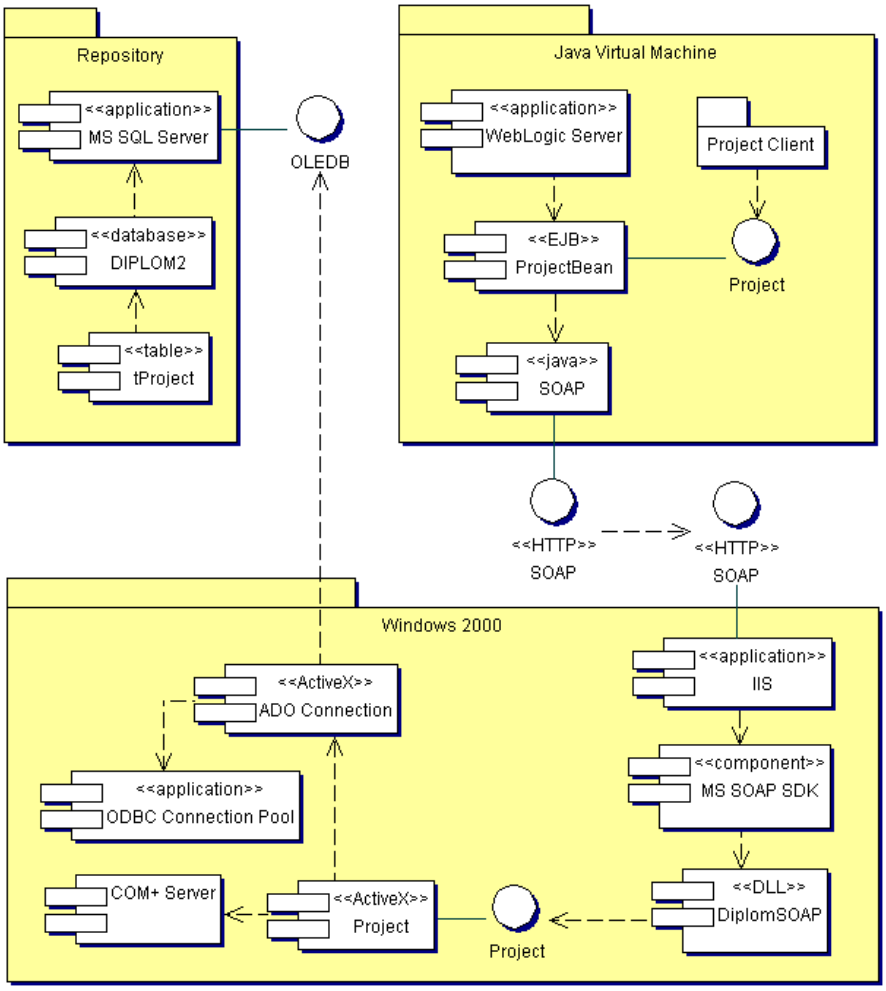
For å hente data fra repository bruker ActiveX komponenten ADODB som er en ActiveX implementasjon over OLEDB. En instans av ADODB.Connection objektet brukes til å lagen en kanal gjennom ODBC via Data Source Name (DSN) ”DIPLOM” til repository. Ved å sende en SQL spørring (Tabell 17) gjennom denne kanalen så returneres en instans av ADODB.Recordset objektet. Denne instansen gir tilgang til data returnert fra repository, og brukes til å bygge elementene i SOAP-RPC response meldingen (Figur 13).

```
SELECT dbo.tProject.pro_no, dbo.tProject.pro_title
FROM dbo.tParticipant
INNER JOIN dbo.tProject ON dbo.tParticipant.pro_id = dbo.tProject.pro_id
WHERE dbo.tParticipant.emp_id = [?]
```

Tabell 17: SQL spørring for å hente prosjektliste

ActiveX komponenten kompileres i Visual Basic til native prosessorkode i filen DiplomSOAP.DLL. Denne filen inneholder informasjon til å registrere og instansiere komponenten etter behov i Windows DNA miljø. Klient som ønsker å instansiere ActiveX komponenten må kunne lokalisere hvor den fysiske komponenten er lokalisert. Informasjon om dette må lagres i Windows Registry. Dette gjøres ved å kjøre regsvr32.exe med DiplomSOAP.DLL som parameter.

ProjectSrv komponentens grensesnitt er etter dette eksponert mot omverdenen og kan benyttes gjennom DCOM protokollen. Se Figur 12.



Figur 12: Java-COM via SOAP eksperimentell prosedyre

9.3.1.2.3 Oppsett av IIS 5.0

IIS vil fungere som en router for SOAP meldinger. En virtuell katalog settes opp i IIS med navn DiplomSOAP. I denne katalogen må det lages en fil (DiplomSOAP.asp) (Tabell 18) som inneholder Active Server Pages (ASP) skriptkode for å overføre instansene av Request og Response objektene, som blir opprettet av IIS, til ProjectSrv ActiveX komponenten. Denne ASP-filen er limet mellom IIS og ProjectSrv komponenten.

```
Set ProjectSrv = Server.CreateOpbject("DiplomSOAP.ProjectSrv")
ProjectSrv.Process Request, Response
```

Tabell 18: ASP skript kode for å linke IIS mot ProjectSrv

ActiveX ProjectSrv komponenten er nå eksponert til omverden via SOAP-RPC og kan ta imot forespørsler og genere svar i form av SOAP-RPC meldinger. Se Figur 12.

9.3.1.2.4 EJB Project proxy komponent

Det er for denne testen laget et build.cmd skript for å kompilere og deploye Project proxy komponenten samt å bygge en enkel klientapplikasjon mot denne. Se Figur 12. Det er i tillegg laget et go.cmd skript som starter klientapplikasjonen.

ProjectBean.java er implementasjonen av proxy funksjonaliteten. Project.java er grensesnittet for business metoder utad. ProjectHome er fabrikkeringsklassen for å generere nye instanser av ProjectBean objekter. EJB Project proxy komponenten er avhengig av Java-biblioteker listet i Tabell 19.

```
import java.net.*;
import java.util.*;
import javax.ejb.CreateException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import org.apache.soap.util.xml.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;
```

Tabell 19: Avhengigheter for EJB Project proxy med SOAP

Business metoden som gir proxy funksjonaliteten i EJB Project komponenten er implementert som listet i Tabell 20. Det er viktig i prosessen med å debugge SOAP-RPC meldingene at både forespørsel og svar kan snappes opp og vises av en meldingssporer. Til testen er sporingsapplikasjonen MSSoapT (Figur 13) fra MS SOAP Toolkit brukt. MSSoapT er implementert som en generell proxy for SOAP meldinger. MSSoapT mottar en SOAP forespørsel og viser denne i grensesnittet. Meldingen videresendes til den egentlige SOAP serveren som prosesserer og returnerer en SOAP response melding til MSSoapT. MSSoapT viser denne meldingen og videresender den til klienten, som i denne testen er EJB Project komponenten.

```
public String getProjects (int emp_id)
  {

  URL url;

  try {
                        url = new URL ("http://localhost:8080/DiplomSOAP/DiplomSOAP.asp");

  // Build the call.
  Call call = new Call ();
  call.setTargetObjectURI ("uri:Project");
  call.setMethodName ("GetProjects");
  call.setEncodingStyleURI( Constants.NS_URI_SOAP_ENC );
  Vector params = new Vector ();
  params.addElement (new Parameter("emp_id", int.class,
new Integer (emp_id), Constants.NS_URI_SOAP_ENC));
  call.setParams (params);

  log("getProjects call called");
  Response resp = call.invoke ( url, /* actionURI */ "" );
  log("getProjects call returned");

  // Check the response.
```

```
    if (resp.generatedFault ()) {
      Fault fault = resp.getFault ();
      return fault.getFaultString ();
    } else {
      Parameter result = resp.getReturnValue ();
      return (String) result.getValue();
    }
                } catch (Exception e) {
                }
                return new String("should not get here!");
  }
```

Tabell 20: Implementasjon av EJB Project proxy metode med SOAP

9.3.1.2.5 Klientapplikasjon for Java-COM med SOAP

Essensen i klientapplikasjonen er implementert som vist i Tabell 21. En instans av EJB Project klassen opprettes. Denne bruker metoden som implementerer proxy funksjonaliteten mot ActiveX ProjectSrv via SOAP. Metoden kalles 1000 ganger i profileringstesten (da er MSSoapT koplet ut) som omtalt i kapittelet for resultater.

```
  public void example()
     throws CreateException, RemoteException, RemoveException
   {
     Project project = (Project) narrow(home.create(), Project.class);

     Calendar timingStart = Calendar.getInstance();

     for (int i=0; i<1000; i++) {
        project.getProjects( 1 );
     }

     Calendar timingEnd = Calendar.getInstance();

     long timing;
     timing = timingEnd.getTime().getTime() - timingStart.getTime().getTime();

     log("Timing: " + timing );

     project.remove();

   }
```

Tabell 21: Implementasjon av klient mot EJB Project proxy

9.3.1.2.6 Oppsett av WebLogic Server

For å integrere Apache SOAP med WebLogic så må CLASSPATH i startWebLogic.cmd oppdateres til å peke til implementasjonsfilene for Apache SOAP som vist i Tabell 22. Etter at dette er gjort så kan EJB komponenter benytte Apache SOAP klasser. Det er imidlertid viktig at xerces.jar biblioteket listes i CLASSPATH før andre XML parsere. Da unngås problemet med at WebLogic inneholder en XML parser som ikke er kompatibel med Apache SOAP implementasjonen.

```
set JAVA_CLASSPATH=c:\j2sdkee1.2.1\lib\j2ee.jar;
d:\java\xerces-1_2_3\xerces.jar;
d:\java\javamail-1.2\mail.jar;
d:\java\jaf-1.0.1\activation.jar;
d:\java\soap-2_1\lib\soap.jar;
.\classes\boot;.\eval\cloudscape\lib\cloudscape.jar
```

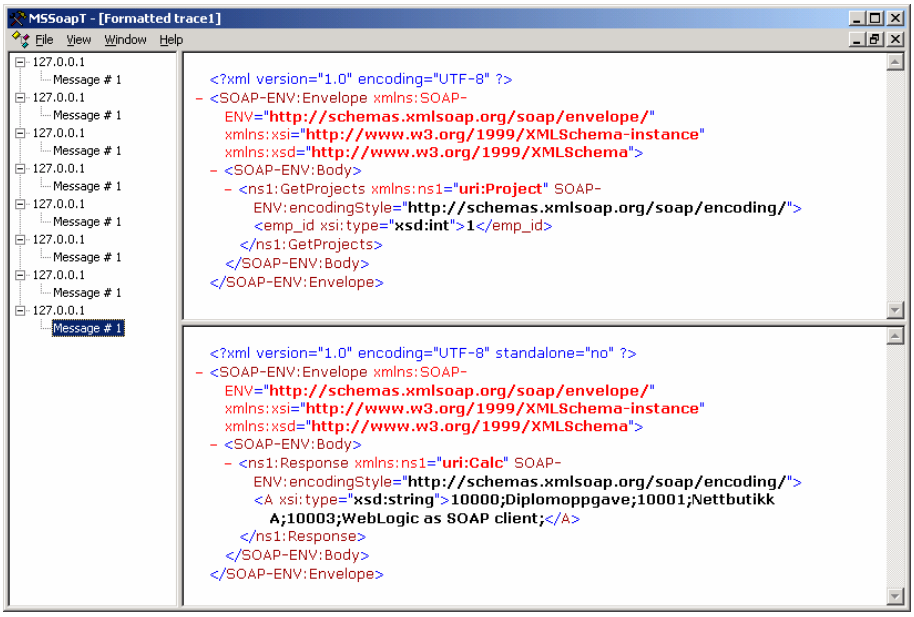
```
set WEBLOGIC_CLASSPATH=c:\j2sdkee1.2.1\lib\j2ee.jar;
d:\java\xerces-1_2_3\xerces.jar;d:\java\soap-2_1\lib\soap.jar;
.\license;.\classes;.\lib\weblogicaux.jar;
.\myserver\serverclasses;
.\mssqlserver4v70\classes;
.\mssqlserver4v70\license
```

Tabell 22: WebLogic classpath for Apache SOAP integrasjon

Det siste trinnet i WebLogic oppsettet er å oppdatere weblogic.properties fila med referanse til biblioteket som inneholder EJB Project proxy komponenten. I Tabell 23 er det vist hvordan dette ble implementert i testen.

```
weblogic.ejb.deploy=C:/weblogic/myserver/ejb_basic_project_soap.jar
```

Tabell 23: Deploy properties for EJB Project SOAP proxy



Figur 13: Sporing av SOAP-RPC medlinger

9.3.1.3 Resultater

Det lykkes å oppnå en stabil kommunikasjon fra klientapplikasjon mot repository gjennom EJB Project proxy komponent mot COM+ komponent på business laget. Klientapplikasjonen som er utviklet for denne testen trenger ikke å kjenne til underliggende arkitektur for å dra nytte av EJB Project komponenten.

9.3.1.3.1 Problemstillinger

Ved bruk av MSSoapT ble det avdekket at MS SOAP Toolkit ActiveX komponentene som skal hjelpe å kode/dekode SOAP-RPC meldinger ikke implementerte den generelle spesifikasjonen av SOAP. MS SOAP Toolkit koder SOAP-RPC meldinger uten å ta hensyn til entitetstyper i parametre. Meldingene kan derfor ikke dekodes av

Apache SOAP. Apache SOAP er avhengig av identifikatorer i SOAP meldingen (Figur 13) som indikerer hva slags type entitet en parameter er. Det var derfor nødvendig å skrive programkode for å rette opp dette.

Ny udokumentert versjon 2.1 av Apache SOAP. Problemer med integransjon mot siste versjon av xerces. Apache SOAP klarer ikke å dekode entiteter i SOAP-RPC til riktige native Java objekter. Det ble etter mye testing konstatert at nedgradering av xerces til versjon 1.2.3 var det som fungerte.

9.3.1.3.2 JAVA-COM SOAP Timing

Timing ble gjort fra klientapplikasjonen som benytter EJB Project proxy komponenten. Under timing sekvensen ble debuggingsverktøyet MSSoapT koplet ut slik at komponentene kunne testes i et virkelighetsnært driftsmiljø. Timingen ble utført med all server og klientsoftware installert fysisk på samme maskin. Maskinen som ble brukt var en Pentium 500 med 128 MB ram.

Det ble gjort 1000 kall mot EJB Project proxy komponentens metode getProjects med ansattid som parameter. Dette medførte at meldingsflyten som vist i Figur 12 ble gjennomløpt 1000 ganger. 3 gjennomkjøringer ble gjennomført med resultater som listet i Tabell 24.

Antall kall	Kjøring 1	Kjøring 2	Kjøring 3
1000 RPC Kall	23033 ms	23063 ms	23163 ms

Tabell 24: Resultater fra Java mot COM timing med SOAP

9.3.1.4 Drøfting

Etter dette eksperimentet med SOAP/HTTP som transportkanal fra EJB til COM+ så sitter man igjen med at dette er en metode som gir åpenhet på flere plan. SOAP grensesnittet mot Project komponenten er i prinsippet eksponert mot alle heterogene systemer som kan snakke TCP/IP. Mot SOAP grensesnittet oppnås det derfor en ekstremt stor grad av åpenhet. EJB Project proxy komponenten som eksponerer COM+ Project komponenten gjennom SOAP grensesnittet tilbyr J2EE miljøet en standard EJB komponent som er tilgjengelig gjennom EJB Server kontaineren som EJB native komponent og som CORBA-kompatibel komponent eksponert gjennom IIOP og WebLogic CosNaming navne tjeneste.

For SOAP er det ikke noen definerte entiteter for sikkerhet eller autentisering av brukere. Sikkerhet i form av kryptering mellom EJB Project proxy komponenten og

SOAP grensesnittet kan implementeres med Secure Socket Layer (SSL). Autentisering kan implementeres som standard HTTP autentisering over HTTP med eller uten SSL. Det ble imidlertid ikke funnet ut hvordan sikkerhet eller autentisering eksplisitt skulle implementeres i Apache SOAP. I og med at autentisering ikke er implementert så er dette en sikkerhetsrisiko da anonyme brukere kan gjøre kall mot SOAP grensesnittet. Det er mulig å konfigurere integrert sikkerhet med Windows 2000 brukerdatabasen men dog kun med en proxybruker som alle SOAP-kall vil gå gjennom. Autentisering er da på OS laget og utenfor implementasjonens kontroll. Hvis WebLogic kjører i kontekst av en bruker som har tilgang til IIS 5.0 virtuell katalog der implementasjonen av SOAP grensesnittet er lokalisert så oppnås en ”kanal” mellom SOAP grensesnittet og EJB Proxy komponenten. Dette kan gi en viss grad av sikkerhet. Med SSL vil i tillegg nettverkspakkene mellom SOAP grensesnittet og EJB proxy komonenten krypteres.

Det er ikke system for å håndtere transaksjoner gjennom SOAP da dette kun er en transportkanal. Windows DNA bruker Distributed Transaction Coordinator (DTC) for å håndtere distribuerte transaksjoner mellom heterogene systemer. Det er imidlertid en forutsetning at systemer utenfor Windows DNA struktur må ha eksplisitt støtte for DTC implementert for å kunne delta i en transaksjon i kontekt av denne. Dette er ikke tilfelle med J2EE. Det kan være mulig å oppnå støtte for transaksjoner via tredjeparts transaksjonssystemen. Dette vil kreve explisitt implementasjon men det er utenfor kontekst av denne oppgaven og kan være et område for videre forskning.

For å vurdere levetiden til et system implementert metode og teknologi som vist i dette eksperimentet er det viktig å se at systemet egentlig er to separate systemer som er koplet seriellt. Det første systemet er SOAP grensesnittet som tilbys utad mot heterogene systemer. Det andre systemet er EJB grensesnittet som tilbys utad fra EJB proxy komponenten. Begge systemene kan fritt reimplementeres med annen teknologi bakenfor bare grensesnittene beholdes utad. Det gir en god grad av fleksibilitet å kunne tilby to nivåer av grensesnitt.

Microsoft’s måte å implementere SOAP koding på gjennom støttekomponenter i Microsoft SOAP Toolkit har som default å ikke identifisere parametre i SOAP-RPC eksplisitt. Microsoft mapper internt en parameter til variabelstrukturen VARIANT som er Microsofts generaliserte måte å kode verdier på. Parametre som mottas via SOAP-RPC mot Microsoft SOAP Toolkit dekodes av en smart-dekoder som tolker verdien til en passe type av numerisk eller streng representasjon. Dette er et problem for Apache SOAP implementasjonen som eksplisitt krever at parameterens verditype skal identifiseres i

SOAP-RPC meldingen. Denne problemstillingen unngås ved å kode SOAP-RPC Response meldingen eksplisitt med identifisering av parametertyper istendenfor å bruke hjelpekomponenter for dette fra Microsoft SOAP Toolkit.

Minnekravene til eksperimentløsningen er relativt høye. Dette grunnes med at eksperimentet kjører over en rekke tyngre serverløsninger. Verstingene i minnekrav er IIS 5.0 og MS SQL Server 2000 som tar mye av minne til potensiell cache.

SOAP-implementasjon har vist seg stabil i forhold til testscenariet. Det er ikke avdekket noen svakheter som er målbare i testscenariet.

9.4 Konklusjon

SOAP egner seg til å åpne enkle tjenester mot heterogene systemer ved å tilby et enkelt grensesnitt over HTTP. Egner seg ikke veldig bra for realtime integrasjon, noe resultatene av eksperimentet viser i forhold til ytelse. En tettere integrasjon av EIS systemer som krever god ytelse hastighetsmessig kommer bedre ut med propritære bridger og tettere integrasjonsmetoder.

Sikkerhet ved bruk av SOAP er et område som det bør testes videre på. Den beste løsningen man kan oppnå i kontekst av eksperimentet vil være å implementere SSL på HTTP transportkanalen for å skape en ”tunnel” for så å kjøre J2EE server miljø og IIS 5.0 virtuell katalog for SOAP grensesnitt under kontekst av samme Windows 2000 bruker.

9.5 Kilder

RFC2068: Hypertext Transfer Protocol
Network Working Group 1997
http://info.internet.isi.edu/in-notes/rfc/files/rfc2068.txt

HTTP Extension Framework
By W3C, Henrik Frystyk Nielsen 2000
http://www.w3.org/Protocols/HTTP/ietf-http-ext

Extensible Markup Language (XML) 1.0 (Second Edition)
By W3C 2000
http://www.w3.org/TR/WD-xml-lang

XML-Data Specification
By W3C 1998
http://www.w3.org/TR/1998/NOTE-XML-data

Document Content Description for XML
By W3C 1998
http://www.w3.ort/TR/NOTE-dcd.

Namespaces in XML
By W3C 1999
http://www.w3.org/TR/REC-xml-names

RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax and Semantics
By Network Working group
http://www.ietf.org/rfc/rfc2396.txt

Microsoft .NET, Web Site
© Microsoft Corporation, Inc.
http://www.microsoft.net/net/

Microsoft Visual Studio .NET
© Microsoft Corporation, Inc.
http://msdn.microsoft.com/vstudio/nextgen/default.asp

MSDN, SOAP Developer Resources
© Microsoft Corporation, Inc.
http://msdn.microsoft.com/soap/

Simple Object Access Protocol (SOAP) 1.1
By W3C 2000
http://www.w3.org/TR/SOAP/

SOAP Messages with Attachments
By W3C 2000
http://www.w3.org/TR/SOAP-attachments

The Apache XML Procjet: Apache SOAP
© The Apache Software Foundation 1999
http://xml.apache.org/soap/

JAVA™ SERVLET TECHNOLOGY
© Sun Microsystems, Inc.
http://java.sun.com/products/servlet/

The Apache XML Project: Xerces Java Parser
© The Apache Software Foundation 2000
http://xml.apache.org/xerces-j/

JAVAMAIL™ API
© Sun Microsystems, Inc.
http://java.sun.com/products/javamail/

JAVABEANS™ ACTIVATION FRAMEWORK (JAF)

© Sun Microsystems, Inc.

http://java.sun.com/products/javabeans/glasgow/jaf.html

Messaging: The transport part of the XML puzzle

By Gordon Van Huizen, Director of Product Management, Progress Software

http://www-106.ibm.com/developerworks/library/xml-messaging/

XML Parser for Java

© IBM

http://www.alphaworks.ibm.com/tech/xml4j

IBM : developerWorks : Open source projects : bsf Project

© IBM

http://oss.software.ibm.com/developerworks/projects/bsf

Java™ Secure Socket Extension (JSSE) 1.0.2

© Sun Microsystems, Inc. 2000

http://java.sun.com/products/jsse/

alphaBeans - SMTP Project

© IBM

http://oss.software.ibm.com/developerworks/opensource/smtp/

alphaBeans - POP3 Project

© IBM

http://oss.software.ibm.com/developerworks/opensource/pop3/

XML Protocol Comparisons

© W3C.

http://www.w3.org/2000/03/29-XML-protocol-matrix

SOAP FAQ

© Developmentor.

http://www.develop.com/soap/soapfaq.htm

A Quick-Start Guide for Installing Apache SOAP

© Xmethods.

http://www.xmethods.com/gettingstarted/apache.html

10 CORBA

10.1 Innledning

10.1.1 CORBA

CORBA er forkortelsen av Common Object Request Broker Architecture, som er OMG's åpne og leverandøruavhengige arkitektur og infrastruktur som applikasjoner kan benytte for å samarbeide over nettverk. Ved å benytte standard protokollen IIOP så kan et CORBA basert program uavhengig av leverandør kjøre på nesten hvilken som helst datamaskin, operativsystem, utviklingspråk og nettverk og samspille med andre CORBA-baserte applikasjoner på kryss av miljøene.

10.1.2 J2EE og CORBA

CORBA (Common Object Recuest Broker Architecture) er en åpen standard for heterogene systemer. CORBA kompletterer Java™ plattformen ved å tilby et rammeverk for distribuerte objekter, tjenester for å støtte opp tundt rammeverket og inetoperabilitet med andre utviklingsspråk. Java™ plattformen kompletterer CORBA ved å tilby "Write Once, Run Anywhere™" portabilitet. Ved å kombinere Java plattformen med CORBA og andre nøkkelteknologier så tilbyr J2EE (Java 2 Platform Enterprise Edition) en komplett plattform for EIS systemer. CORBA standarder gir infrastruktur for interoperabilitet til J2EE.

CORBA er en integrert del av J2EE gjennom EJB (Enterprise JavaBeans™ technology), RMI over IIOP, Java IDL, og JTS (Java Transaction Service). EJB komponent modellen på tjener-siden forenkler utvikling av mellomvare komponenter som krever transaksjoner, salerbarhet og portabilitet. EJB tjenere tar bort en del av kompleksiteten ved å utvikle mellomvare komponenter ved å automatisk ta hånd om transaksjoner, sikkerhet og oppkopling mot repository. EJB komponenter bruker RMI/IDL CORBA delen for distribuert objektmodell, og bruker JTS for å støtte distribuerte transaksjoner. Siden EJB er implementert ved å benytte RMI-IIOP protokollen for å støtte interaktivitet mellom EJB komponeter i heterogene tjener-miljøer så innebærer dette implisitt at EJB arkitekturen gjennom CORBA tilbyr følgende interoperabilitet:

- En klient som bruker en ORB fra en leverandør kan aksessere EJB komponenter i en EJB tjener fra en annen leverandør.
- EJB komponenter i en EJB tjener kan aksessere EJB komponenter i en annen EJB tjener.
- En klient som ikke er utviklet i Java kan via CORBA aksessere en EJB komponent i en EJB tjener.

RMI over IIOP er en del av J2EE og J2SE (Java 2 Platform™ Standard Edition) versjon 1.3. RMI over IIOP gir utviklere mulighet til å skrive CORBA applikasjoner for Java™ plattformen uten å trenge å lære seg CORBA IDL (Interface Definition Language). For å lage CORBA applikasjoner i andre utviklingsspråk kan IDL genereres ut fra Java grensesnitt. RMI over IIOP inkluderer full funksjonalitet for en CORBA ORB (Object Request Broker).

Java IDL er en del av J2SE. Java IDL inkluderer en CORBA ORB som er tilgjengelig i alle installasjoner av J2SE. For å utvikle applikasjoner med Java IDL kan man definere grensesnitt ved å bruke IDL, og bruke en IDL-til-Java kompilator (idlj) for å generere programkode for CORBA grensesnitt og tjenerskjelett.

10.2 Valg til eksperiment

10.2.1 WebLogic Server 5.1

WebLogic Server implementerer J2EE standarden og har derfor implisitt implementert støtte for IIOP. Det er derfor også implementert navnetjener i WebLogic Server som har egenskap av å tilby Java komponenter utad som CORBA objekter. En klient mot en EJB komponent via IIOP vil oppleve EJB komponenten som en CORBA komponent. Det er derfor til eksperimentet valgt å benytte den innebygde navnetjeneren i WebLogic Server for å identifisere og lokalisere EJB komponenter som CORBA objekter i nettverket.

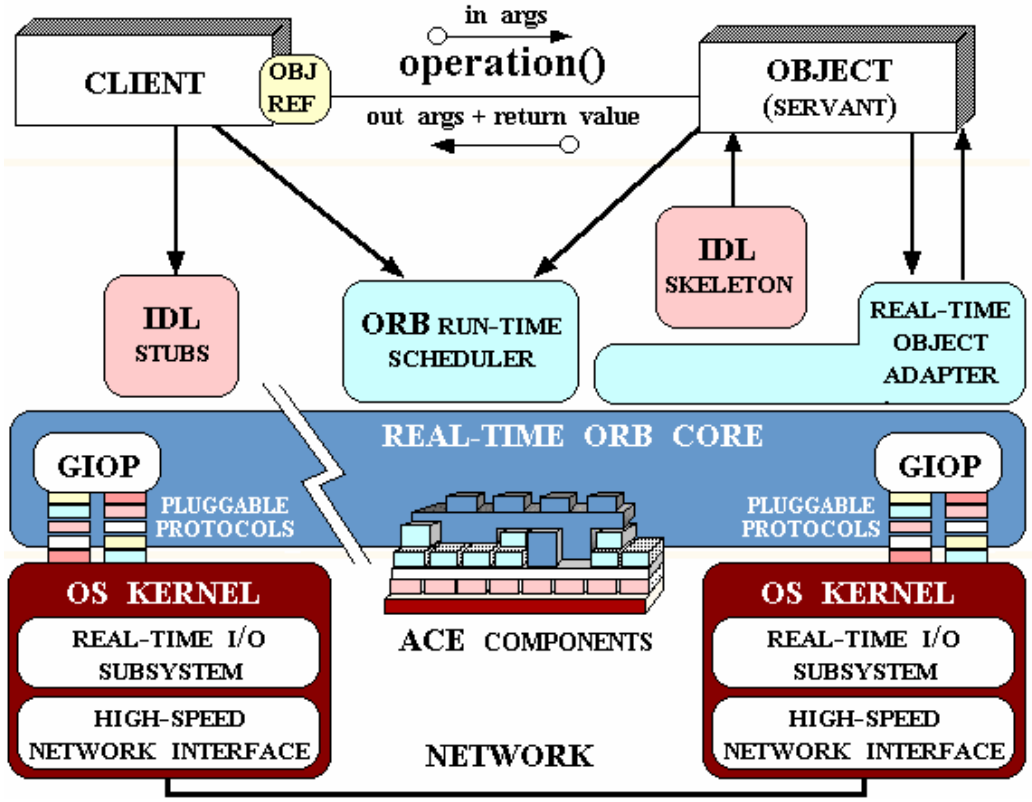
10.2.2 TAO

TAO er en OpenSource C++ ORB som implementerer de fleste egenskaper og komponenter i CORBA 2.4 spesifikasjonen. TAO kan lastes ned fra Internett uten utvikler- eller kjøretidslisensiering. TAO bygger på ACE (ADAPTIVE Communication

Environment™) som er en objekt orientert verktøypakke for å utvikle programkomponenter mot nettverk.

TAO er portet til OS plattformer som Win32 (WinNT 3.5.x, 4.x, 2000, og Win95/98 ved MSVC++, Borland's C++ Builder, og IBM's VisualAge på Intel and Alpha platforms), mange versjoner av UNIX (Solaris 1.x og 2.x på SPARC og Intel, SGI IRIX 6.x, HP-UX 10.x og 11.x, DEC/Compaq UNIX 4.x, AIX 4.x, SCO, og gratis tilgjengelige UNIX implementasjoner, som Debian Linux 2.x, RedHat Linux 5.2, 6.x, and 7.x, FreeBSD, and NetBSD), real-time operativsystemer (LynxOS, VxWorks, QnX Neutrino, og Chorus ClassiX 4.0), og MVS OpenEdition. I tillegg er TAO testet til å fungere i interoperabilitet med mange andre ORB imlementasjoner (Orbix, JacORB, ORB Express, VisiBroker, etc.). Dette indikerer at IIOP implementasjonen i TAO er robust og åpen. Med TAO følger IDL Compiler, Inter-ORB Protokoll Engine, ORB Core, Portable Object Adapter og CORBA Object Services (COS) Support.

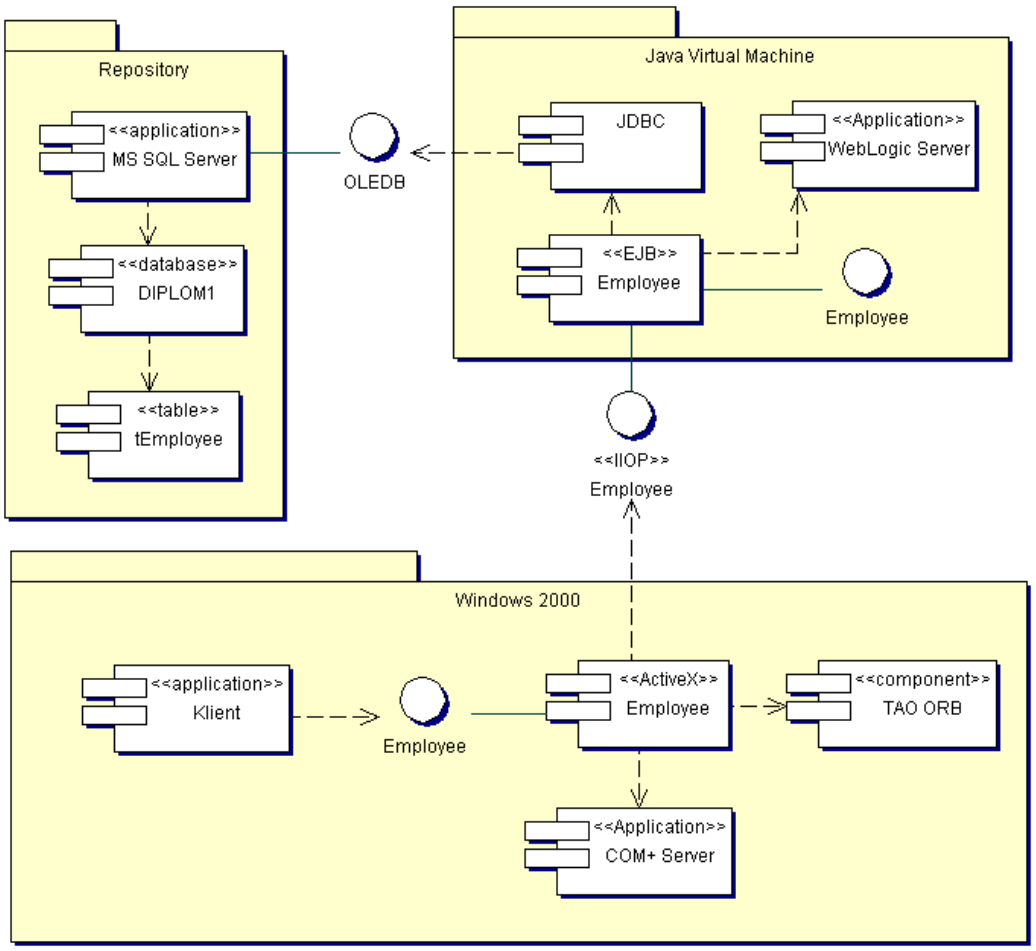
Målgruppen for TAO prosjektet er utviklere av distribuerte og integrerte applikasjoner som stiller krav til ytelse. Figur 14 viser skjematisk oversikt over TAO.



Figur 14: Oversikt over ACE+TAO ORB

10.2.3 Oversikt

I Figur 15 vises en grafisk presentasjon av det eksperimentelle scenariet som ønskes oppnådd i gjennomføringen av eksperimentet. Det gjøres først eksperimenter med gjennombruddsteknologi før selve implementasjonen av det eksperimentell scenariet startes.



Figur 15: COM+ via CORBA mot EJB eksperimentell prosedyre

10.3 Gjennomføring av eksperiment

10.3.1 IIOP som transport mellom COM+ og EJB

10.3.1.1 Innledning og hypotese

For å oppnå integrasjon fra COM+ komponent som klient og EJB komponent som tjener så bør det være mulig på implementere interoperabilitet via IIOP og CORBA. J2EE er implementert med støtte for IIOP. WebLogic Server inneholder en Navnetjener som er CORBA kompatibel. Det skal være mulig i WebLogic Server å registrere EJB

komponenter eksponert mot IIOP i Navnetjeneren. Det vil si at applikasjoner som snakker IIOP utenfor WebLogic Serveren skal identifisere EJB komponenter som CORBA komponenter uten å trenge å differensiere mellom disse.

Hvis det kan oppnås at en COM+ komponent kan snakke IIOP gjennom å implementere en ORB som proxy i COM+ komponenten så vil dette gi COM+ komponenten en kanal å kommunisere med EJB komponenter gjennom. For å implementere proxy ORB'en er det valgt TAO som er en gratis OpenSource CORBA 2.4 ORB implementasjon. Det er flere mulige valg innen gratis ORB implementasjoner så valget av TAO er tilfeldig. Det er forsøkt å lokalisere tilsvarende setting for eksperimentering via Internett, der WebLogic Server og TAO er involvert, men dette ble ikke funnet.

Da CORBA er relativt upløyd mark for meg som oppgaveskriver er det tatt en avgjørelse om at man først tester et generisk eksempel som følger med WebLogic der man tilrettelegger en RMI komponent mot navnetjener som CORBA komponent. Videre kopler man til denne komponenten med en C++ klient via Visibroker ORB. Visibroker er byttet ut med TAO fra starten av eksperimentet.

10.3.1.2 Eksperimentell prosedyre

10.3.1.2.1 Konfigurere WebLogic Server

For å konfigurere WebLogic Server til å laste Hello implementasjonen ved oppstart så legges konfigurasjon vist i Tabell 25 inn i weblogic.properties fila.

```
###################################
# WEBLOGIC RMI OVER IIOP PROPERTIES
# ------------------------------------------------
# This property is required to run the RMI over IIOP examples.
#
weblogic.system.startupClass.hello=examples.rmi_iiop.hello.HelloImpl
```

Tabell 25: RMI over IIOP konfigurasjon for WebLogic

10.3.1.2.2 Generere IDL

For å generere IDL fra Java grensesnitt ble det brukt hjelpeprogrammet *java2idl*. IDL filen inneholder, i tillegg til grensesnitt definert med IDL, referanser til andre IDL filer som må inkluderes sammen med denne IDL fila i forskjellige situasjoner. Referanser til andre IDL filer er i dette eksperimentet kommentert bort som vist i Tabell 26.

```
/* This is a generated file. Do not edit this file */
/**
 * This code was automatically generated at 16:28:09 on 24.mar.01 from
 * examples.rmi_iiop.hello.HelloWorld
 * by weblogic.rmi.rmic.IDLGenerator -- do not edit.
```

```
 *
 * @version 5.1.0 04/03/2000 17:13:23 #66825
 * @author Copyright (c) 2001 by BEA Systems, Inc. All Rights Reserved.
 */

//#include "orb.idl"
//#include "java/rmi/Remote.idl"

#ifndef ___examples_rmi_iiop_hello_HelloWorld
#define ___examples_rmi_iiop_hello_HelloWorld

module examples {
module rmi_iiop {
module hello {
  interface HelloWorld {
    void sayHello();

#pragma ID HelloWorld "RMI:examples.rmi_iiop.hello.HelloWorld:0000000000000000"

};
};
};
};
#endif
```

Tabell 26: Generert og revidert IDL for Hello klient

10.3.1.2.3 Identifisere navneserver

For at en klient som ønsker å bruke en tjener komponent implementert i CORBA skal kunne lokalisere denne så kan den snakke med en navnetjener som returnerer tjenerens referanse på kryss av nettverk til klienten. Navnetjeneren lokaliseres ved hjelp av en IOR (Inter-operable Object Reference) referanse. For å generere en IOR referanse så kan man bruke hjelpeprogrammet *host2ior*. I eksperimentet så kjører tjener og klient på samme fysiske maskinvare. Host er da localhost mens WebLogic Server lytter på port 7001 (se Tabell 27).

```
java utils.host2ior localhost 7001

Resultat:
IOR:000000000000002849444c3a6f6d672e6f72672f436f734e616d696e672f4
e616d696e67436f6e746578743a312e3000000000010000000000000660001
01000000000a6c6f63616c686f7374001b5900000038004245410000000000
0002849444c3a6f6d672e6f72672f436f734e616d696e672f4e616d696e67436f6
e746578743a312e3000000000080000000100000000000000a00000000000
000000000
```

Tabell 27: IOR referanse til navnetjener i WebLogic Server 5.1

10.3.1.2.4 Kompilere TAO

Før man kan bruke TAO som ORB så må TAO kompileres slik at nødvendige C++ lib filer er generert. TAO ble kompilert med Microsoft Visual C++ 6.0 i deler i release mode, dvs uten debug kode aktivert. Under kompilering av Hello-klienten ble det iterativt identifisert hvilke TAO C++ lib filer som måtte genereres. Disse ble grenerert fortløpende etter behov.

10.3.1.2.5 Klientapplikasjon

For klientapplikasjonen er det satt opp et prosjekt i MS Visual C++ 6.0. I prosjektet er det satt opp nødvendige referanser til C++ header- og lib-filer for at TAO skal kunne brukes som kodebibliotek. Utgangspunktet for C++ koden i klientapplikasjonen er RMI-IIOP eksemplet Hello som følger med WebLogic Server 5.1. Koden er endret slik at IOR referansen til navnetjeneren er kodet inn i klientapplikasjonen listet i Tabell 28. Dette er viktig fordi klientkoden senere i eksperimentet skal innkapsuleres i en COM+ komponent som ikke kan lese IOR fra kommandolinje.

```
#include <orbsvcs/orbsvcs/CosNamingC.h>
#include <iostream>
#include "HelloWorldC.h"

int main(int argc, char* argv[])
{

  try {
    CORBA::ORB_var orb =
                  CORBA::ORB_init(argc, argv);

                if (argc != 2) {
                cout << "Usage: HelloClient <ServerIOR>" << endl;
                return argc;
    }

  const char* ior =
"IOR:000000000000002849444c3a6f6d672e6f72672f436f734e616d696e672f4e616d696e67436f6e746578743a312e30000000000
100000000000000066000101000000000a6c6f63616c686f7374001b5900000038004245410000000000000002849444c3a6f6d672e6
f72672f436f734e616d696e672f4e616d696e67436f6e746578743a312e3000000000080000000100000000000000a000000000000
00000000";

  // string to object
    CORBA::Object_ptr o = orb->string_to_object(ior);

    // obtain a naming context
    CosNaming::NamingContext_var context = CosNaming::NamingContext::_narrow(o);
    CosNaming::Name name;
    name.length(1);
    name[0].id = "HelloServer";
    name[0].kind = "";
    name[0].kind = "";

    // resolve and narrow to RMI object
    CORBA::Object_var object = context->resolve(name);
    examples::rmi_iiop::hello::HelloWorld_var hi =
      examples::rmi_iiop::hello::HelloWorld::_narrow(object);

    if(!CORBA::is_nil(hi)) {
                  hi->sayHello();
    }
  }
  catch(const CORBA::Exception& e) {
    cout << "Failure: " << e << endl;
  }
  return 0;
}
```

Tabell 28: CORBA klient med TAO mot WebLogic Server

10.3.1.2.6 Teste klient mot WebLogic Server

Klienten er testet mot WebLogic Server og klarer å aktivere Hello RMI komponenten som er installert på tjeneren. Dette gir springbrettet for neste fase som er å innkapsulere klientfunksjonaliteten i en COM+ komponent som en proxy tjeneste.

10.3.1.2.7 COM+proxy komponenten

For å lage COM+ proxy komponent er det brukt Microsoft Visual C++ 6.0. Visual C++ har en wizard for å lage rammeverket for ActiveX komponenter i kontekst av ATL (Active Template Library). Rammeverk er opprettet og koden fra CORBA klienten er tilrettelagt og implementert i en metode i ActiveX komponenten som vist i Tabell 29. ActiveX komponenten er kompilert og registrert i COM+ som en tjener komponent.

```
// Hello.cpp : Implementation of CHello
#include "stdafx.h"
#include "HelloCOM.h"
#include "Hello.h"

#include <orbsvcs/orbsvcs/CosNamingC.h>
#include <iostream>
#include "HelloWorldC.h"

/////////////////////////////////////////////////////////////////////////////
// CHello

STDMETHODIMP CHello::SayHello()
{
        int argc = 0;
        char* argv[] = {"\0"};
  try {
    CORBA::ORB_var orb =
          CORBA::ORB_init(argc, argv);

 const char* ior =
"IOR:000000000000002849444c3a6f6d672e6f72672f436f734e616d696e672f4e616d696e67436f6e746578743a312e3000000000010
000000000000006600010100000000a6c6f63616c686f7374001b5900000038004245410000000000000002849444c3a6f6d672e6f726
72f436f734e616d696e672f4e616d696e67436f6e746578743a312e300000000008000000010000000000000000a0000000000000000000";

    // string to object
    CORBA::Object_ptr o = orb->string_to_object(ior);

    // obtain a naming context
    CosNaming::NamingContext_var context = CosNaming::NamingContext::_narrow(o);
    CosNaming::Name name;
    name.length(1);
    name[0].id = "HelloServer";
    name[0].kind = "";
    name[0].kind = "";

    // resolve and narrow to RMI object
    CORBA::Object_var object = context->resolve(name);
    examples::rmi_iiop::hello::HelloWorld_var hi =
      examples::rmi_iiop::hello::HelloWorld::_narrow(object);

    if(!CORBA::is_nil(hi)) {
                hi->sayHello();
    }
  }
  catch(const CORBA::Exception& e) {
//    cout << "Failure: " << e << endl;
  }
        return S_OK;
}
```

Tabell 29: Implementasjon av Hello COM+ proxy med TAO

10.3.1.2.8 Testing av COM+ proxy komponent

Ved testing ble det avdekket at COM+ proxy komponenten ikke fungerte. Det var ikke mulig å opprette en TAO ORB på grunn av en peker internt i TAO som pekte til en ulovlig minneblokk. Problemet ble lokalisert til å ligge i ORB_Init funksjonen. Det ble ikke funnet noen direkte løsning på problemet. En mulig løsning kan allikevel være å opprette ORB'en utenfor COM+ komponenten og servere en minnereferanse til ORB'en som en parameter inn mot COM+ komponenten. Dette er ikke testet. Testen stopper her da det ikke ble funnet noen løsning på dette.

10.3.1.3 Resultater

Det ble i dette eksperimentet oppnådd kontakt med WebLogic via TAO som ORB på klientsiden gjennom en C++ klient. Dette viser at man kan skape en vei inn til WebLogic Server komponenter via IIOP ved å bruke TAO.

Målet med testen var å få en COM+ komponent til å kommunisere med en EJB komponent. Det som viste seg som et problem var at ORB'en ikke vil initialiseres i kontekst av et ActiveX objekt. Det ble ikke funnet noen løsning på dette. Det var ikke hensiktsmessig å utføre timing tester da det ønskelige miljøet ikke ble oppnådd.

10.3.1.4 Drøfting

WebLogic er implementert etter J2EE standarden. Det er derfor en naturlig vinkling av oppgaven å se på muligheter for interoperabilitet mellom J2EE og Windows DNA ved bruk av IIOP. WebLogic eksponerer tjener komponenter som CORBA kompatible komponenter gjennom sin innebygde navnetjener.

TAO har vist seg gjennom eksperimentet som en kandidat til å implementere klientapplikasjoner via CORBA mot WebLogic Server i C++ programkode. Oppgaven med å integrere en TAO ORB i en COM+ komponent er i kontekst av eksperimentet ennå åpen. Implementert i COM+ komponenten vil ikke en TAO ORB initialisere. Dette kan kanskje løses ved å initialisere en orb og gi COM+ komponenten en referanse til ORB'en som parameter. Dette er i strid med oppgaveteksten da det ikke skal være nødvendig for en klient som benytter COM+ komponenten å kjenne til og supportere de underliggende arkitekturiske elementer.

Hvis man antar at problemet med å initialisere en TAO ORB fra en COM+ kopmponent kan løses så kommer man raskt videre til en annen aktuell problemstilling. Det er ofte ikke ønskelig å kode inn IOR referansen til navnetjeneren i programkoden slik

det er gjort i eksperimentet. En COM+ komponent tar ikke parametre fra kommandolinje slik som en ordinær applikasjon gjør. Det kan derfor være aktuellt å dra nytte av Windows Registry for å lagre og lokalisere IOR referansen slik at COM+ komponenter kan gjøre oppslag for å finne IOR referansen.

10.4 Konklusjon

CORBA er en etablert metodologi for å integrere komponenter i heterogene systemer via IIOP. J2EE er basert på IIOP som protokoll for interoperabilitet mot andre systemer og kan derfor tilby tjener komponenter som CORBA objekter mot omverdenen. WebLogic Server implementerer J2EE standarden og inneholder en navnetjener for å registrere og lokalisere tjener komponenter som CORBA komponenter. Det er derfor mulig for applikasjoner implementert i andre leverandørers ORB'er å kommunisere med WebLogic Server tjener komponenter.

Eksperimentet er forsøkt gjennomført med TAO som er en gratis delvis implementasjon av CORBA 2.4. Eksperimentet ble delvis vellykket i og med at kontakt mellom Windows DNA og J2EE ble oppnådd, men ikke i kontekst som forventet. Dette er derfor et område for mulig videre forskning

10.5 Kilder

CORBA Technology and the Java™ Platform
©1995-2001 Sun Microsystems, Inc.
http://java.sun.com/j2ee/corba/

Java™ IDL
©1995-2001 Sun Microsystems, Inc.
http://java.sun.com/products/jdk/idl/

RMI over IIOP
©1995-2001 Sun Microsystems, Inc.
http://java.sun.com/products/rmi-iiop/

Java™ Transaction Service (JTS)
©1995-2001 Sun Microsystems, Inc.
http://java.sun.com/products/jts/

CORBA, J2EE, and Tuxedo Interoperability and Coexistence
©2000 BEA Systems, Inc.
http://e-docs.bea.com/wle/interop/index.htm

A Detailed Comparison of CORBA, DCOM and Java/RMI

Av Gopalan Suresh Raj

http://www.execpc.com/~gopalan/misc/compare.html

TAO – Real time solutions for real time systems

By Douglas C. Schmidt.

http://www.cs.wustl.edu/~schmidt/TAO.html

ACE - The ADAPTIVE Communication Environment

By Douglas C. Schmidt.

http://www.cs.wustl.edu/~schmidt/ACE.html

OpenSource.org Web Site

© 2001 by the Open Source Initiative

http://www.opensource.org/

CORBA 2.4 Specification

© 1997-2001 Object Management Group, Inc.

http://www.omg.org/cgi-bin/doc?formal/01-02-01

11Generell drøfting

Det er i oppgaven testet tre hovedgrupper av metoder for å kople sammen J2EE og Windows DNA på business laget. Metodene er bridging, RPC over HTTP og IIOP. Disse metodene er kandidater til interkommunikasjon mellom heterogene systemer generellt.

Bridger er implementasjoner av programvarekomponenter for å knytte EIS systemer sammen. Noen bridger er implementert propritært mot EIS systemet som skal eksponeres (eks. Microsoft SDK for Java), mens andre bridger er mere åpne og generalisert til bestemte grupper av miljøer (eks. JACOB). Bridging gir den tetteste integrasjonen mellom EIS systemer innen de hovedgrupper av metoder som er vurdert i denne oppgaven. En genereralisert vurdering er at jo nærmere, og mer propritært, en bridge er implementert i forhold til et EIS system jo større er sannsynligheten for å oppnå bedre ytelse enn ved de andre hovedgrupper av metoder. Sikkerhet som autentisering kan ofte gjennom brigder ivaretas integrert med de EIS systemer som bridgen er konstruert for. Distribuerte transaksjoner er ikke triviellt å implementere gjennom bridger som ikke har integrert støtte for dette. En bridge skjermer for utviklerene kompleksitet i integrasjonen mellom EIS systemer. Bridger implementerer forskjellig grad av toleranse i forhold til friheter utviklerene har. Det er derfor viktig å vurdere hva slags funksjonalitet som er ønsket før en bridgeløsning skaffes. Flere kommersielle og gratis implementasjoner av bridger er tilgjengelige fra mange leverandører. Gratisimplementasjoner av bridger vedlikeholdes ofte av ideelle interessegrupperinger som f.eks. Open Source.

SOAP er definisjonen på en protokoll for interkommunikasjon mellom heterogene systemer. SOAP pakkes inn i XML over HTTP protokollen noe som gjør at SOAP innehar den høyeste grad av åpenhet som er testet i denne oppgaven. Det må imidlertid vurderes åpenhet mot sikkerhet hvis SOAP er et alternativ for integrasjon av EIS systemer. SOAP gir alene stor grad av åpenhet utad, men i kontekst av å være en kanal for informasjonsflyt mellom komponenter på business laget så er ofte sikkerhet og kryptering av data en aktuell problemstilling. Siden SOAP bruker HTTP som transportkanal så kan sikkerhet og kryptering eventuellt implementeres på transportnivå i HTTP. HTTP-autentisering gir mulighet for tilgangskontroll, mens kryptering med SSL gir mulighet for kryptering peer-to-peer. En faktor som spiller for SOAP er at Microsoft implementerer SOAP og XML som grunnkomponenter i sin .NET strategi. Dette betyr at tilgjengeligheten til tjenester via SOAP vil har et stort potensiale.

CORBA er en etablert metodologi for å integrere komponenter i heterogene systemer via IIOP. J2EE er basert på IIOP som protokoll for interoperabilitet mot andre systemer og kan derfor tilby tjener komponenter i form av CORBA objekter mot omverdenen. J2EE standarden implementerer en navnetjener for å registrere og lokalisere tjener komponenter i form av CORBA komponenter. Det er derfor mulig for applikasjoner implementert i andre leverandørers ORB'er å kommunisere med et EIS system basert på J2EE miljø. Microsoft DNA støtter ikke IIOP protokollen direkte så den må implementeres gjennom en bridge eller med en generell ORB-implementasjon som basis for en proxyløsning. Bridger for dette er å få gjennom flere ORB-produsenter som har til formål å tilby sin CORBA implementasjon integrasjon mot Windows DNA.

CORBA 3.0 spesifikasjonen beskriver "CORBA/DCOM Interoperability" som en ny egenskap. Dette er tenkt å gi direkte interoperabilitet mellom CORBA og DCOM.

12Konklusjon

Gjennom oppgaven er det vurdert løsninger innen tre hovedgrupper av teknologi. Hovedgruppene er bridging, RPC over HTTP og IIOP. Det som tilsynelatende er fellestrekkene for disse teknologiene er at de tilsynelatende er gjevngode kandidater til integrasjon av EIS systemer som J2EE og Windows DNA. Det er imidlertid grunnleggende forskjeller i egenskapene til de tre teknologiene.

Ønskes fokus på ytelse og tett integritet mot og mellom EIS systemene så er det med overveidene sannsynlighet at en bridge tilpasset formålet er den beste løsningen. Dette vil kunne gi utviklere en enkel plattform å implementere interoperabilitet over. Større og etablerte leverandører av mellomvarekomponenter tilbyr en rekke forskjellige bridge-løsninger. Ønskes en bridge-løsning er det viktig å vurdere grunding hvilke oppgaver i integrasjonen mellom J2EE og COM+ som skal løses slik at en egnet bridge-løsning kan velges.

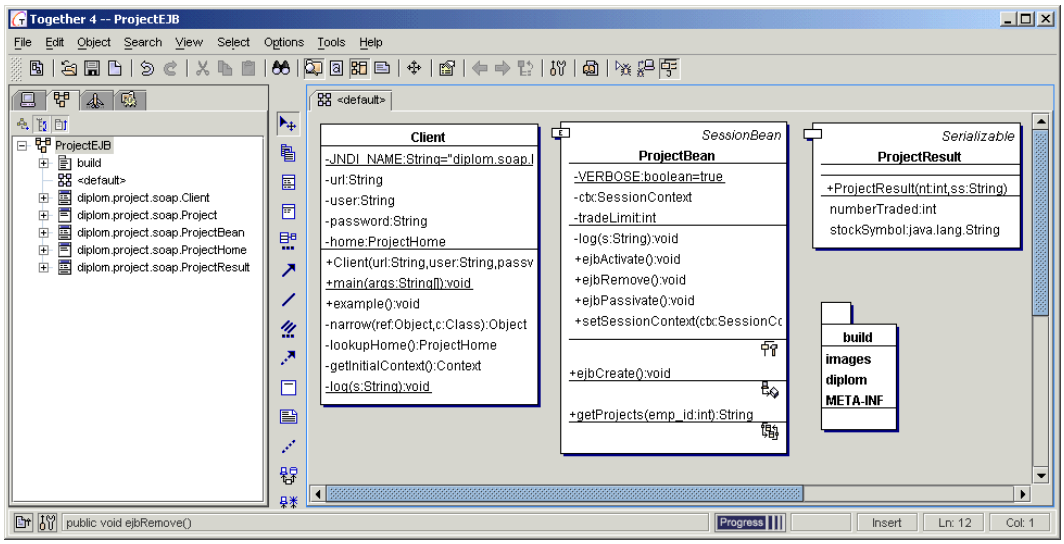
Ønskes en balanse mellom åpenhet og ytelse så vil CORBA og IIOP være god kandidat for en løsning. Dette er en mere kompleks løsning der det bør være forankret bakgrunnsinformasjon om CORBA som teknologi i organisasjonen. I miljøer som har andre løsninger implementert med CORBA så er dette den løsningen som gir størst grad av kompletthet. Siden det i CORBA 3.0 spesifikasjonen er beskrevet CORBA/DCOM interoperabilitet så er dette en teknologi som det kan være vel verdt å følge opp i fremtiden.

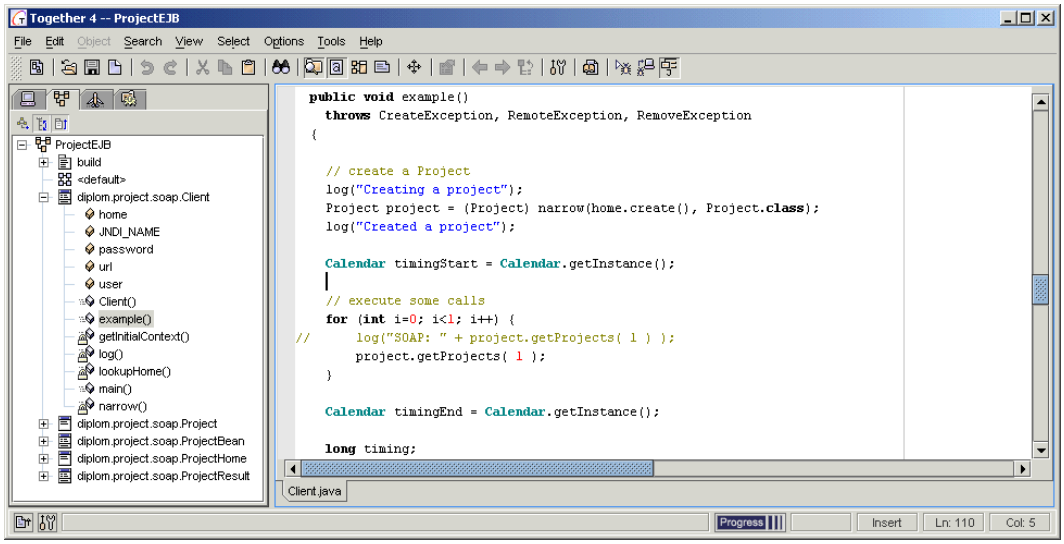
Ønskes fokus på stor grad av åpenhet og høy grad interoperabilitet mot heterogene miljøer så er SOAP en utmerket kandidat for en løsning. SOAP er imidlertid den av hovedteknologiene som gir den dårligste ytelsen. SOAP er i tillegg en spennende ny mulighet for integrasjon i og mot .NET miljøet til Microsoft. SOAP er den av de tre valgte teknologier som er enklest å implementere over brannmurer da all trafikk enkelt kan konfigureres til å sendes via kun en port.

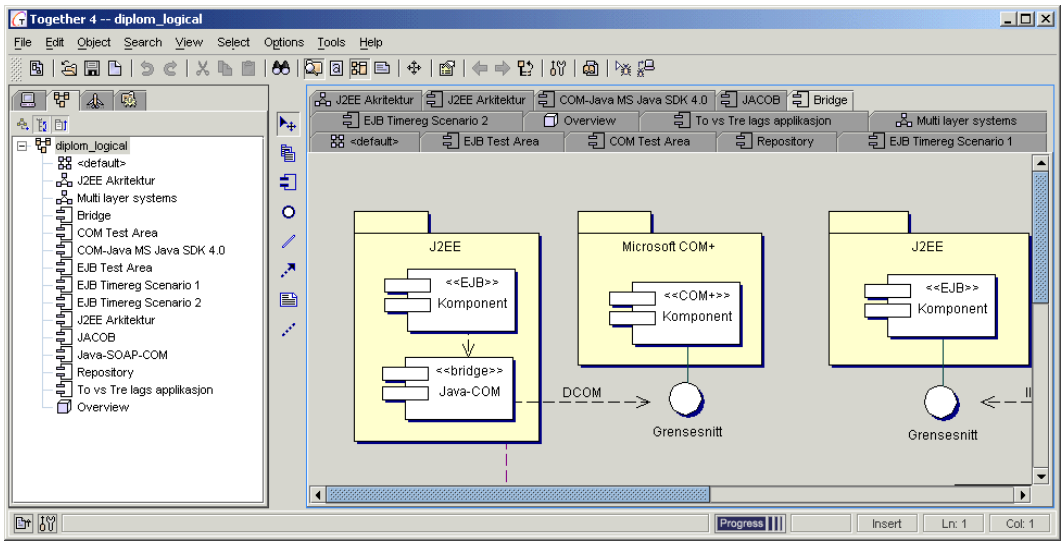
Ut fra eksperimenter og refleksjoner som er gjort i denne hovedoppgaven dukker det frem områder som kan være interessante for videre fordypning. Det aller mest spennende området for tett integrasjon mellom J2EE og COM+ er etter min mening CORBA. Forslag til videre studier som springer ut fra konteksten av denne hovedoppgaven kan være å teste kommersielle implementasjoner av CORBA som hevder at de implementerer bridging mellom CORBA og COM+ i forskjellig grad.

Vedlegg I. Verktøy grensesnitt

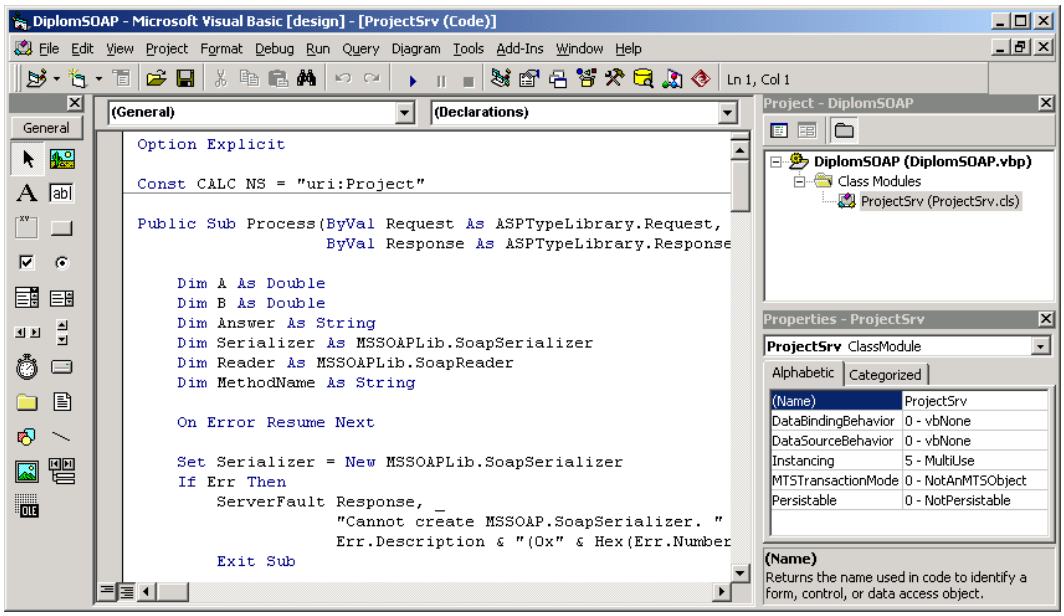
A. Together 4.1 IDE Grensesnitt







B. Visual Studio 6.0 IDE Grensesnitt



Vedlegg II. Programkode

A. setEnvMS.cmd

```
@echo on
@rem This script should be used to set up your environment for
@rem compiling and running the examples included with WebLogic
@rem Server. It contains the following variables:
@rem
@rem WL_HOME   - This must point to the root directory of your WebLogic
@rem             installation.
@rem JAVA_HOME - Determines the version of Java used to compile
@rem             and run examples. This variable must point to the
@rem             root directory of a complete JDK installation. See
@rem             the WebLogic platform support page
@rem             (http://www.weblogic.com/docs51/platforms/index.html)
@rem             for an up-to-date list of supported JVMs on Windows NT.
@rem
@rem When setting these variables below, please use short file names(8.3).
@rem To display short (MS-DOS) filenames, use "dir /x". File names with
@rem spaces will break this script.
@rem
@rem jDriver for Oracle users: This script assumes that native libraries
@rem required for jDriver for Oracle have been installed in the proper
@rem location and that your system PATH variable has been set appropriately.
@rem For additional information, refer to Installing and Setting up WebLogic
@rem Server (/install/index.html in your local documentation set or on the
@rem Internet at http://www.weblogic.com/docs51/install/index.html).

@rem Set user-defined variables.
set WL_HOME=C:\weblogic
@rem set JAVA_HOME=c:\java1.1.7
@rem set JAVA_HOME=c:\progra~1\micros~1.2
set JAVA_HOME=c:\JAVA\MSDK4
@rem set JAVA_HOME=C:\JDK1.3

@if exist %WL_HOME%\classes\boot\weblogic\Server.class goto checkJava
@echo.
@echo The WebLogic Server wasn't found in directory %WL_HOME%.
@echo Please edit the setEnv.cmd script so that the WL_HOME
@echo variable points to the WebLogic Server installation directory.
@goto finish

:checkJava
@if exist %JAVA_HOME%\bin\java.exe goto java
@if exist %JAVA_HOME%\Bin\JView.exe goto jview
@if exist %JAVA_HOME%\..\JView.exe goto jview
@echo.
@echo The JDK wasn't found in directory %JAVA_HOME%.
@echo Please edit the setEnv.cmd script so that the JAVA_HOME
@echo variable points to the location of your JDK.
@goto finish

:java
set RMIFORMS=
set JDK_CLASSES=%JAVA_HOME%\lib\tools.jar;
@if exist %JAVA_HOME%\lib\classes.zip goto jdk11
@goto setEnv

:jdk11
set JDK_CLASSES=%JAVA_HOME%\lib\classes.zip;
@goto setEnv

:jview
set JDK_CLASSES=%windir%\Java\classes\classes.zip;
set RMIFORMS=;%WL_HOME%\lib\rmiForMs.zip
@goto setEnv

:setEnv
set CLIENT_CLASSES=%WL_HOME%\myserver\clientclasses
set SERVER_CLASSES=%WL_HOME%\myserver\serverclasses
set SERVLET_CLASSES=%WL_HOME%\myserver\servletclasses
```

```
set CLASSPATH=c:\java\jacob\jacob.jar;c:\java\xerces-1_3_1\xerces.jar;c:\java\soap-2_1\lib\soap.jar;%JDK_CLASSES%%WL_HOME%\license;%WL_HOME%\classes;%WL_HOME%\lib\weblogicaux.jar;%CLIENT_CLASSES%;%SERVER_CLASSES%%RMIFORMS%

set PATH=%WL_HOME%\bin;%JAVA_HOME%\bin;%PATH%

:finish
```

Vedlegg III. Programkode grunnl. J2EE testmiljø

A. Build.cmd

```
@REM Adjust these variables to match your environment
if "" == "%JAVA_HOME%" set JAVA_HOME=\java
if "" == "%WL_HOME%" set WL_HOME=\weblogic
set MYSERVER=%WL_HOME%\myserver
set
MYCLASSPATH=%JAVA_HOME%\lib\classes.zip;%WL_HOME%\classes;%WL_HOME%\lib\weblogicaux.jar;%MYSERVER%\clientcla
sses

@REM Create the build directory, and copy the deployment descriptors into it
mkdir build build\META-INF
copy *.xml build\META-INF

@REM Compile ejb classes into the build directory (jar preparation)
javac -d build -classpath %MYCLASSPATH% Employee.java EmployeeHome.java ProcessingErrorException.java EmployeeBean.java
Hour.java HourHome.java HourBean.java

@REM Make a standard ejb jar file, including XML deployment descriptors
cd build
jar cv0f std_ejb_diplom.jar META-INF diplom
cd ..

@REM Run ejbc to create the deployable jar file
java -classpath %MYCLASSPATH% -Dweblogic.home=%WL_HOME% weblogic.ejbc -compiler javac build\std_ejb_diplom.jar
%MYSERVER%\ejb_diplom.jar

@REM Compile ejb interfaces & client app into the clientclasses directory
javac -d %MYSERVER%\clientclasses -classpath %MYCLASSPATH% Employee.java EmployeeHome.java Hour.java HourHome.java
ProcessingErrorException.java Client.java

@REM Compile servlets into the servletclasses directory
@REM javac -d %MYSERVER%\servletclasses -classpath %MYCLASSPATH% Servlet.java
```

B. ejb-jar.xml

```
<?xml version="1.0"?>

<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN' 'http://java.sun.com/j2ee/dtds/ejb-
jar_1_1.dtd'>

<ejb-jar>
   <enterprise-beans>
    <entity>
              <ejb-name>diplom</ejb-name>
              <home>diplom.EmployeeHome</home>
              <remote>diplom.Employee</remote>
              <ejb-class>diplom.EmployeeBean</ejb-class>
              <persistence-type>Bean</persistence-type>
              <prim-key-class>java.lang.String</prim-key-class>
              <reentrant>False</reentrant>

      <resource-ref>
       <res-ref-name>diplom1Pool</res-ref-name>
       <res-type>javax.sql.DataSource</res-type>
       <res-auth>Container</res-auth>
      </resource-ref>

    </entity>
    <entity>
              <ejb-name>diplom2</ejb-name>
              <home>diplom.HourHome</home>
              <remote>diplom.Hour</remote>
              <ejb-class>diplom.HourBean</ejb-class>
              <persistence-type>Bean</persistence-type>
              <prim-key-class>java.lang.Long</prim-key-class>
              <reentrant>False</reentrant>

      <resource-ref>
       <res-ref-name>diplom1Pool</res-ref-name>
```

```
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
     </resource-ref>

    </entity>
   </enterprise-beans>
   <assembly-descriptor>
    <container-transaction>
              <method>
                <ejb-name>diplom</ejb-name>
                <method-intf>Remote</method-intf>
                <method-name>*</method-name>
              </method>
              <trans-attribute>Required</trans-attribute>
    </container-transaction>
   </assembly-descriptor>
 </ejb-jar>
```

C. weblogic-ejb-jar.xml

```
<?xml version="1.0"?>

<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN'
'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'>

<weblogic-ejb-jar>
   <weblogic-enterprise-bean>
    <ejb-name>diplom</ejb-name>
    <caching-descriptor>
              <max-beans-in-cache>100</max-beans-in-cache>
    </caching-descriptor>
    <reference-descriptor>

      <resource-description>
       <res-ref-name>
        diplom1Pool
       </res-ref-name>
       <jndi-name>
        weblogic.jdbc.jts.diplom1Pool
       </jndi-name>
      </resource-description>

    </reference-descriptor>

    <jndi-name>diplom.EmployeeHome</jndi-name>
   </weblogic-enterprise-bean>

   <weblogic-enterprise-bean>
    <ejb-name>diplom2</ejb-name>
    <caching-descriptor>
              <max-beans-in-cache>100</max-beans-in-cache>
    </caching-descriptor>
    <reference-descriptor>

      <resource-description>
       <res-ref-name>
        diplom1Pool
       </res-ref-name>
       <jndi-name>
        weblogic.jdbc.jts.diplom1Pool
       </jndi-name>
      </resource-description>

    </reference-descriptor>

    <jndi-name>diplom.HourHome</jndi-name>
   </weblogic-enterprise-bean>
 </weblogic-ejb-jar>
```

D. EmployeeHome.java

```
package diplom;
```

```
import javax.ejb.CreateException;
import javax.ejb.EJBHome;
import javax.ejb.FinderException;
import java.rmi.RemoteException;
import java.util.Enumeration;

public interface EmployeeHome extends EJBHome {

  public Employee create(String emp_no, String emp_name)
    throws CreateException, RemoteException;

  public Employee findByPrimaryKey(String primaryKey)
    throws FinderException, RemoteException;

  public Enumeration findByNames(String name)
    throws FinderException, RemoteException;

}
```

E. Employee.java

```
package diplom;

import java.rmi.RemoteException;
import javax.ejb.EJBObject;

public interface Employee extends EJBObject {

  public String getName()
    throws RemoteException;

  public int getID()
    throws RemoteException;

}
```

F. EmployeeBean.java

```
package diplom;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Enumeration;
import java.util.Vector;

import javax.ejb.CreateException;
import javax.ejb.DuplicateKeyException;
import javax.ejb.EJBException;
import javax.ejb.EntityBean;
import javax.ejb.EntityContext;
import javax.ejb.FinderException;
import javax.ejb.NoSuchEntityException;
import javax.ejb.ObjectNotFoundException;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

public class EmployeeBean implements EntityBean {

  final static private boolean VERBOSE = true;

  private EntityContext ctx;
  private String emp_no; // also the primary Key
  private String emp_name;
  private int emp_id;

  public void setEntityContext(EntityContext ctx) {
    log("setEntityContext called");
```

```
  this.ctx = ctx;
}

public void unsetEntityContext() {
  log("unsetEntityContext (" + id() + ")");
  this.ctx = null;
}

public void ejbActivate() {
  log("ejbActivate (" + id() + ")");
}

public void ejbPassivate() {
  log("ejbPassivate (" + id() + ")");
}

public void ejbLoad() {
  log("ejbLoad: (" + id() +  ")");

  Connection con = null;
  PreparedStatement ps = null;
  emp_no = (String) ctx.getPrimaryKey();

  try {
    con = getConnection();
    ps  = con.prepareStatement("select emp_id, emp_name from tEmployee where emp_no = ?");
    ps.setString(1, emp_no);
    ps.executeQuery();
    ResultSet rs = ps.getResultSet();
    if (rs.next()) {
      emp_id = rs.getInt(1);
      emp_name = rs.getString(2);
    } else {
      String error = "ejbLoad: EmployeeBean (" + emp_no + ") not found";
      log(error);
      throw new NoSuchEntityException (error);
     }
  } catch (SQLException sqe) {
    log("SQLException:  " + sqe);
    throw new EJBException(sqe);
  } finally {
    cleanup(con, ps);
  }
}

public void ejbStore() {
  log("ejbStore (" + id() + ")");

  Connection con = null;
  PreparedStatement ps = null;

  try {
    con = getConnection();
    ps = con.prepareStatement("update tEmployee set emp_name = ? where emp_no = ?");
    ps.setString(1, emp_name);
    ps.setString(2, emp_no);
    if (!(ps.executeUpdate() > 0)) {
      String error = "ejbStore: EmployeeBean (" + emp_no + ") not updated";
      log(error);
      throw new NoSuchEntityException (error);
    }
  } catch(SQLException sqe) {
    log("SQLException:  " + sqe);
    throw new EJBException (sqe);
  } finally {
    cleanup(con, ps);
  }
}


public String ejbCreate(String emp_no, String emp_name)
  throws CreateException
{
  log("EmployeeBean.ejbCreate( id = " + emp_no + ", " + "emp_name = " + emp_name + ")");
  this.emp_no = emp_no;
  this.emp_name = emp_name;
```

```
  Connection con = null;
  PreparedStatement ps = null;
  PreparedStatement ps2 = null;

  try {
    con = getConnection();
    con.setAutoCommit(false);
    ps = con.prepareStatement("insert into tEmployee (emp_no, emp_name) values (?, ?)");
    ps.setString(1, emp_no);
    ps.setString(2, emp_name);
    if (ps.executeUpdate() != 1) {
      String error = "JDBC did not create any row";
      log(error);
      throw new CreateException (error);
    }

    ps2  = con.prepareStatement("select max(emp_id) from tEmployee");
    ps2.executeQuery();
    ResultSet rs = ps2.getResultSet();
    if (rs.next()) {
      emp_id = rs.getInt(1);
      con.commit();
    } else {
      con.rollback();
      String error = "ejbCreate: EmployeeBean (" + emp_no + ") not found";
      log(error);
      throw new NoSuchEntityException (error);
     }


    return emp_no;
  } catch (SQLException sqe) {
  try {
      ejbFindByPrimaryKey(emp_no);
    } catch(ObjectNotFoundException onfe) {
      String error = "SQLException: " + sqe;
      log(error);
      throw new CreateException (error);
    }
    String error = "An Employee already exists in the database with Primary Key " + emp_no;
    log(error);
    throw new DuplicateKeyException(error);
  } finally {
    cleanup(con, ps);
  }
}

public void ejbPostCreate(String emp_no, String emp_name) {
  log("ejbPostCreate (" + id() + ")");
}

public void ejbRemove() {
  log("ejbRemove (" + id() + ")");

  Connection con = null;
  PreparedStatement ps = null;

  try {
    con = getConnection();
    emp_no = (String) ctx.getPrimaryKey();
    ps = con.prepareStatement("delete from tEmployee where emp_no = ?");
    ps.setString(1, emp_no);
    if (!(ps.executeUpdate() > 0)) {
      String error = "EmployeeBean (" + emp_no + " not found";
      log(error);
      throw new NoSuchEntityException (error);
    }
  } catch (SQLException sqe) {
    log("SQLException:  " + sqe);
    throw new EJBException (sqe);
  } finally {
    cleanup(con, ps);
  }
}
```

```
public String ejbFindByPrimaryKey(String pk)
  throws ObjectNotFoundException
{
  log("ejbFindByPrimaryKey (" + pk + ")");

  Connection con = null;
  PreparedStatement ps = null;

  try {
    con = getConnection();
    ps  = con.prepareStatement("select emp_id, emp_name from tEmployee where emp_no = ?");
    ps.setString(1, pk);
    ps.executeQuery();
    ResultSet rs = ps.getResultSet();
    if (rs.next()) {
      emp_id = rs.getInt(1);
      emp_name = rs.getString(2);
      emp_no = pk;
    } else {
      String error = "ejbFindByPrimaryKey: EmployeeBean (" + pk + ") not found";
      log(error);
      throw new ObjectNotFoundException (error);
     }
  } catch (SQLException sqe) {
    log("SQLException:  " + sqe);
    throw new EJBException (sqe);
  } finally {
    cleanup(con, ps);
  }

  log("ejbFindByPrimaryKey (" + pk + ") found");
  return pk;
}


public Enumeration ejbFindByNames(String name) {
  log("ejbFindByNames (name like " + name + ")");
  Connection con = null;
  PreparedStatement ps = null;

  try {
    con = getConnection();
    ps = con.prepareStatement("select emp_no from tEmployee where emp_name like ?");
    ps.setString(1, name);
    ps.executeQuery();
    ResultSet rs = ps.getResultSet();
    Vector v = new Vector();
    String pk;
    while (rs.next()) {
      pk = rs.getString(1);
      v.addElement(pk);
    }
    return v.elements();
  } catch (SQLException sqe) {
    log("SQLException: " + sqe);
    throw new EJBException (sqe);
  } finally {
    cleanup(con, ps);
  }
}

public String getName() {
  return emp_name;
}

public int getID() {
  return emp_id;
}

private Connection getConnection()
  throws SQLException
{
  InitialContext initCtx = null;
  try {
    initCtx = new InitialContext();
    DataSource ds = (javax.sql.DataSource)
```

```
        initCtx.lookup("java:comp/env/diplom1Pool");
      return ds.getConnection();
    } catch(NamingException ne) {
      log("UNABLE to get a connection from diplom1Pool!");
      log("Please make sure that you have setup the connection pool properly");
      log("ERROR: " + ne);
      throw new EJBException(ne);
    } finally {
      try {
        if(initCtx != null) initCtx.close();
      } catch(NamingException ne) {
        log("Error closing context: " + ne);
        throw new EJBException(ne);
      }
    }
  }

  private void log(String s) {
    if (VERBOSE) System.out.println(s);
  }

  private String id() {
    return "PK = " + (String) ctx.getPrimaryKey();
  }

  private void cleanup(Connection con, PreparedStatement ps) {
    try {
      if (ps != null) ps.close();
    } catch (Exception e) {
      log("Error closing PreparedStatement: "+e);
      throw new EJBException (e);
    }

    try {
      if (con != null) con.close();
    } catch (Exception e) {
      log("Error closing Connection: " + e);
      throw new EJBException (e);

    }
  }
}
```

G. HourHome.java

```
package diplom;

import javax.ejb.CreateException;
import javax.ejb.EJBHome;
import javax.ejb.FinderException;
import java.rmi.RemoteException;
import java.util.Enumeration;
import java.sql.Date;

public interface HourHome extends EJBHome {

  public Hour create(Date hou_date, String hou_desc, double hou_hours, long pro_id, long emp_id)
    throws CreateException, RemoteException;

  public Hour findByPrimaryKey(Long primaryKey)
    throws FinderException, RemoteException;

  public Enumeration findByEmployee(long emp_id)
    throws FinderException, RemoteException;

}
```

H. Hour.java

```
package diplom;

import java.rmi.RemoteException;
import javax.ejb.EJBObject;
```

```
import java.sql.Date;

public interface Hour extends EJBObject {

  public long getID()
    throws RemoteException;

  public Date getDate()
    throws RemoteException;

  public String getDesc()
    throws RemoteException;

  public double getHours()
    throws RemoteException;

  public long getProject()
    throws RemoteException;

  public long getEmployee()
    throws RemoteException;

}
```

I. HourBean.java

```
package diplom;

import java.io.Serializable;
import java.sql.*;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Enumeration;
import java.util.Vector;

import javax.ejb.CreateException;
import javax.ejb.DuplicateKeyException;
import javax.ejb.EJBException;
import javax.ejb.EntityBean;
import javax.ejb.EntityContext;
import javax.ejb.FinderException;
import javax.ejb.NoSuchEntityException;
import javax.ejb.ObjectNotFoundException;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

public class HourBean implements EntityBean {

  final static private boolean VERBOSE = true;

  private EntityContext ctx;
  private long hou_id; // also the primary Key
  private long pro_id;
  private long emp_id;
  private double hou_hours;
  private Date hou_date;
  private String hou_desc;

  public void setEntityContext(EntityContext ctx) {
    log("setEntityContext called");
    this.ctx = ctx;
  }

  public void unsetEntityContext() {
    log("unsetEntityContext (" + id() + ")");
    this.ctx = null;
  }

  public void ejbActivate() {
    log("ejbActivate (" + id() + ")");
  }
```

```
 public void ejbPassivate() {
   log("ejbPassivate (" + id() + ")");
 }

 public void ejbLoad() {
   log("ejbLoad: (" + id() +  ")");

   Connection con = null;
   PreparedStatement ps = null;
   hou_id = ((Long) ctx.getPrimaryKey()).longValue();

   try {
     con = getConnection();
     ps  = con.prepareStatement("select emp_id, pro_id, hou_date, hou_hours, hou_desc from tHour where hou_id = ?");
     ps.setLong(1, hou_id );
     ps.executeQuery();
     ResultSet rs = ps.getResultSet();
     if (rs.next()) {
       emp_id = rs.getLong(1);
       pro_id = rs.getLong(2);
       hou_date = rs.getDate(3);
       hou_hours = rs.getLong(4);
       hou_desc = rs.getString(5);
     } else {
       String error = "ejbLoad: HourBean (" + hou_id + ") not found";
       log(error);
       throw new NoSuchEntityException (error);
      }
   } catch (SQLException sqe) {
     log("SQLException:  " + sqe);
     throw new EJBException(sqe);
   } finally {
     cleanup(con, ps);
   }
 }

 public void ejbStore() {
   log("ejbStore (" + id() + ")");

   Connection con = null;
   PreparedStatement ps = null;

   try {
     con = getConnection();
     ps = con.prepareStatement("update tHour set hou_date = ?, hou_desc = ?, hou_hours = ?, pro_id = ?, emp_id = ? where hou_id =
?");
     ps.setDate(1, hou_date);
     ps.setString(2, hou_desc);
     ps.setDouble(3, hou_hours );
     ps.setLong(4, pro_id );
     ps.setLong(5, emp_id );
     ps.setLong(6, hou_id);
     if (!(ps.executeUpdate() > 0)) {
       String error = "ejbStore: HourBean (" + hou_id + ") not updated";
       log(error);
       throw new NoSuchEntityException (error);
     }
   } catch(SQLException sqe) {
     log("SQLException:  " + sqe);
     throw new EJBException (sqe);
   } finally {
     cleanup(con, ps);
   }
 }


 public Long ejbCreate(Date hou_date, String hou_desc, double hou_hours, long pro_id, long emp_id)
   throws CreateException
 {
   log("HourBean.ejbCreate( Date = " + hou_date + ", " + "emp_id = " + emp_id + ")");
   this.hou_id = hou_id;
   this.hou_date = hou_date;
   this.hou_desc = hou_desc;
   this.hou_hours = hou_hours;
   this.pro_id = pro_id;
   this.emp_id = emp_id;
```

```
  Connection con = null;
  PreparedStatement ps = null;
  PreparedStatement ps2 = null;

  try {
    con = getConnection();
    con.setAutoCommit(false);
    ps = con.prepareStatement("insert into tHour (hou_date, hou_desc, hou_hours, pro_id, emp_id) values (?, ?, ?, ?, ?)");
    ps.setDate(1, hou_date);
    ps.setString(1, hou_desc);
    ps.setDouble(2, hou_hours);
    ps.setLong(3, pro_id);
    ps.setLong(4, emp_id);
    if (ps.executeUpdate() != 1) {
      String error = "JDBC did not create any row";
      log(error);
      throw new CreateException (error);
    }

    ps2  = con.prepareStatement("select max(hou_id) from tHour");
    ps2.executeQuery();
    ResultSet rs = ps2.getResultSet();
    if (rs.next()) {
      hou_id = rs.getLong(1);
      con.commit();
    } else {
      con.rollback();
      String error = "ejbCreate: HourBean (" + hou_id + ") not found";
      log(error);
      throw new NoSuchEntityException (error);
     }


    return new Long(hou_id);
  } catch (SQLException sqe) {
    try {
      ejbFindByPrimaryKey(new Long(hou_id));
    } catch(ObjectNotFoundException onfe) {
      String error = "SQLException: " + sqe;
      log(error);
      throw new CreateException (error);
    }
    String error = "A Hour already exists in the database with Primary Key " + hou_id;
    log(error);
    throw new DuplicateKeyException(error);
  } finally {
    cleanup(con, ps);
  }
}

public void ejbPostCreate(Date hou_date, String hou_desc, double hou_hours, long pro_id, long emp_id) {
  log("ejbPostCreate (" + id() + ")");
}

public void ejbRemove() {
  log("ejbRemove (" + id() + ")");
  // we need to get the primary key from the context because
  // it is possible to do a remove right after a find, and
  // ejbLoad may not have been called.

  Connection con = null;
  PreparedStatement ps = null;

  try {
    con = getConnection();
    hou_id = ((Long) ctx.getPrimaryKey()).longValue();
    ps = con.prepareStatement("delete from tHour where hou_id = ?");
    ps.setLong(1, hou_id);
    if (!(ps.executeUpdate() > 0)) {
      String error = "HourBean (" + hou_id + " not found";
      log(error);
      throw new NoSuchEntityException (error);
    }
  } catch (SQLException sqe) {
    log("SQLException:  " + sqe);
```

```
    throw new EJBException (sqe);
   } finally {
    cleanup(con, ps);
   }
  }

  public Long ejbFindByPrimaryKey(Long pk)
   throws ObjectNotFoundException
  {
   log("ejbFindByPrimaryKey (" + pk + ")");

   Connection con = null;
   PreparedStatement ps = null;

   try {
    con = getConnection();
    ps  = con.prepareStatement("select emp_id, pro_id, hou_date, hou_hours, hou_desc from tHour where hou_id = ?");
    ps.setLong(1, pk.longValue() );
    ps.executeQuery();
    ResultSet rs = ps.getResultSet();
    if (rs.next()) {
     this.hou_id = pk.longValue();
     this.emp_id = rs.getLong(1);
     this.pro_id = rs.getLong(2);
     this.hou_date = rs.getDate(3);
     this.hou_hours = rs.getLong(4);
     this.hou_desc = rs.getString(5);
    } else {
     String error = "ejbFindByPrimaryKey: HourBean (" + pk + ") not found";
     log(error);
     throw new ObjectNotFoundException (error);
    }
   } catch (SQLException sqe) {
    log("SQLException:  " + sqe);
    throw new EJBException (sqe);
   } finally {
    cleanup(con, ps);
   }

   log("ejbFindByPrimaryKey (" + pk + ") found");
   return pk;
  }


  public Enumeration ejbFindByEmployee(long emp_id) {
   log("ejbFindByEmployee (" + emp_id + ")");
   Connection con = null;
   PreparedStatement ps = null;

   try {
    con = getConnection();
    ps = con.prepareStatement("select hou_id from tHour where emp_id = ?");
    ps.setLong(1, emp_id);
    ps.executeQuery();
    ResultSet rs = ps.getResultSet();
    Vector v = new Vector();
    Long pk;
    while (rs.next()) {
     pk = new Long(rs.getLong(1));
     v.addElement(pk);
    }
    return v.elements();
   } catch (SQLException sqe) {
    log("SQLException: " + sqe);
    throw new EJBException (sqe);
   } finally {
    cleanup(con, ps);
   }
  }

  public long getID() {
   return hou_id;
  }

  public Date getDate() {
   return hou_date;
```

```
  }

  public String getDesc() {
    return hou_desc;
  }

  public double getHours() {
    return hou_hours;
  }

  public long getProject() {
    return pro_id;
  }

  public long getEmployee() {
    return emp_id;
  }

  private Connection getConnection()
    throws SQLException
  {
    InitialContext initCtx = null;
    try {
      initCtx = new InitialContext();
      DataSource ds = (javax.sql.DataSource)
        initCtx.lookup("java:comp/env/diplom1Pool");
      return ds.getConnection();
    } catch(NamingException ne) {
      log("UNABLE to get a connection from diplom1Pool!");
      log("Please make sure that you have setup the connection pool properly");
      throw new EJBException(ne);
    } finally {
      try {
        if(initCtx != null) initCtx.close();
      } catch(NamingException ne) {
        log("Error closing context: " + ne);
        throw new EJBException(ne);
      }
    }
  }

  private void log(String s) {
    if (VERBOSE) System.out.println(s);
  }

  private String id() {
    return "PK = "; //+ (Long) ctx.getPrimaryKey();
  }

  private void cleanup(Connection con, PreparedStatement ps) {
    try {
      if (ps != null) ps.close();
    } catch (Exception e) {
      log("Error closing PreparedStatement: "+e);
      throw new EJBException (e);
    }

    try {
      if (con != null) con.close();
    } catch (Exception e) {
      log("Error closing Connection: " + e);
      throw new EJBException (e);

    }
  }
}
```

Vedlegg IV. Programkode for grunnl. COM+ testmiljø

A. Diplom.Project class

```
Public Function GetProjects(emp_id As Integer) As String
  Dim sProjects As String
  Dim conn As New ADODB.Connection
  Dim rs As ADODB.Recordset
  Dim sSQL As String

  sSQL = "SELECT dbo.tProject.pro_no, dbo.tProject.pro_title " & _
       "FROM dbo.tParticipant INNER JOIN " & _
       "dbo.tProject ON dbo.tParticipant.pro_id = dbo.tProject.pro_id " & _
       "WHERE (dbo.tParticipant.emp_id = " & emp_id & ")"

  conn.Open "DIPLOM2", "diplom2", "diplom2"
  Set rs = conn.Execute(sSQL)

  While Not rs.EOF
    sProjects = sProjects & rs("pro_no").Value & ";" & rs("pro_title").Value & ";"

    rs.MoveNext
  Wend

  GetProjects = sProjects

End Function
```

B. Diplom.vbp

```
Type=OleDll
Reference=*\G{00020430-0000-0000-C000-000000000046}#2.0#0#C:\WINNT\System32\stdole2.tlb#OLE Automation
Reference=*\G{00000201-0000-0010-8000-00AA006D2EA4}#2.1#0#C:\Program Files\Common
Files\System\ADO\msado21.tlb#Microsoft ActiveX Data Objects 2.1 Library
Class=Project; Project.cls
Startup="(None)"
ExeName32="Diplom.dll"
Command32=""
Name="Diplom"
HelpContextID="0"
CompatibleMode="1"
MajorVer=1
MinorVer=0
RevisionVer=0
AutoIncrementVer=0
ServerSupportFiles=0
VersionCompanyName="HiA"
CompilationType=0
OptimizationType=0
FavorPentiumPro(tm)=0
CodeViewDebugInfo=0
NoAliasing=0
BoundsCheck=0
OverflowCheck=0
FlPointCheck=0
FDIVCheck=0
UnroundedFP=0
StartMode=1
Unattended=0
Retained=0
ThreadPerObject=0
MaxNumberOfThreads=1
ThreadingModel=1

[MS Transaction Server]
AutoRefresh=1
```

Vedlegg V. Programkode Java-COM JACOB test

A. Build.cmd

```
@REM Adjust these variables to match your environment
if "" == "%JAVA_HOME%" set JAVA_HOME=\java
if "" == "%WL_HOME%" set WL_HOME=\weblogic
set MYSERVER=%WL_HOME%\myserver
set
MYCLASSPATH=C:\java\jacob\jacob.jar;%JAVA_HOME%\lib\classes.zip;%WL_HOME%\classes;%WL_HOME%\lib\weblogicaux.jar;%
MYSERVER%\clientclasses

@REM Create the build directory, and copy the deployment descriptors into it
mkdir build build\META-INF build\images
copy *.xml build\META-INF
copy *.gif build\images

@REM Compile ejb classes into the build directory (jar preparation)
javac -d build -classpath %MYCLASSPATH% Project.java ProjectHome.java ProjectBean.java

@REM Make a standard ejb jar file, including XML deployment descriptors
cd build
jar cv0f std_ejb_project_jacob.jar META-INF diplom images
cd ..

@REM Run ejbc to create the deployable jar file
java -classpath %MYCLASSPATH% -Dweblogic.home=%WL_HOME% weblogic.ejbc -compiler javac build\std_ejb_project_jacob.jar
%MYSERVER%\ejb_basic_project_jacob.jar

@REM Compile ejb interfaces & client app into the clientclasses directory
javac -d %MYSERVER%\clientclasses -classpath %MYCLASSPATH% Project.java ProjectHome.java Client.java
```

B. ejb-jar.xml

```
<?xml version="1.0"?>

<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN' 'http://java.sun.com/j2ee/dtds/ejb-
jar_1_1.dtd'>

<ejb-jar>
   <small-icon>images/green-cube.gif</small-icon>
   <enterprise-beans>
    <session>
      <small-icon>images/orange-cube.gif</small-icon>
              <ejb-name>projectSession</ejb-name>
              <home>diplom.project.jacob.ProjectHome</home>
              <remote>diplom.project.jacob.Project</remote>
              <ejb-class>diplom.project.jacob.ProjectBean</ejb-class>
              <session-type>Stateless</session-type>
              <transaction-type>Container</transaction-type>
    </session>
   </enterprise-beans>
   <assembly-descriptor>
    <container-transaction>
              <method>
                <ejb-name>projectSession</ejb-name>
                <method-intf>Remote</method-intf>
                <method-name>*</method-name>
              </method>
              <trans-attribute>Required</trans-attribute>
    </container-transaction>
   </assembly-descriptor>
 </ejb-jar>
```

C. weblogic-ejb-jar.xml

```
<?xml version="1.0"?>

<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN'
'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'>
```

```
<weblogic-ejb-jar>
   <weblogic-enterprise-bean>
    <ejb-name>projectSession</ejb-name>
    <caching-descriptor>
              <max-beans-in-free-pool>100</max-beans-in-free-pool>
    </caching-descriptor>
    <jndi-name>diplom.jacob.ProjectHome</jndi-name>
   </weblogic-enterprise-bean>
 </weblogic-ejb-jar>
```

D. ProjectHome.java

```
package diplom.project.jacob;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface ProjectHome extends EJBHome {

  Project create() throws CreateException, RemoteException;
}
```

E. Project.java

```
package diplom.project.jacob;

import java.net.*;
import java.rmi.RemoteException;
import javax.ejb.EJBObject;

import com.jacob.com.*;
import com.jacob.activeX.*;

public interface Project extends EJBObject {


  public String getProjects (int emp_id)
    throws RemoteException;

}
```

F. ProjectBean.java

```
package diplom.project.jacob;

import java.net.*;
import java.util.*;
import javax.ejb.CreateException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import com.jacob.com.*;
import com.jacob.activeX.*;

public class ProjectBean implements SessionBean {

  private static final boolean VERBOSE = true;
  private SessionContext ctx;
  private int tradeLimit;

  private void log(String s) {
    if (VERBOSE) System.out.println(s);
  }

  public void ejbActivate() {
 //   log("ejbActivate called");
  }
```

```
  public void ejbRemove() {
//    log("ejbRemove called");
  }

  public void ejbPassivate() {
    log("ejbPassivate called");
  }

  public void setSessionContext(SessionContext ctx) {
    log("setSessionContext called");
    this.ctx = ctx;
  }

  public void ejbCreate () throws CreateException {
//    log("ejbCreate called");

    try {
      InitialContext ic = new InitialContext();

      Integer tl = (Integer) ic.lookup("java:/comp/env/tradeLimit");

      tradeLimit = tl.intValue();
    } catch (NamingException ne) {
      throw new CreateException("Failed to find environment value "+ne);
    }
  }

  public String getProjects (int emp_id)
    {
      Object sControl = null;
      String lang = "VBScript";
                                String expr = "1";
                                //int empid = 1;

                //log("getProjects: Creating ActiveXComponent");

                                ActiveXComponent sC = new ActiveXComponent("Diplom.Project");
                //log("getProjects: Created ActiveXComponent");

                                sControl = sC.getObject();
                //log("getProjects: Got the handle to the object");

                Variant result = Dispatch.call(sControl, "GetProjects", expr);
                //log("getProjects: Variant returned" + result );

      return new String( result.toString() );

  }

}
```

G. Client.java

```
package diplom.project.jacob;

import java.rmi.RemoteException;
import java.util.Properties;
import java.util.*;
import javax.ejb.CreateException;
import javax.ejb.RemoveException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;

public class Client {

  private static final String JNDI_NAME = "diplom.jacob.ProjectHome";

  private String url;
  private String user;
  private String password;
  private ProjectHome home;
```

```
public Client(String url, String user, String password)
  throws NamingException
{

  this.url       = url;
  this.user      = user;
  this.password  = password;

  home = lookupHome();
}

public static void main(String[] args) throws Exception {

  log("\nBeginning diplom.project.soap.Client...\n");

  String url       = "t3://localhost:7001";
  String user      = null;
  String password  = null;

  // Parse the command-line parameters
  switch(args.length) {
  case 3:
    password = args[2];
    // fall through
  case 2:
    user     = args[1];
    // fall through
  case 1:
    url      = args[0];
    break;
  }

  Client client = null;
  try {
    client = new Client(url, user, password);
  } catch (NamingException ne) {
    System.exit(1);
  }

  try {
    client.example();
  } catch (Exception e) {
    log("There was an exception while creating and using the Project.");
    log("This indicates that there was a problem communicating with the server: "+e);
  }

  log("\nEnd diplom.project.soap.Client...\n");
}

public void example()
  throws CreateException, RemoteException, RemoveException
{

  // create a Project
  log("Creating a project");
  Project project = (Project) narrow(home.create(), Project.class);
  log("Created a project");

  Calendar timingStart = Calendar.getInstance();

  // execute some calls
  for (int i=0; i<1000; i++) {
    //log("SOAP: " + project.getProjects( 1 ) );
      project.getProjects( 1 );
  }

  Calendar timingEnd = Calendar.getInstance();

  long timing;
  timing = timingEnd.getTime().getTime() - timingStart.getTime().getTime();

  log("Timing: " + timing );

  // remove the Project
  log("Removing the project");
  project.remove();
```

```
  }

  private Object narrow(Object ref, Class c) {
    return PortableRemoteObject.narrow(ref, c);
  }

  private ProjectHome lookupHome()
    throws NamingException
  {
    // Lookup the beans home using JNDI
    Context ctx = getInitialContext();

    try {
      Object home = ctx.lookup(JNDI_NAME);
      return (ProjectHome) narrow(home, ProjectHome.class);
    } catch (NamingException ne) {
      log("The client was unable to lookup the EJBHome.  Please make sure ");
      log("that you have deployed the ejb with the JNDI name "+JNDI_NAME+" on the WebLogic server at "+url);
      throw ne;
    }
  }

  private Context getInitialContext() throws NamingException {

    try {
      // Get an InitialContext
      Properties h = new Properties();
      h.put(Context.INITIAL_CONTEXT_FACTORY,
        "weblogic.jndi.WLInitialContextFactory");
      h.put(Context.PROVIDER_URL, url);
      if (user != null) {
        log ("user: " + user);
        h.put(Context.SECURITY_PRINCIPAL, user);
        if (password == null)
          password = "";
        h.put(Context.SECURITY_CREDENTIALS, password);
      }
      return new InitialContext(h);
    } catch (NamingException ne) {
      log("We were unable to get a connection to the WebLogic server at "+url);
      log("Please make sure that the server is running.");
      throw ne;
    }
  }

  private static void log(String s) {
    System.out.println(s);
  }

}
```

H. go.cmd

```
java  diplom.project.jacob.Client
```

Vedlegg VI. SQL Skript for å lage tabeller til testmiljø

A. Diplom1 database tabeller

```
CREATE TABLE [dbo].[tEmployee] (
	[emp_id] [int] IDENTITY (1, 1) NOT NULL ,
	[emp_no] [varchar] (10) COLLATE Danish_Norwegian_CI_AS NOT NULL ,
	[emp_fname] [varchar] (50) COLLATE Danish_Norwegian_CI_AS NULL ,
	[emp_mname] [varchar] (50) COLLATE Danish_Norwegian_CI_AS NULL ,
	[emp_lname] [varchar] (50) COLLATE Danish_Norwegian_CI_AS NULL ,
	[emp_email] [varchar] (50) COLLATE Danish_Norwegian_CI_AS NULL ,
	[emp_phone] [varchar] (20) COLLATE Danish_Norwegian_CI_AS NULL ,
	[emp_name] [varchar] (50) COLLATE Danish_Norwegian_CI_AS NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[tHour] (
	[hou_id] [int] IDENTITY (1, 1) NOT NULL ,
	[hou_date] [datetime] NULL ,
	[hou_hours] [decimal](18, 0) NULL ,
	[pro_id] [int] NULL ,
	[emp_id] [int] NULL ,
	[hou_desc] [varchar] (50) COLLATE Danish_Norwegian_CI_AS NULL
) ON [PRIMARY]
GO
```

B. Diplom2 database tabeller

```
CREATE TABLE [dbo].[tProject] (
	[pro_id] [int] IDENTITY (1, 1) NOT NULL ,
	[pro_no] [varchar] (10) COLLATE Danish_Norwegian_CI_AS NULL ,
	[pro_title] [varchar] (50) COLLATE Danish_Norwegian_CI_AS NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[tParticipant] (
	[par_id] [int] IDENTITY (1, 1) NOT NULL ,
	[emp_id] [int] NULL ,
	[pro_id] [int] NULL
) ON [PRIMARY]
GO
```