FFI-rapport 2013/00441

Enterprise Service Bus – definisjon og bruksområder

Ketil Lund, Trude H. Bloebaum og Frank T. Johnsen

FFI Forsvarets forskningsinstitutt

Enterprise Service Bus – definisjon og bruksområder

Ketil Lund, Trude H. Bloebaum og Frank T. Johnsen

Forsvarets forskningsinstitutt (FFI)

3. juni 2013

FFI-rapport 2013/00441

1176

P: ISBN 978-82-464-2272-5
E: ISBN 978-82-464-2273-2

Emneord

Nettverksbasert forsvar

Enterprise Service Bus

Tjenesteorientert arkitektur

Godkjent av

Rolf Rasmussen	Prosjektleder
Anders Eggen	Avdelingssjef

Sammendrag

I denne rapporten diskuterer vi begrepet *Enterprise Service Bus* (ESB), og definerer det som *en programvarearkitektur som muliggjør applikasjonsintegrasjon ved bruk av meldingsruting*. Et typisk ESB-produkt har mange ulike funksjoner, bl.a. det å overvåke og kontrollere ruting av meldinger mellom tjenester, distribusjon/versjonkontroll av tjenester og ofte en del annen funksjonalitet også (f.eks. protokollkonvertering, sikkerhet). Vi tar for oss en produktportefølje, WSO2, og ser på funksjonaliteten den tilbyr og hvordan den kanskje kunne vært benyttet av Forsvaret. Produktet ble valgt som eksempel fordi det er lett tilgjengelig, ikke fordi det nødvendigvis er å foretrekke over andre produkter på markedet. Hvilket produkt Forsvaret bør velge gir ikke denne rapporten noen anbefalinger om, snarere peker den på ønsket funksjonalitet og standarder som man bør legge til grunn når man velger et produkt i framtiden.

English summary

This report defines the term *Enterprise Service Bus* (ESB) as *a software architecture facilitating application integration using message routing*. A typical ESB product supports many different functions, such as monitoring and controlling message routing between services, distribution/version control of services, and often a lot of other functionality as well (for example protocol conversion, security). We investigate one specific product, WSO2, and discuss the functionality it offers and how it potentially could be used. This product was chosen as an example because it is easy to obtain, and not because it necessarily is better than other products on the market. This report does not suggest which ESB to choose, rather it discusses the desired functionality and standards support that should be considered when choosing a product in the future.

Innhold

1 Innledning

Formålet med rapporten er å introdusere en produktgruppe kalt *Enterprise Service Bus* (ESB), og diskutere denne i kontekst av tjenesteorientert arkitektur (*Service-Oriented Architecture* (SOA)) og Nettverksbasert Forsvar. I rapporten drøftes ulike aspekter ved ESBer, og vi søker å bringe klarhet rundt tre sentrale spørsmål:

1. Hva er en ESB (hva er den laget for å gjøre)?
2. Hvorfor er ESBer av interesse for Forsvaret?
3. Hvordan bør Forsvaret benytte ESBer?

Begrepet ESB ble introdusert av Sonic Software i 2002, som beskrivelse av en plattform for Enterprise Application Integration (EAI). Siden den tid har produktkategorien modnet vesentlig, og ESBer er i dag en viktig komponent innen tjenesteorienterte arkitekturer. Vi ser imidlertid at bruken av begrepet ESB etter hvert har blitt noe utvannet, og det har blitt vanskelig å gi en nøyaktig definisjon av hva det er. Det er likevel mulig å identifisere en del grunnleggende funksjonalitet, og i bunn og grunn er en ESB en mellomvareløsning basert på tjenesteorientering, som gjør det mulig å integrere heterogene systemer ved å modellere applikasjonenes endepunkter som tjenester [18].

Ved FFI har vi primært fokusert på SOA på taktisk nivå. På dette nivået, med lave båndbredder og begrensede maskinressurser, har bruk av ESBer så langt ikke vært et tema, da eksisterende produkter ikke er beregnet for slike forhold. I prosjektet har vi derfor gjort lite praktisk eksperimentering med ESBer[1], og denne rapporten bygger derfor i stor grad på anbefalinger og råd fra andre instanser, bl.a. NC3A [9], Gartner [26] og Burton [18]. Vi fokuserer på de verktøyene som vanligvis inngår i en ESB, og hvordan disse kan brukes.

Samtidig har vi i prosjektet inngående kjennskap til arbeidet med NATOs SOA Baseline [7], som spesifiserer standarder og profiler som bør følges av alle systemer og applikasjoner som skal inngå i Forsvarets informasjonsinfrastruktur. Ettersom ESBer vil inngå i den samme infrastrukturen er det svært viktig at også disse følger retningslinjene som gis i SOA Baseline.

Denne rapporten er organisert som følger: Kapittel 2 diskuterer hva en ESB egentlig er, og hvilke oppgaver den er tiltenkt å løse. Kapittel 3 drøfter ulik funksjonalitet som kan støttes av en ESB, og tar for seg både påkrevde og valgfrie funksjonsmoduler. Kapittel 4 presenterer en case study av ett bestemt ESB-produkt. Vi valgte å se nærmere på denne ESBen fordi den er fritt tilgjengelig, men det er ikke dermed gitt at den bør velges som del av en informasjonsinfrastruktur, noe som poengteres i dette kapittelet. Kapittel 5 omtaler hvordan man kan tenke seg å benytte en/flere ESBer i Forsvarets informasjonsinfrastruktur. Rapportens hovedpunkter oppsummeres i konklusjonen i Kapittel 6.

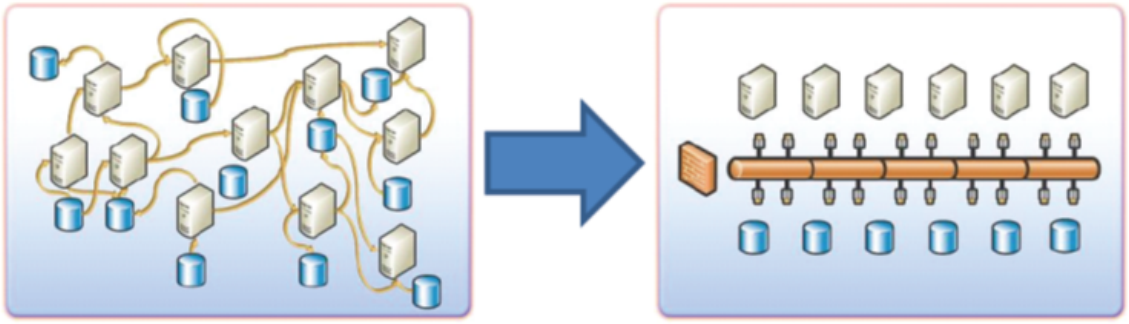
[1] Noe testing har vært gjort gjennom studentarbeid. Dette arbeidet er oppsummert i Appendiks A.

2 Definisjon og bakgrunn

ESBer ble opprettet for å møte krav og å unngå vanlige problemer som andre Enterprise Application Integration (EAI) plattformer ikke kunne løse. Den populære "hub and spoke" EAI-plattformen, der alle integrerte programmer fungerer gjennom en enkelt meldingsbroker, skaper et "single point of failure", noe som utgjør en risiko i et distribuert system. En ESB, på den annen side, kan ha flere brokere og dermed redusere denne risikoen.

En ESB er også mer egnet for å implementere SOA enn tradisjonelle EAI-løsninger. I motsetning til "hub and spoke"-plattformer tilbyr en ESB løs kopling av systemer og bruk av åpne standarder, to aspekter som kjennetegner de fleste vellykkede SOA-implementeringer. Sikkerhet, smidighet, tilgjengelighet, og andre viktige kjennetegn ved EAI er forbedret ytterligere med dagens ESBer.

Mens en ESB kan hjelpe, er det samtidig viktig å velge riktig produkt. En lettvekts-ESB kan implementeres raskt og billig, og er ofte foretrukket av utviklere som arbeider etter strenge tidsfrister og stramme budsjetter. Hvilken ESB-løsning man velger må være en avveining mellom kompleksitet og ønsket funksjonalitet. Man må dekke de umiddelbare behov, men samtidig kan man ha en langsiktig visjon og et ønske om en mer altomfattende ESB for å støtte en bred SOA-strategi.



Figur 2.1 En ESB er et produkt som tilbyr en meldingsplattform: Snarere enn å kople alle tjenester og klienter direkte (venstre), kan man kople tjenester og klienter til en felles meldingsbuss (høyre).

En ESB er en programvarearkitektur som benyttes for å implementere samhandling og kommunikasjon for gjensidig samspill mellom ressurser (se Figur 2.1). Med andre ord, det er en plattform som kan brukes for å realisere en tjenesteorientert arkitektur. Som en arkitekturmodell for programvare for distribuert databehandling fremmer den smidighet og fleksibilitet med hensyn på kommunikasjon og samhandling mellom applikasjoner. En ESBs primære bruk er i integrasjon av heterogene og komplekse systemer, og ofte ser man at bedrifter velger å kjøpe en ESB for raskt å deployere en tjenesteorientert arkitektur. De fleste ESBer benytter åpne standarder (en ESB er ofte et knippe Web services-standarder som er implementert og pakket i et produkt, selv om dette ikke er noe krav i seg selv), og for interoperabilitet mot andre bedrifter kan man ofte benytte standard Web services for å kople sin ESB mot Web services i andre bedrifter. Det er imidlertid viktig å være klar over at ulike produkter ikke nødvendigvis støtter den samme funksjonaliteten og de samme standardene, og dette

må tas med i betraktningen ved valg av ESB.

Det er også viktig å innse at bruk av en ESB ikke er noen forutsetning for å lage en tjenesteorientert arkitektur, og det å ta i bruk en ESB er heller ikke noen automatisk garanti for at man oppnår en tjenesteorientert arkitektur. I stedet er en ESB en verktøykasse, og hvilke verktøy den inneholder varierer fra produkt til produkt. Dersom man ønsker å benytte en ESB i sin tjenesteorienterte arkitektur, så er det derfor viktig å velge et produkt som inneholder de verktøyene man faktisk trenger. I tillegg bør man så langt mulig unngå å låse seg for mye til én leverandør (vendor lock-in).

En ESB er først og fremst et komplett mellomvareprodukt, og dette produktet kan være bygget opp på ulike måter. For eksempel kan det godt benyttes annen teknologi enn Web services internt i produktet – blant annet er flere basert på ulike implementasjoner av meldingskøer. I prinsippet kan man tenke seg en ESB som en "svart boks" – du trenger ikke å kjenne til hva som finnes i boksen, i stedet bruker du et kjent grensesnitt for å kople dine tjenester og klienter til denne boksen.

3 Funksjonalitet

Generelt kan man si at ESB er en programvarearkitektur som muliggjør applikasjonsintegrasjon. En ESB er verdifull for innføringen av en tjenesteorientert arkitektur fordi den ruter meldinger, utfører transaksjoner, orkestrerer tjenester og støtter abonnementstjenester mellom ulike og distribuerte applikasjoner.

En ESB kan ha mange ulike oppgaver, der de vanligste er:

1. Håndtere ruting av meldinger mellom tjenester.
2. Distribusjon og versjonskontroll av tjenester.
3. Støtte tjenester som hendelseshåndtering, datatransformasjon og kartlegging, meldings- og hendelseskøer, sikkerhets- og unntakshåndtering, protokollkonvertering, samt å håndheve tjenestekvalitet (QoS).

Punkt 1 ovenfor, meldingsruting, regnes som en meget sentral funksjon som støttes av alle produkter som kan påføres merkelappen "ESB". Punkt 2, distribusjon og versjonskontroll av tjenester, befatter seg med aspekter av tjenesteforvaltning og eventuell integrasjon av en applikasjonsserver. Det er dette, i kombinasjon med punkt 3, som gjør at man oppnår "full" utnyttelse av ESBen, ved at man abstraherer vekk alt som har med kommunikasjon og meldingsformater å gjøre, og i stedet virtualiserer tjenestene. Samtidig vil vi understreke at vi ikke anbefaler Forsvaret å benytte seg av så tett integrasjon mot et produkt, da det gir en stor grad av vendor lock-in, og at det kan være uheldig i forvaltningen av et nettverksbasert forsvar.

ESBer fungerer, som nevnt over, som en felles mekanisme som legger til rette for kommunikasjon mellom de ulike entitetene som deltar i en tjenesteorientert infrastruktur. Det finnes en rekke leverandører av mellomvareprodukter som har ESBer i sin produktportefølje, men det er store forskjeller i hva slags funksjonalitet som tilbys av de ulike produktene. Dette kapittelet omtaler et sett med

funksjonalitet som ofte forekommer i de tilgjengelig produktene, men det er viktig å legge merke til at de enkelte produktene ikke nødvendigvis har støtte for all denne funksjonaliteten. De aller fleste produkter som selges under ESB-merkelappen vil ha støtte for meldingshåndtering (se seksjon 3.1) og mediasjon (se seksjon 3.2), mens det er større forskjeller når det kommer til mer avansert funksjonalitet.

3.1 Meldingshåndtering

Den mest grunnleggende funksjonaliteten i en ESB er meldingshåndteringen. Ved bruk av en ESB vil meldinger som sendes mellom to entiteter sendes via ESBen i stedet for direkte mellom entitetene. ESBen tar da ansvar for å rute meldingene fram til riktig mottaker. På denne måten blir koplingen mellom en ressurs og en ressursbruker løsere enn ved direkte kommunikasjon.

3.2 Mediasjon

Mediasjon er viktig i en SOA for å muliggjøre informasjonsflyt mellom tjenester som benytter ulike dataformater. For tjenester realisert ved Web services er det standarden Extensible Stylesheet Language Transformations (XSLT) som ligger til grunn for mediasjon. Denne standarden er også utpekt i NATOs SOA Baseline, så man bør ved anskaffelse av en ESB fortrinnsvis undersøke om den støtter XSLT.

3.3 Tjenesteoppdagelse

Tjenesteoppdagelse er et viktig aspekt i SOA fordi det muliggjør løs kopling ved at klienter og tjenester ikke trenger å kjenne til hverandre på forhånd – det er tilstrekkelig at de kan finne hverandre via en mekanisme for tjenesteoppdagelse. Mange ESBer støtter tjenesteoppdagelse, ofte med proprietære løsninger. At en ESB inneholder proprietære teknologier er ikke nødvendigvis et problem, så lenge det er mulig å benytte et standardisert grensesnitt utad. Det finnes i dag tre standarder for tjenesteoppdagelse for Web services:

- Universal Description, Discovery and Integration (UDDI),
- electronic business using eXtensible Markup Language (ebXML), og
- Web Services Dynamic Discovery (WS-Discovery).

I SOA Baseline er UDDI valgt som tjenesteregister mens ebXML er valgt som metadataregister. WS-Discovery nevnes ikke i SOA Baseline, men har i den senere tid blitt møtt med økende interesse i NATO. Den har bla annet vært benyttet med hell i flere eksperimenter i senere tid (Combined Endeavor 2009, CoNSIS 2012, CWIX 2012, CWIX 2013). Vi vil derfor ikke foreslå at man velger ESB basert på støtte for en bestemt tjenesteoppdagelsesstandard, ettersom det er uvisst hvilken av disse som det blir mest moment bak i framtiden og alle fortsatt er akuelle for NATO pr i dag.

3.4 Tjenesteforvaltning

Tjenesteorienterte arkitekturer baserer seg på et sett med grunnleggende prinsipper som, i tillegg til å muliggjøre oppdaging og sammenkopling av entiteter, også hjelper til med å sikre at en tjenesteorientert *infrastruktur* kan kobles sammen med andre infrastrukturer som er basert på de samme prinsippene. Sentralt blant disse prinsippene finner man konseptet *løs kobling*, som betyr at de ulike entitetene og ressursene som til sammen utgjør infrastrukturen kan utvikles og vedlikeholdes uavhengig av hverandre. Ved å basere kommunikasjonen mellom disse entitetene på åpne standarder kan man sørge for at det allikevel er mulig å enkelt oppnå interoperabilitet mellom ressurser og ressursbrukere.

Denne tankegangen om løs kobling medfører at tjenestene i infrastrukturen i utgangspunket er autonome, og at infomasjon flyter fritt mellom entiteter uten at det er noen sentral styring av denne infomasjonsflyten. I mange tilfeller passer en slik fri flyt av informasjon dårlig sammen med behovet for informasjonsstyring og kontroll. Når kritiske forretningsprosesser er avhengige av at tjenester er tilgjengelig og har en høy ytelssesgrad oppstår det et behov for å kunne overvåke ytelse, oppdage feilsituasjoner og foreta automatiske omprioriteringer av ressurser. Dette er funksjonalitet som man kan finne i mange ESBer.

3.5 Sikkerhetsfunksjonalitet

Tradisjonelle sikkerhetsaspekter som *konfidensialitet*, *integritet*, og *tilgjengelighet* er naturligvis også viktige i systemer som baserer seg på en ESB. I dette tilfellet vil håndtering av konfidensialitet innebære at man vil sikre at kun rette vedkommende får tilgang til informasjon i ESBen, dvs. kun klienter med den rette autorisasjonen skal få tilgang til gitte tjenester. Videre vil hensynet til integritet innebære at meldingene som utveksles mellom tjenester og klienter ikke skal kunne endres av uvedkommende (eller dersom de blir endret, at man oppdager at dette har skjedd). For å oppnå konfidensialitet og integritet i en tjenesteorientert arkitektur som er basert på Web services finnes det en mengde standarder som tilbyr digitale signaturer, kryptering, autorisasjon- og identitetskontroll, osv. For en grundig innføring i standarder relatert til sikkerheten i Web services, se [20]. Typisk vil en ESB implementere støtte for et subset av disse standardene.

Tilgjengelighet innebærer at en tjeneste skal være tilgjengelig når det er behov for den, samt at man skal få tilbud om et akseptabelt tjenestekvalitetsnivå. Dette aspektet omfattes ikke i nevneverdig grad av dagens Web services standarder, så man må her se mer mot nettverksmekansimer for prioritetshåndtering, lastbalansering i lag-7-svitsj (også kjent som content switch), lastbalansering av applikasjonstjenere, osv. Vi ser likevel at enkelte ESBer tilbyr funksjonalitet som dekker aspekter ved tilgjengelighetsbehovet, slik som pålitelig meldingsruting, prioritering av meldinger internt i ESBen, osv.

Det er verdt å merke seg at sikkerhetsfunksjonaliteten vil være helt og holdent produktspesifikk, ettersom det ikke finnes noen standard for ESB-sikkerhet. Man er derfor henvist til å undersøke hvilke teknikker og standarder de produktene man ønsker å benytte støtter, og eventuelt supplere

med tilleggsmoduler slik som f.eks. en XML firewall for ytterligere sikkerhetsfunksjonalitet.

4 Case study

Dette kapittelet tar for seg en fritt tilgjengelig ESB og ser på noen av dens muligheter og begrensninger for å synliggjøre potensialet som ligger i det å benytte et slikt produkt. Denne ESBen er valgt kun fordi den er lett tilgjengelig og kan benyttes fritt. Vi stiller oss på ingen måte bak dette produktet, eller ønsker å fremheve det framfor andre tilgjengelige løsninger. Dersom man skal ta ibruk en ESB er det viktig å gjøre dette på riktig måte, dvs. *først kartlegge bedriftens behov*, før man basert på behovsanalysen *så anskaffer en ESB som best oppfyller de kravene man har*.

4.1 Introduksjon

Det finnes mange ulike ESB-løsninger. Enkelte er kommersielle (se Tabell 4.1) og selges som produkter med alt det innebærer (brukerstøtte osv.), mens andre er gratis å laste ned og ta i bruk (gis ut som open source (se Tabell 4.2)). Noen gratisløsninger har også en kommersiell versjon som gir flere garantier overfor brukeren, samt at det tilbys brukerstøtte. WSO2, som vi skal se nærmere på i dette kapittelet, er et eksempel på sistnevnte.

Firma	**Produkt**
Adea Solutions	Adea ESB Framework
Cordys	Cordys ESB
Fiorano Software Inc.	Fiorano ESB™ 2006
IBM	IBM® WebSphere® Enterprise Service Bus
Intersystems	Intersystems Ensemble
iWay Software	iWay Adaptive Framework for SOA
Microsoft	.NET Platform Microsoft BizTalk Server
Oracle	Oracle Integration products
PolarLake	Integration Suite
Software AG	EntireX
Tibco	ActiveMatrix
Progress	Sonic Software: Sonic ESB
Workday	Integration On-Demand

Tabell 4.1 Noen kommersielle ESBer

4.2 WSO2 case study

WSO2 er et globalt firma som ble grunnlagt i 2005 og i dag har kontorer i USA, UK, og Sri Lanka. WSO2 tilbyr en komplett open source mellomvareplattform bestående av mange ulike produkter. Disse produktene er modulære og det er valgfritt hvilke moduler og hvor mange av dem man tar i bruk. På denne måten kan man skreddersy produktporteføljen til best å dekke de behovene man har

Open source
Synapse ESB
ServiceMix ESB
Blackbird ESB
Chainbuilder ESB
ESB.NET
Jboss ESB
Mule ESB
OpenAdapter
OpenESB
OpenEAI
WSO2 ESB (basert på Apache Synapse)

Tabell 4.2 Noen fritt tilgjengelige ESBer

(se Tabell 4.3 for en oversikt). Ett av disse mellomvareproduktene er den såkalte *WSO2 ESB* som vi skal se nærmere på her.

WSO2 ESB er en mediasjonsplattform som er basert på (og forbedrer) Apache Synapse ESB, og tilbyr støtte for tjenesteorienterte systemer med høy belastning. For eksempel kan det nevnes at Ebay bruker denne ESBen i sine systemer (see Figur 4.1 for flere eksempler på firmaer som benytter WSO2 ESB). Ettersom ESBen er bygget på et Apache-initiativ så er den open source, utgitt med Apache License 2.0. Produktet er fritt tilgjengelig[2] og det er relativt lett å komme igang med[3].

WSO2s produktportefølje støtter en lang rekke standarder: SOAP 1.1, SOAP 1.2, MTOM, WSDL 1.1, WSDL 2.0, WS-Addressing, WS-Security, WS-Trust, WS-SecureConversation, WS-Policy, WS-PolicyAttachment, WS-SecurityPolicy, WS-ReliableMessaging, WS-Discovery, SAML, XACML, WS-BPEL, mm. Videre er det støtte for ulike transportlag slik som HTTP, HTTPS, JMS, mm. Det er likevel verdt å merke seg at ikke noe produkt støtter absolutt alle tilgjengelige standarder. For eksempel har WSO2 valgt å støtte WS-Eventing for å tilby publish/subscribe-paradigmet, mens den konkurrerende og mer omfattende spesifikasjonen WS-Notification, som er valgt for NATOs SOA baseline, ikke er implementert. Dette betyr at det er viktig å se nøye på hva som tilbys før man velger hvilket produkt man skal bruke, men det er slett ikke sikkert at man finner fram til ett produkt som kan løse alle behov. Et godt eksempel på dette er den eksperimentelle informasjonsinfrasturkturen som NC3A benyttet i forbindelse med en kombinert demonstrasjon og eksperiment i forbindelse med arbeidet i NATO RTO IST-090 (se Figur 4.2). Der benyttet man WSO2 ESB til alle request/response-tjenester, mens ServiceMix ESB ble benyttet til publish/subscribe [14]. Grunnen til dette er at ServiceMix støtter WS-Notification, som NATO har utpekt som standarden å bruke for abonnementstjenester. Naturligvis støtter ServiceMix også request/response, så det

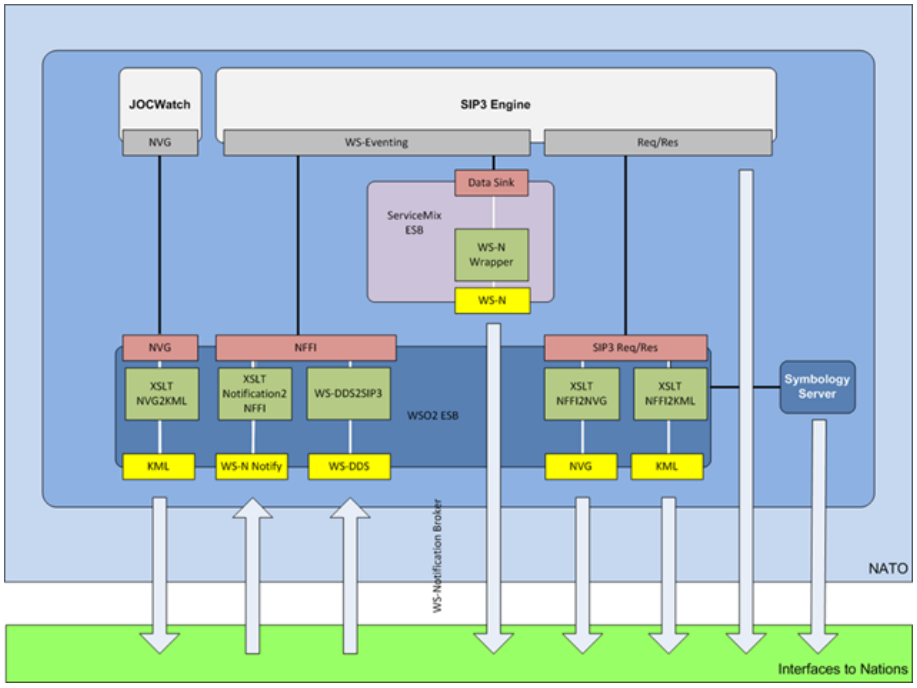
[2]WSO2 ESB kan lastes ned fra: `http://wso2.com/products/enterprise-service-bus/`

[3]Installasjonsguide er tilgjengelig her: `http://docs.wso2.org/wiki/display/ESB451/Installation+Guide`

Mellomvarekomponent
Carbon (en modul med basiskapabiliteter delt av de øvrige WSO2-komponentene)
API manager
Application server (støtte for Web services og REST)
Business activity monitor
Business process server (business-prosesser med WS-BPEL-standarden)
Business rules server
Complex event processor
Data services server (for å eksponere data som en Web service eller med REST)
Elastic load balancer
Enterprise service bus
Gadget server
Governance registry (service discovery, WS-Discovery)
Identity server (sikkerhetsstøtte, bl.a. WS-Security, SAML, etc.)
Mashup server
Message broker

Tabell 4.3 WSO2 produktportefølje – open source og basert på åpne standarder

Figur 4.1 Eksempler på bedrifter som benytter WSO2 ESB



Figur 4.2 En informasjonsinfrastruktur bestående av WSO2 og ServiceMix ESBer (fra [14])

er mulig at man her kunne klart seg med kun en ESB. På den annen side er WSO2 velprøvd som ESB-løsning i systemer med høy belastning, så det kan ha vært en avgjørende faktor for at den ble benyttet i størstedelen av systemet. Desto flere ulike programvareløsninger man må innlemme i et produksjonssystem, jo større blir kompleksiteten, driftskostnadene og mulighetene for feilkonfigurering. Det kan derfor være tids- og kostnadsbesparende å foreta en grundig behovsanalyse før man bestemmer seg for hvilke(t) produkt(er) man vil benytte.

4.3 Forskning

Vi har foretatt noen eksperimenter som involverer WSO2 ESB. Bl.a. har FFI vært involvert i det tidligere nevnte IST-090-eksperimentet, der WSO2 ESB inngikk som en av to ESBer i infrastrukturen, samt at FFI har veiledet en studentoppgave som tok for seg WSO2 ESB som en komponent for å oppnå rollebasert tjenestekvalitet (se [13] for ytterligere detaljer om tjenestekvalitet for Web services). Studentarbeidet er oppsummert i en artikkel (se Appendiks A) publisert på IEEE HeterWMN 2012, Anaheim, CA, USA 3. desember 2012.

4.4 Oppsummering

En ESB tilbyr en mellomvareplattform som ofte benyttes for å implementere en tjenesteorientert arkitektur. I dette kapittelet har vi sett på ett spesifikt produkt, nemlig WSO2 ESB. Denne ESBen (og de tilhørende modulene) støtter en god del av Web services-standardene, og også REST-paradigmet. Mediasjon, sikkerhet, og forretningsprosesser er også støttet, samt kommunikasjonsparadigmer som request/response og publish/subscribe. Det er verdt å merke seg at ingen produkter implementerer

absolutt alle standarder (noe som ikke er så merkelig tatt i betraktning at det finnes over 150 Web services standarder), så det er viktig at man velger et produkt som understøtter de behovene man har. WSO2 mangler for eksempel støtte for WS-Notification-standarden, noe som er beklagelig ettersom det er denne standarden NATO har valgt å bruke for publish/subscribe (jfr. NATO Core Enterprise Services Working Group [7]).

5 Tanker rundt bruk av ESB i Forsvaret

De grunnleggende egenskapene i C2-filosofien til Alberts og Hayes [1] om *kraft-til-kanten* ("power to the edge") er at man skal gi informasjon som bidrar til at relevant situasjonsforståelse kan oppnås:

- autonomt synkroniserte operasjoner snarere enn autonome operasjoner,
- målrettet innhenting av relevant informasjon framfor ukritisk informasjonskringkasting,
- innsats via samarbeid fremfor enkeltprestasjoner,
- interesseområder ("Communities of Interest" (COI)) snarere enn informasjonssiloer,
- dele data snarere enn å opprettholde private data,
- vedvarende, kontinuerlig informasjonssikkerhet snarere enn en enkeltsjekk ved domenegrensen,
- behovsdrevne kapabiliteter snarere enn forhåndstildelte kapabiliteter,
- bruk av åpne standarder for interoperabilitet,
- felles kjernetjenester i stedet for separate infrastrukturer, og
- hyllevarebaserte nettsentriske kapabiliteter fremfor informasjonssiloer.

Disse egenskapene er forenlige med å benytte en (eller flere) ESBer i informasjonsinfrastrukturen, forutsatt at åpne standarder ligger til grunn for produktet. Vi skal nå se nærmere på hvordan man hypotetisk sett kunne tenke seg å bruke WSO2 sin produktportefølje (introdusert i studien i Kapittel 4) i C2-sammenheng.

5.1 Potensiell bruk av WSO2 i Forsvaret

Nedenfor gis et eksempel på hvordan de ulike aspektene ved en ESB kan benyttes. Det er ikke ment som en anbefaling av WSO2 på noen måte. ESB må velges som resultat av en grundig behovsanalyse, samt en vurdering av hvorvidt produktet kan sertifiseres til ønsket sikkerhetsnivå. Sistnevnte er ikke et fokus i denne rapporten, som hovedsakelig peker på fordeler og muligheter ved å ta i bruk en ESB. Disse betraktningene baserer seg på vurderinger gjort i [30], hvor man kan finne ytterligere detaljer om hvordan WSO2 kan understøtte militære informasjonsbehov.

5.1.1 WSO2 Carbon

Dette er et integrert miljø som inneholder felles funksjonalitet som deles av alle WSO2-komponenter, inkludert et innebygget register, OSGi, tjeneste- og brukeradministrasjon, ulike transportlag, sikkerhet, logging, clustering, caching med mer. Denne modulen er grunnsteinen i WSO2-porteføljen, og vil i

en C2-sammenheng være ansvarlig for informasjonshåndtering og -distribusjon.

5.1.2 WSO2 API manager

Dette er en løsning for å publisere, opprette og administrere APIer. I C2-sammenheng kan man tenke seg at modulen blir benyttet for å oppnå en viss selvbetjening av grensesnitt, slik at brukerne kan utvide og konfigurere systemets egenskaper basert på et knippe forhåndsgodkjente funksjoner og tjenester.

5.1.3 WSO2 Application server

WSO2 sin applikasjonstjener støtter åpne standarder og kan benyttes for å deployere Web services og understøtte webapplikasjoner. Andre leverandørers applikasjonstjenere tilbyr liknende funksjonalitet, så også her gjelder det å velge et produkt som støtter de behovene man har. F.eks. vil Microsoft sin IIS fungere spesielt godt for tjenester utviklet i .NET, mens Oracles GlassFish har en tett integrering mot Enterprise Java. Om man f.eks. bruker WSO2 sin ESB så er det ikke noe krav om at man deployerer tjenester inne i ESBen; man kan fint bruke for eksempel GlassFish til tjenestene sine og så knytte denne mot ESBen.

5.1.4 WSO2 Business Activity Monitor

Denne overvåker systemmetrikker og sentrale forretningsprosessindikatorer. Den kan konfigureres for å rapportere statusendringer som brukeren er interessert i, for eksempel varslinger basert på organisatorisk definerte prioriteringer.

5.1.5 WSO2 Business Process Server

Komponenten muliggjør enkel deployering av forretningsprosesser uttrykt med WS-BPEL-standarden. Kan tenkes brukt til å tillate hurtig og konfigurerbar spesifisering og utplassering av prosesser som kan understøtte C2.

5.1.6 WSO2 Business Rules Server

Denne tjeneren tilbyr lagring og vedlikehold av forretningsregler. For C2 kan man tenke seg å lagre operasjonelle regler her, slik at systemer kan benytte dem i organisatorisk øyemed.

5.1.7 WSO2 Complex Event Processor

Denne modulen identifiserer de viktigste hendelser i hendelsesrommet, analyserer deres virkninger, og håndterer dem i sanntid. For C2 kan en slik modul kanskje nyttiggjøres til å analysere operasjonelle data i sanntid, og varsle basert på taktiske og/eller strategiske prioriteringer. Modulen kan kanskje også benyttes til sensorfusjon og være et hjelpemiddel i ulike beslutningsprosesser.

5.1.8 WSO2 Data Services Server

Enkelt fortalt er dette en modul som tilbyr et overordnet, felles grensesnitt mot ulike lagringsformer. Den kan for eksempel knyttes mot en eller flere relasjonsdatabaser. En slik modul er viktig fordi den muliggjør integrasjon av allerede eksisterende datalagre i en SOA-basert infrastruktur, og på en slik måte fremmer interoperabilitet mot eksisterende lagringsløsninger.

5.1.9 WSO2 Elastic Load Balancer

Denne sørger for automatisk lastbalansering når belastningen endrer seg. Den tilbyr styring i runtime som sikrer at systemer oppnår den nødvendige pålitelighet og tilgjengelighet. Lastbalansering i en eller annen form er viktig i alle distribuerte systemer der det tilbys mer enn én instans av en ressurs for å oppnå en mest mulig fornuftig fordeling av ressursbruken.

5.1.10 WSO2 Enterprise Service Bus

ESBer er hovedfokuset i denne rapporten, og som vi allerede har nevnt tidligere er det ingen universell enighet om nøyaktig hva en ESB er. WSO2 definerer sin ESB til å være en plattform som tilbyr transport, transformasjon, og mediasjon av data. Denne grunnfunksjonaliteten kan koples sammen med noen eller alle de øvrige WSO2-produktene, eller med tilsvarende produkter fra andre leverandører, ettersom den baserer seg på åpne standarder.

5.1.11 Øvrige produkter

Øvrige produkter i porteføljen til WSO2 som ikke omtales i detalj her er: Gadget Server, Governance Registry, Identity Server, Message Broker, og Web services framework for PHP.

5.2 Behov for fødererte ESBer

Det er forventet at NATO NEC-miljøer i mange tilfeller vil bestå av føderasjoner av ESBer [9]. Det er derfor svært viktig å definere standarder for grensesnitt og kommunikasjon mellom slike (gjerne ulike) ESBer, slik at data skal kunne flyte mellom dem, og for å kunne integrere ESBene i informasjonsinfrastrukturen. Her må NATOs SOA Baseline være utgangspunktet, og sannsynligvis vil de eksisterende spesifikasjonene herfra dekke de fleste behov i ESB-sammenheng. Det kan imidlertid ikke utelukkes at det vil måtte gjøres enkelte utvidelser på sikt.

6 Konklusjon

En viktig konklusjon fra denne rapporten er at bruk av en ESB ikke er noen forutsetning for å lage en tjenesteorientert arkitektur, og at det å ta i bruk en ESB er heller ikke noen automatisk garanti for at man oppnår en tjenesteorientert arkitektur. En ESB er en verktøykasse, og hvilke verktøy den inneholder varierer fra produkt til produkt. Dersom man ønsker å benytte en ESB i sin tjenesteorienterte arkitektur, så er det derfor viktig å velge et produkt (eller flere i kombinasjon) som

inneholder de verktøyene man faktisk trenger. Man bør så langt det er mulig unngå å låse seg for mye til én leverandør (vendor lock-in).

Innledningsvis stilte vi tre spørsmål som nå kan besvares:

1. *Hva er en ESB (hva er den laget for å gjøre)?*
   ESB er en programvarearkitektur som muliggjør applikasjonsintegrasjon. Den grunnleggende funksjonaliteten som må være på plass for at et produkt skal kunne kalles en ESB er meldingsruting. I tillegg til dette vil som oftest produkter støtte funksjonalitet som mediasjon, tjenesteoppdagelse, tjenesteforvaltning, sikkerhet, med mer. En ESB egner seg godt som plattform for å realisere en tjenesteorientert arkitektur.
2. *Hvorfor er ESBer av interesse for Forsvaret?*
   Tjenesteorientering står sentralt i tanken om et Nettverksbasert Forsvar, og bør benyttes i informasjonsinfrastrukturen for å harmonisere med sentrale ideer fra NATO (jfr. NNEC Feasibility Study [3] og SOA Baseline [7]). Til forretningsbruk er ESBer i dag den foretrukne metoden for å implementere en tjenesteorientert arkitektur, blant annet på grunn av sin skalérbarhet. Forsvaret bør derfor se nærmere på denne teknologien, med tanke på å nyttiggjøre seg eksisterende produkter basert på åpne standarder.
3. *Hvordan bør Forsvaret benytte ESBer?*
   Innledningsvis bør Forsvaret kartlegge sine SOA-behov, for best å kunne velge en (eller flere) ESBer å benytte i sin SOA-baserte infrastruktur. En ESB vil være et naturlig produkttype å velge for å realisere kjernetjenestene i informasjons-infrastrukturen, der det er ønskelig å kunne tilby et effektivt og robust mellomvaresystem som kan håndtere mange samtidige brukere. Det er svært viktig å prioritere produkter som benytter standardene som anbefales i NATOs SOA Baseline.

For ytterligere tanker om bruk av ESBer i forsvarssammenheng, da med spesielt fokus på NATOs behov, anbefaler vi interesserte lesere å se på rapporten fra NCIA (tidligere NC3A) om emnet [9].

Referanser

[1] D. Alberts and R. Hayes. Power to the Edge: Command and Control in the Information Age. CCRP Publication Series, ISBN 1-893723-13-5, 2003.

[2] J. Babiarz, K. Chan, and F. Baker. Configuration Guidelines for DiffServ Service Classes. RFC 4594 (Informational), Aug. 2006.

[3] P. Bartolomasi, T. Buckman, A. Campbell, J. Grainger, J. Mahaffey, R. Marchand, O. Kruidhof, C. Shawcross, and K. Veum. NATO network enabled capability feasibility study. Version 2.0, October 2005.

[4] T. Bilski. Trends in quality of service on long-distance connections. in proceedings of the Military Communications and Information Systems Conference (MCC 2009), Prague, Czech Republic, September 2009.

[5] D. Black, S. Brim, B. Carpenter, and F. L. Faucheur. Per Hop Behavior Identification Codes. RFC 3140 (Proposed Standard), June 2001.

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Service. RFC 2475 (Informational), Dec. 1998. Updated by RFC 3260.

[7] Consultation, Command and Control Board (C3B). CORE ENTERPRISE SERVICES STANDARDS RECOMMENDATIONS: THE SOA BASELINE PROFILE VERSION 1.7. Enclosure 1 to AC/322-N(2011)0205, NATO Unclassified releasable to EAPC/PFP, 11 November 2011.

[8] B. Davie, A. Charny, J. Bennet, K. Benson, J. L. Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. An Expedited Forwarding PHB (Per-Hop Behavior). RFC 3246 (Proposed Standard), Mar. 2002.

[9] R. Fiske, M. Lehmann, G. Lunt, D. Marco-Mompel, L. Schenkels, and V. de Sortis. Esb implementation considerations. Technical report, NATO C3 Agency, 2011.

[10] D. Grossman. New Terminology and Clarifications for Diffserv. RFC 3260 (Informational), Apr. 2002.

[11] M. Hauge, J. Andersson, M. A. Brose, and J. Sander. Multi-topology routing for improved network resource utilization in mobile tactical networks. IEEE MILCOM, 2010.

[12] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. RFC 2597 (Proposed Standard), June 1999. Updated by RFC 3260.

[13] F. T. Johnsen, T. H. Bloebaum, and M. R. Brannsten. Quality aspects of Web services. FFI-rapport 2012/02494.

[14] F. T. Johnsen, T. H. Bloebaum, L. Schenkels, R. Fiske, M. van Zelm, V. de Sortis, A. van der Zanden, J. Sliwa, and P. Caban. SOA over disadvantaged grids experiment and demonstrator.

Military Communications and Information Systems Conference (MCC) 2012, Gdansk, Poland, 8-9 October, 2012.

[15] F. T. Johnsen, T. Hafsøe, M. Hauge, and Ø. Kolbu. Cross-layer quality of service based admission control for web services. in proceedings of the 6th International Workshop on Heterogeneous, Multi-hop, Wireless and Mobile Networks (HeterWMN) 2011, Houston, TX, USA, 2011.

[16] J. Krygier, P. Lubkowski, and K. Maslanka. Simulation study of selected qos supporting components for nec compliant networks. in proceedings of the Military Communications and Information Systems Conference (MCC 2009), Prague, Czech Republic, September 2009.

[17] E. Kubera, J. Milewski, and M. Rozycki. Qos support in the special wireless networks. in proceedings of the Military Communications and Information Systems Conference (MCC 2009), Prague, Czech Republic, September 2009.

[18] A. T. Manes. Enterprise Service Bus: A Definition. Technical report, The Burton Group, 2007.

[19] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474 (Proposed Standard), Dec. 1998. Updated by RFCs 3168, 3260.

[20] N. A. Nordbotten. *XML and Web Services Security*. FFI report 2008/00413, 2008.

[21] OASIS. Security assertion markup language (saml) v2.0 technical overview, committee draft 02, 25 march 2008. `http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0-cd-02.pdf`.

[22] OpenSAML website. Home page. `https://wiki.shibboleth.net/confluence/display/OpenSAML/Home/`.

[23] Oracle. Glassfish home page. http://glassfish.java.net/.

[24] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault. Multi-Topology (MT) Routing in OSPF. RFC 4915 (Proposed Standard), June 2007.

[25] S. Sakr. Xml compression techniques: A survey and comparison. *Journal of Computer and System Sciences*, 75(5):303 – 322, 2009.

[26] R. W. Schulte. Succeeding with multiple soa service domains and disparate esbs. Gartner Research, report ID Number: G00143293, 3 May 2007.

[27] M. Skjegstad. Mobiemu home page. `https://code.google.com/p/mobiemu/`.

[28] M. Tian, A. Gramm, H. Ritter, and J. Schiller. Efficient selection and monitoring of qos-aware web services with the ws-qos framework. IEEE/WIC/ACM International Conference on Web Intelligence (WI'04), pp. 152-158, 2004.

[29] WSO2. Wso2 home page. http://wso2.com/products/enterprise-service-bus.

[30] WSO2 White Paper. The Revolution in Military Affairs 2.0: Information Dominance and the Democratization of Information Technology. http://wso2.com/whitepapers/, December 18, 2012.

[31] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end-to-end qos constraints. ACM Transactions on the Web (TWEB), Volume 1 Issue 1, Article No. 6, May 2007.

Ordliste

Forkortelse	Beskrivelse
C2	Forkortelse for *Kommando og Kontroll* (fra engelsk “Command and Control”)
CoNSIS	Coalition Network for Secure Information Sharing
CWIX	Coalition Warrior Interoperability eXploration, eXperimentation and eXamination, eXercise
EAI	Enterprise Application Integration
ebXML	electronic business using XML
ESB	Enterprise Service Bus
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IST	Information Systems Technology
JMS	Java Message Service
MTOM	Message Transmission Optimization Mechanism
OSGi	OSGi Alliance (tidligere *Open Services Gateway Initiative*)
NATO RTO	NATO Research and Technology Organization (nylig omdøpt til NATO STO)
NATO STO	NATO Science and Technology Organization (tidligere NATO RTO)
NbF	Nettverksbasert Forsvar – Norges tilnærming til NNEC
NEC	Network Enabled Capability
NNEC	Forkortelse for NATO NEC
SAML	Security Assertion Markup Language
SOA	Service-Oriented Architecture / Tjenesteorientert arkitektur
SOAP	Tidligere en forkortelse for *Simple Object Access Protocol*, men nå bare et navn.
UDDI	Universal Description, Discovery and Integration
WS-	Forkortelse for *Web Services*, benyttes ofte om prefiks i navn på standarder (f.eks. WS-BPEL).
WS-BPEL	Web Services Business Process Execution Language
WS-Discovery	Web Services Dynamic Discovery
WSDL 1.1	Forkortelse for *Web Services Definition Language* (merk nytt navn i versjon 2)
WSDL 2.0	Forkortelse for *Web Services Description Language* versjon 2.
WSO2	WSO2 er et firma med spesielt fokus på open source og SOA-løsninger.
XACML	Extensible Access Control Markup Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

Appendiks A Role-based Quality of Service for Web Services

Abstract

We have designed and implemented a prototype system providing role based Quality of Service (QoS) for Web services in heterogeneous networks. We leverage industry standards to the fullest extent, in an attempt to bring role based QoS support to standard Web services. We have extended an existing enterprise service bus to accommodate the changes necessary for prioritization on the server side, and created a custom client library to ensure prioritization in both the request and the response of the Web services message exchange. Finally, roles are defined using Security Markup Assertion Language (SAML) tokens. Our framework has been released as open source.

Our evaluation shows that the concept is viable, and that prioritization on the application level of the OSI model, combined with network level prioritization as provided by DiffServ, is beneficial in networks with low bandwidth.

A.1 Introduction

The Service-Oriented Architecture (SOA) principle is becoming a common approach to use when designing and building large distributed software systems. SOA allows for loose coupling of systems, and is thus ideally suited for building large-scale systems-of-systems. The most mature and most common technology used for implementing SOAs is Web services, a middleware technology which is based on standards, and which is capable of implementing a system based on SOA principles.

So far Web services have mostly been used on the Internet and within business networks, and the most commonly used protocol bindings are best suited for these types of networks. However, because of the flexibility offered by Web services, it is beginning to be more used also in wireless networks, ranging from IEEE 802.11 networks to specialized military radio communication networks.

This shift from mainly high capacity wired networks to more widespread use has led to an increased focus on the network resource consumption of Web services. Specific compression mechanisms have been developed for the eXtensible Markup Language (XML) format used by Web services [25], and other protocol bindings have been developed as an alternative to the standard HTTP-over-TCP binding used in most systems.

Due to the interoperability benefits offered by Web services, such as loose coupling and a strong standards foundation, NATO has decided to focus on this technology as a basis for interoperability between coalition members. Additional benefits, such as the ability to reuse services to build more complex services, and the flexibility of Web service enabling already existing systems, has led to further adoption of this technology among partner nations.

Quality of Service (QoS) is an important aspect of any network, but it is of particular importance for systems that need to be able to function both in infrastructure networks and in radio based communication networks. On the Internet and in corporate networks the desired QoS can often be

achieved by overprovisioning of resources in the networks. In wireless networks this is not possible to the same extent, because the maximum available bandwidth in such networks is a function of radio capabilities, distance between the communicating nodes, interference, node mobility, the number of network hops, and so on. This means that there is a need for QoS frameworks supporting such networks. In this paper, we address the issue of role based QoS support for using Web services technology across heterogeneous, multi-hop wireless networks.

The remainder of this paper is organized as follows: In Section A.2 we present related work. Section A.3 discusses the design and implementation of our prototype software, whereas Section A.5 covers the evaluation. Section A.6 concludes the paper.

A.2 Related work

Previously, we have implemented a QoS based admission control mechanism, which provides priority based access to the network, while at the same time avoiding overloading the limited network capacity that is available [15]. In this paper we expand on those ideas by building a role based QoS framework founded on industry standards for Web services clients and services.

Hauge et al. have shown how *Multi-Topology routing* [24] can be leveraged in heterogeneous tactical mobile ad hoc networks to improve the network resource utilization [11]. They suggest using a QoS model with a routing protocol which maintains several distinctive network topologies, where each topology is tailored to support either a single or multiple QoS classes. We assume the presence of such Multi-Topology routing for the work we present in this paper.

The Differentiated Services (DiffServ) mechanism can be used to ensure that some traffic is prioritized, a task that it performs quite well through actual use and simulations, see experiment details in [16], [28], [4], and [17]. DiffServ relies on tagging the type-of-service (TOS) field in the IP header with a bit pattern corresponding to a certain traffic class. This allows DiffServ-enabled routers to prioritize the IP packets with a deterministic per-hop-behavior (PHB). DiffServ is covered by several RFCs:

- Definition of the Differentiated Services Field in the IPv4 and IPv6 Headers [19],
- An Architecture for Differentiated Services [6],
- Assured Forwarding PHB Group [12],
- Per Hop Behavior Identification Codes [5],
- An Expedited Forwarding PHB [8],
- New Terminology and Clarifications for DiffServ [10], and
- Configuration Guidelines for DiffServ Service Classes [2].

In this paper, we rely on DiffServ for enforcing network level QoS.

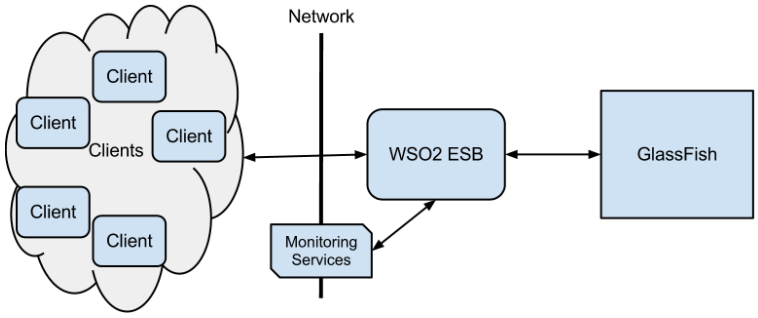
Yu et al. [31] have investigated issues regarding service selection with multiple QoS constraints and proposed several algorithms. Their work is orthogonal to ours, in that their approach can be leveraged when you have multiple available services. In this paper our services reside in an enterprise service bus (ESB), thus yielding a single service source. In future versions of our prototype we may add

support for these algorithms as well for even further gains when there are multiple copies of the available services in the network.

A.3 Design and implementation

Essential to Network Based Defense (NBD) is the concept of end-to-end QoS, which in turn requires employing cross-layer QoS signaling. This means that QoS must be considered at all layers of the OSI model, and that QoS information must traverse these layers. To achieve the end-to-end QoS needed in NBD, QoS metadata must also be allowed to cross both network and national boundaries. There are QoS mechanisms that can be used on the transport layer and below, and thus we focus our research efforts on the application layer and issues regarding cross-layer QoS signaling. Having IP as a common protocol and assuming DiffServ as the network level QoS framework, we focus on the application level solutions in this paper (i.e., the Web services middleware). Our goal is to provide prioritized access to Web services based on the client's role, and enforce this at the network level by mapping the demands to the TOS field in the IP header, enabling cross-layer QoS signaling. DiffServ provides coarse traffic shaping, so it is desirable to have finer grained control on the application level by taking user needs (represented by a role) and available resources (the current network resource state) into account.

In this section we discuss the design and implementation of the prototype system. First we present the different components used in our solution, then we cover the details regarding the server and client side implementation.



Figur A.1 Prototype overview

A.4 Components

We use the following components as part of our prototype design:

- Multi-Topology router with exposed monitoring service
- WSO2 ESB
- GlassFish application server
- Security Assertion Markup Language (SAML)

The monitoring service exposes the active routing table from the router, thus allowing our software to glean information about the current maximum available bandwidth. The router implements and performs Multi-Topology routing [11]. We employ the router as described in our previous work [15], in this paper we expand on the QoS support given by the Web services layer as described below.

The WSO2 Enterprise Service Bus (ESB) supports industry standards and is an open source platform for deploying Web services. It is in widespread use today, being used by several large companies (e-bay being a prominent example) [29].

We employ the GlassFish Server Open Source Edition for hosting Web services, since it is a robust and free community-supported application server featuring full Java EE 6 platform support [23].

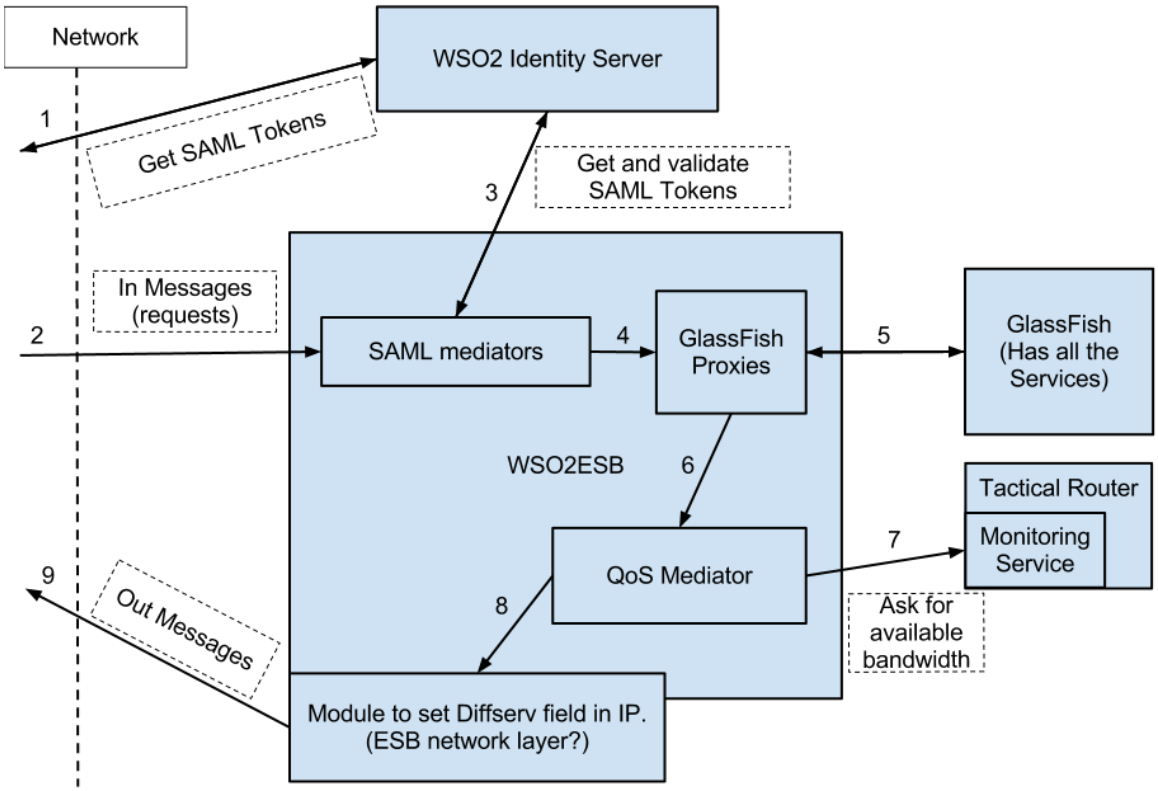
SAML is an XML-based framework for request/response exchanges of authentication and authorization information [21]. SAML assertions describe the results of authentication actions that occurred previously. In our framework, SAML tokens are used to identify the client's role.

These components are used in our prototype, as Fig: A.1 illustrates. There, a cloud of clients (which can be in the same local network or in different networks) access Web services hosted through the WSO2 ESB. The pair <*role, service*> is used by the client library as well as the QoS functionality we implemented in the ESB to determine which QoS class a request and response is handled according to. The ESB mediates the client request messages into GlassFish where the Web services are deployed, and the requests are processed. In order for the ESB to know how much bandwidth each client can utilize, it is dependent on the monitoring service which relays that sort of network information. The monitoring service resides in a Multi-Topology router functioning as a gateway to another network. The Web service response is mediated through the ESB and sent back to the client. The information flows on the server and client sides are discussed in detail below.

A.4.1 Server side architecture

The server side architecture consists of the WSO2 ESB and the GlassFish application server. Furthermore, the server side needs access to one or more monitoring services as provided by a Multi-Topology router. All of these components were already available (see [29, 23, 15]), so what we needed to make were custom *mediators* in the ESB. A mediator is a component in WSO2 ESB which can be used to work on incoming or outgoing messages that pass through the ESB.

Before the client can access a Web service it has to have a SAML token for identification. To get an ID-token it has to contact the Identity Server using the ESB as a *proxy* (i.e., an intermediary between clients and servers) (Fig: A.2-1). Then the client can access a Web service from the ESB. Several things then happen in the ESB: First the request message is sent to the SAML mediator (Fig: A.2-2), this mediator contacts the Identity Server to validate the client's ID-token (Fig: A.2-3). If the token is validated, then the client is granted access to the requested service, and the message is passed on to the GlassFish proxies (Fig: A.2-4), otherwise the message is dropped. The ESB then sends the request along to the corresponding service on the GlassFish server (Fig: A.2-5).



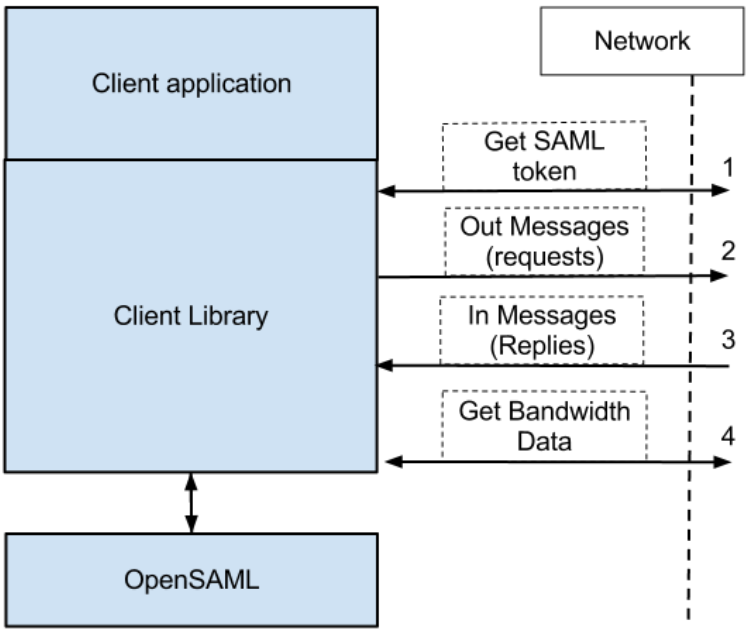
Figur A.2 The Server side Architecture

When the request is received by GlassFish, it processes the request and issues a reply message. This reply message is also passed through the ESB on its way back to the client. First the message is sent to the QoS mediator (Fig: A.2-6). This mediator will first look at the role of the client as well as the service requested, and use this information to assign a priority to the reply. If enabled, it will also perform throttling using our Throttle mediator: The Throttle mediator is used to ensure that high priority messages are sent first. To determine what to disrupt and what to hold back, and for how long, several properties are used; the priority of the message, the available bandwidth, the IP address of the Multi-Topology router, and the real time demand of the request. In order to do this, the mediator must keep a list of sending messages and where those messages are going.

Then the monitoring service is contacted for bandwidth information (Fig: A.2-7), which is used together with the priority to determine whether the message should be sent right away or held back until some higher priority message is finished sending. Finally, the ToS field in the IP header is set (Fig: A.2-8) before the message is sent to the client (Fig: A.2-9). This field is used by the routers in the network to prioritize packet sending.

A.4.2 Client side architecture

The client side architecture is composed of our role based QoS-enabling client library, utilizing OpenSAML [22] for SAML support. Our library can be used by existing client applications, one only needs to replace the existing Web service call with a call to our library to gain QoS support.



Figur A.3 The Client side Architecture

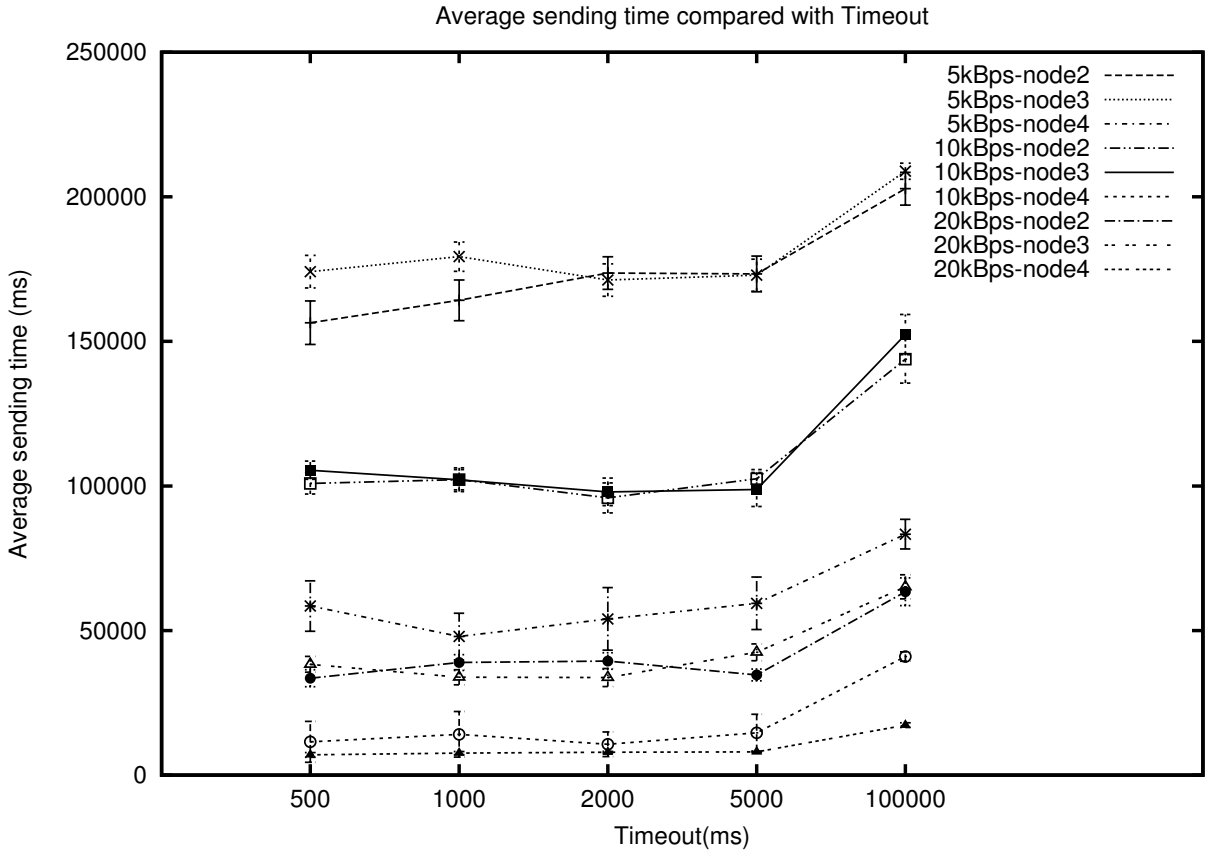
Before the client library can ask for the data the client needs to get a SAML authentication token (Fig: A.3-1). The client library then sends the request from the client to the server (Fig: A.3-2), appending the SAML token to the message as well as adding some metadata in the SOAP header related to the client role and setting the TOS field of the IP packets to the corresponding value.

The reply from the server is examined by our client library for the metadata the server has added to the SOAP header. Relevant metadata is stored for future communication and the message is passed to the client application (Fig: A.3-3).

When new communication is initiated after this first connection is made the client should, if everything went as expected, have the necessary information to prioritize new messages. This means that the client can now make an informed decision about how it should prioritize the messages it sends. In order to do this it can also take into consideration the available bandwidth (Fig: A.3-4) as provided by the monitoring service. This information may then be leveraged for admission control as we described in [15].

Client	Role	Interval	Number of requests	Delay
2 and 3	Low priority	1000	100	10
4	High priority	3000	30	15

Tabell A.1 Client parameters



Figur A.4

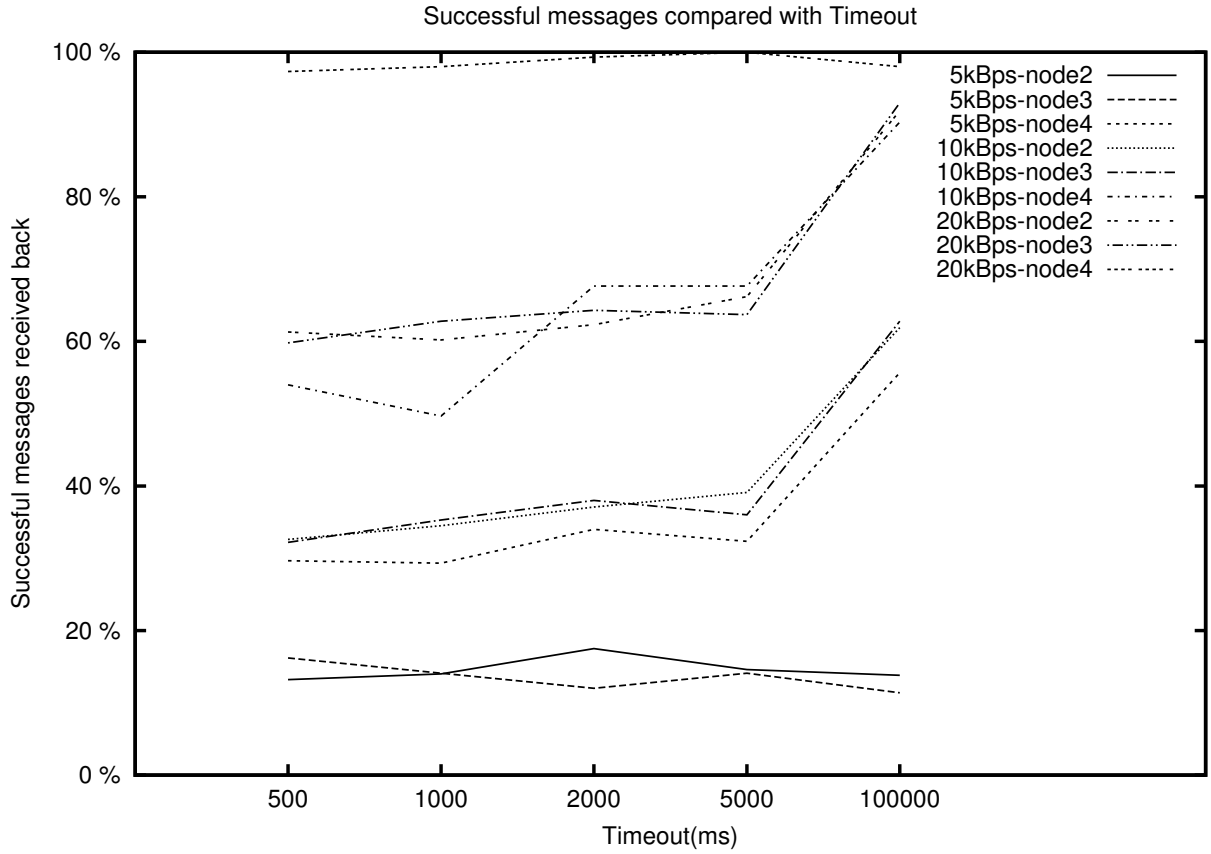
A.5 Evaluation

In this section we first present the test framework employed, before we discuss the test results.

A.5.1 Test framework

Our tests were performed in *mobiemu* — a freely available open source framework for emulating mobile ad-hoc networks with Linux containers (LXC) and ns-3 [27]. The mobiemu framework operates by creating LXC and connecting these to so-called *tap bridges* created by ns-3. This then emulates any network possible to create in ns-3. From the point of view of programs running inside the LXC, they are full Linux machines connected to a real network. This means that any program able to run on Linux should run properly inside the LXC and any messages they send out are sent through ns-3. This means that we have full control over how our network behaves, allowing us to emulate different scenarios.

When mobiemu starts up it creates a number of LXC, it then starts up ns-3 and connects all LXC to a corresponding tap bridge. Inside each of the LXC it then starts the experiment script and waits for the experiment to finish before cleaning up. Before each run mobiemu stores all files and folders in one experiment folder in order to enable easy experiment repetition. When the experiment is done



Figur A.5

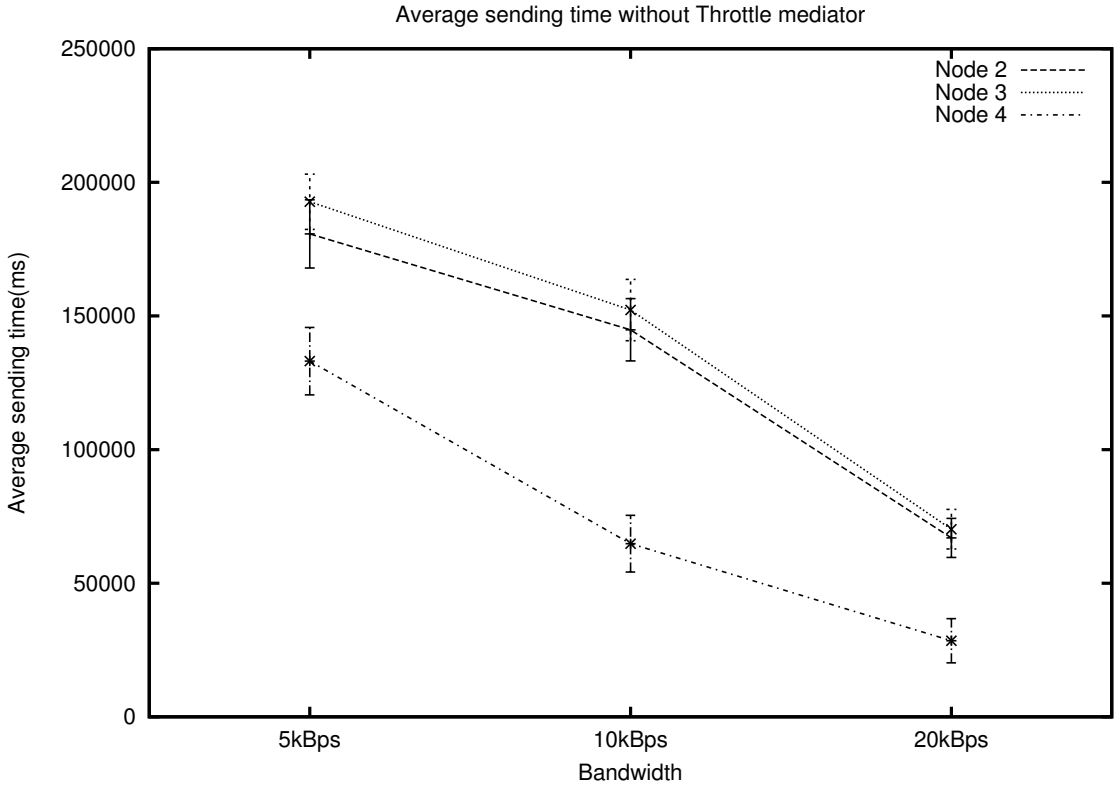
the result files are moved into the result folder.

Our tests were set up as follows: We have three clients, two clients with low priority and one client with high priority. They communicate with the ESB as described above in Sections A.4.1 and A.4.2.

The tests were simple and contained four nodes in mobiemu, functioning as a proof-of-concept of our prototype: Node 1 — ESB, GlassFish, and monitoring service. Nodes 2-4 — Clients: 2-3 have low priority roles, 4 has a high priority role. The clients were configured as shown in Table A.1. All nodes issue the same size request and receive the same size payloads. The request is a SOAP message with one parameter, whereas the reply is a SOAP message with 10 KB payload.

A.5.2 Results

In Figure A.4 we have compared the average sending time with the timeout used in our Throttle mediator and added all the different bandwidths into one graph. From the graph we can see that increasing the timeout dramatically also has an effect on the time needed to send a message. We can also see that a higher bandwidth lowers the sending time, but the high priority node, Node 4 in the graph, generally does quite well and has a substantially lower sending time compared to the other nodes. If we compare the high priority node at a speed of 5 kBps we can see that the node fares better



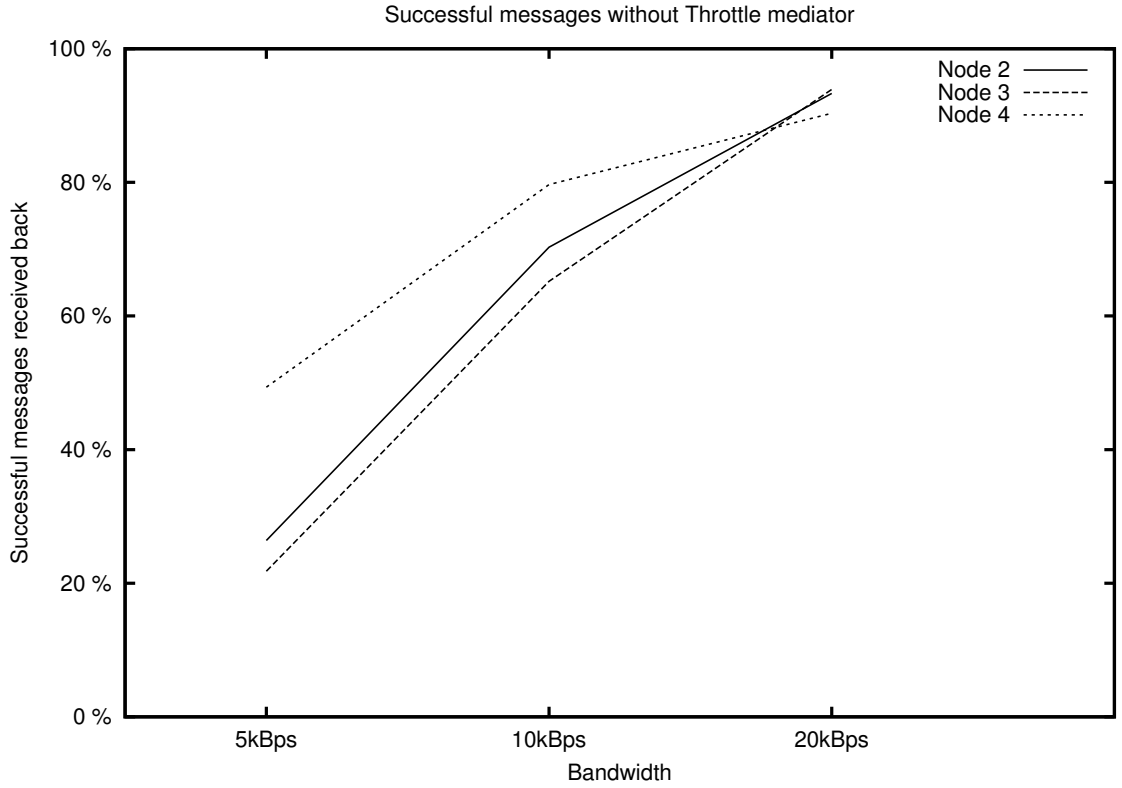
Figur A.6

than the low priority node even though they have twice the bandwidth.

In Figure A.5 this next graph we have compared the percentage of successful messages received back with the timeout used in the Throttle mediator. The first thing we can see is that when it comes to messages a higher timeout is better. The reason behind this is simple as the timeout is the time before the Throttle mediator preempts messages because they have been sending for too long. We can also see that with enough bandwidth in the network the timeout has little effect as the message is sent before we need to preempt. As with the time graph we can see that the higher priority node does considerably better than the lower priority ones.

In Figure A.6 we have compared the average sending time to the bandwidth in the network without the Throttle mediator. As one would expect the higher the bandwidth the lower the sending time is. Note that Node 4 has a substantially lower average sending time than the two other nodes. The reason behind this is twofold: First, since Node 4 sends fewer messages with more time between them it has a smaller sample than the two other nodes. Second, the ESB is still prioritizing messages, even though there is no Throttle mediator present to make sure that there is enough bandwidth available for the higher priority messages.

In Figure A.7 we have compared the successful messages with the bandwidth in the network. If we compare this graph to the graph above which had the Throttle mediator we can see that the lower priority nodes perform better, but at the cost of the higher priority node.



Figur A.7

A.6 Conclusion and future work

We have designed and implemented a prototype system providing role based QoS for Web services in heterogeneous networks. We have extended an existing ESB to accommodate the changes necessary for prioritization on the server side, and created a custom client library to ensure prioritization in both the request and the response of the Web services message exchange.

Our evaluation shows that the concept is viable, and that prioritization on the application level of the OSI model, combined with network level prioritization as provided by DiffServ, is beneficial in networks with low bandwidth. Our prototype software was developed using the Java EE 6 platform, and is available as open source at `https://github.com/magnuskiro/it2901`.

Future work involves addressing aspects regarding accessing multiple simultaneous service deployments across heterogeneous networks, along with further developing the monitoring service. The current service only yields information about maximum bandwidth, but others providing additional metrics may enable even more fine-grained QoS control. Finally, we plan on employing our prototype in a real-life experiment in the future, where heterogeneous communications equipment will take part.