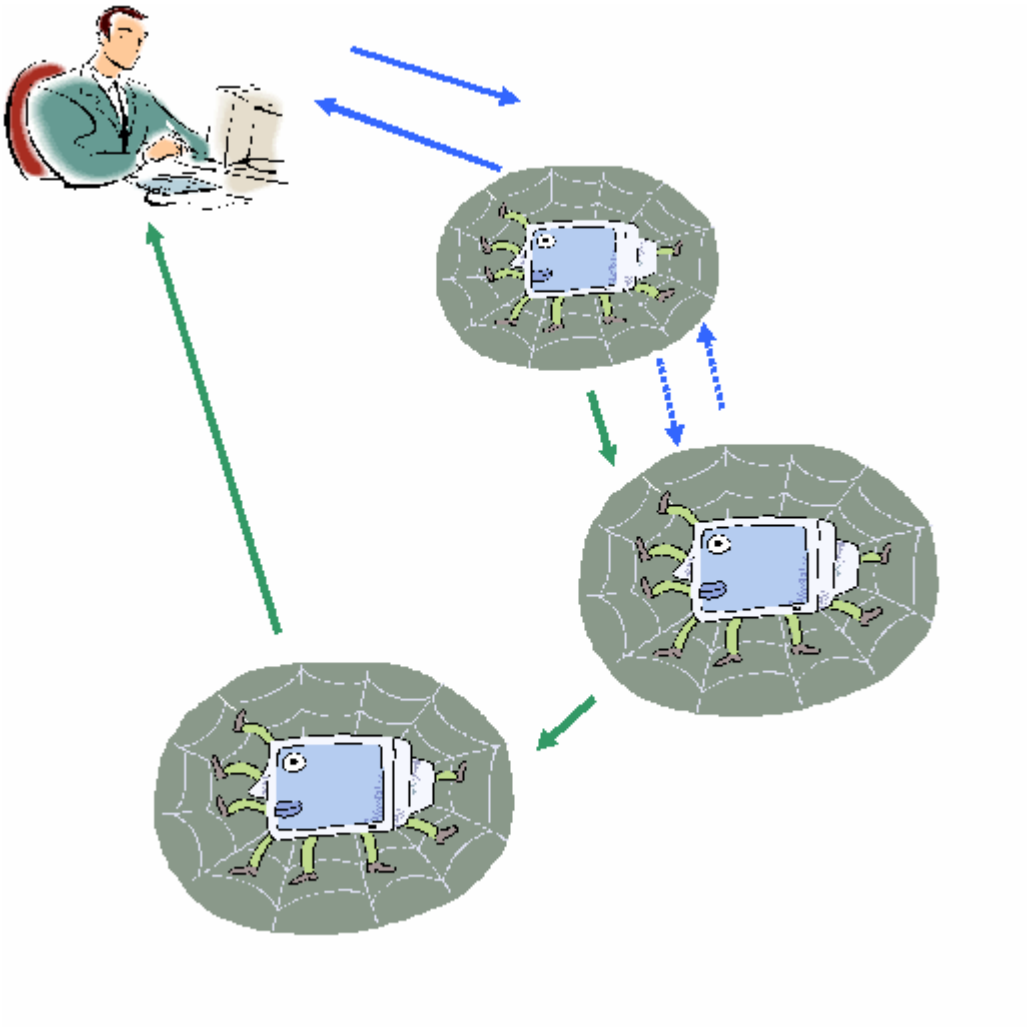
FOI
TOTALFÖRSVARETS
FORSKNINGSINSTITUT

FOI-R--1399--SE
November 2004
ISSN 1650-1942
Vetenskaplig rapport

Alf Bengtsson

Spårning vid samverkande Web Services



Ledningssystem
581 11 Linköping

TOTALFÖRSVARETS FORSKNINGSINSTITUT

Ledningssystem
Box 1165
581 11 Linköping

FOI-R--1399--SE

November 2004

ISSN 1650-1942

Vetenskaplig rapport

Alf Bengtsson

Spårning vid samverkande Web Services

Utgivare	Rapportnummer, ISRN	Klassificering
Totalförsvarets Forskningsinstitut - FOI	FOI-R--1399--SE	Vetenskaplig rapport
Ledningssystem	**Forskningsområde**	
Box 1165	4. Ledning, informationsteknik och sensorer	
581 11 Linköping	**Månad, år**	**Projektnummer**
	November 2004	E7083
	Delområde	
	41 Ledning med samband och telekom och IT-	
	Delområde 2	

Författare/redaktör	Projektledare
Alf Bengtsson	Alf Bengtsson
	Godkänd av
	Martin Rantzer
	Uppdragsgivare/kundbeteckning
	FM
	Tekniskt och/eller vetenskapligt ansvarig
	Alf Bengtsson

Rapportens titel

Spårning vid samverkande Web Services

Sammanfattning (högst 200 ord)

Web Services är en stark kandidat för att realisera den tjänsteorienterade arkitektur, SOA, som har fastställts för det framtida FMLS, Försvarsmaktens Ledningssystem. Många uppgifter kommer att lösas genom att flera Web Services samverkar. Ett viktigt säkerhetskrav är att man skall kunna spåra identiteterna för de Web Services som medverkat vid uppgiftens lösning.

Vi presenterar en modell för spårning av identiteter vid samverkande Web Services. Modellen är ny, speciellt i den meningen att den modellerar en typ av samverkan, via asynkrona envägsmeddelanden, som ännu inte så ofta har implementerats. Modellen innebär att de meddelanden, som skickas mellan Web Services för att utföra en uppgift, innehåller data med identiteter för samverkande aktörer. Dessa data bygger upp ett hierarkiskt träd och de är signerade på ett sätt som förhindrar deltagarna att agera i annans namn eller att dölja sin medverkan.

Nyckelord

SOA, webbtjänster, digital signatur, historik, spårning

Övriga bibliografiska uppgifter	Språk Svenska
ISSN 1650-1942	**Antal sidor:** 42 s.
Distribution enligt missiv	**Pris: Enligt prislista**

Issuing organization	Report number, ISRN	Report type
FOI – Swedish Defence Research Agency Command and Control Systems P.O. Box 1165 SE-581 11 Linköping	FOI-R--1399--SE	Scientific report
	Programme Areas 4. C4ISTAR	
	Month year November 2004	**Project no.** E7083
	Subcategories 41 C4I	
	Subcategories 2	

Author/s (editor/s)	Project manager
Alf Bengtsson	Alf Bengtsson
	Approved by Martin Rantzer
	Sponsoring agency Swedish Defence
	Scientifically and technically responsible Alf Bengtsson

Report title (In translation)

Tracing cooperating Web Services

Abstract (not more than 200 words)

Web Services is a strong candidate to carry out the Service Oriented Architecture, SOA, that has been established for the coming FMLS, the Command and Control System for the Swedish Defence. Many tasks will be solved by Web Services in cooperation. An important security requirement is the possibility to trace the identities of the Web Services that solved the task.

We introduce a model for tracing cooperating Web Services. The model is new, in the sence that it models a type of cooperation, by asynchronous one way messaging, which as yet has not been implemented very often. The model means that the messages sent between the Web Services contain data including the identities of the cooperating Web Services. The data constitute a hierarchic tree, and they are signed in a way that prevents the participators to masquerade in the name of someone else or to hide their participation.

Keywords

SOA, Web Services, digital signature, tracing, history

Further bibliographic information	Language Swedish
ISSN 1650-1942	**Pages** 42 p.
	Price acc. to pricelist

Innehåll

Figurer

1 Sammanfattning

Föreliggande rapport ingår i projektet ”System av system – säkerhetsfrågeställningar vid hopkopplingar”. Grundmotiveringen till projektet är att FMLS, det framtida ledningssystemet, av flera skäl inte kan bestå av ett enda system. Det kommer inte ens att bestå av ett distribuerat system, utan det kommer att bestå av flera samverkande delsystem. Inte minst uppkommer krav att samverka med externa system, t ex civila system eller andra nationers system. Vid hopkopplingen av delsystemen aktualiseras flera olika säkerhetsfrågor.

Huvudsyftet med föreliggande rapport har varit att beskriva en modell för spårning av identiteter vid samverkande Web Services. Modellen är ny, speciellt i den meningen att den modellerar en typ av samverkan, via asynkrona envägsmeddelanden, som ännu inte så ofta har implementerats. Det finns dock flera tillämpningar när denna samverkansmodell är naturlig. I rapporten används en sådan tänkbar tillämpning som ett scenarioexempel.

Modellen innebär att de meddelanden, som skickas mellan Web Services för att utföra en uppgift, innehåller data med identiteter för samverkande aktörer. Data är signerade på ett sätt som förhindrar deltagarna att agera i annans namn eller att dölja sin medverkan.

I avsnitt *2 Bakgrund* motiveras varför det är relevant att studera säkerhetsfrågor avseende Web Services. Motiveringen är att dessa är en stark kandidat för att realisera den tjänsteorienterade arkitektur som har fastställts för FMLS, bland annat på grund av det starka kommersiella intresset för Web Services. I avsnittet beskrivs också den samverkan mellan Web Services som är grunden till den modell för spårning av identiteter som vi tagit fram. För att tydligare illustrera modellen har vi i delavsnitt 2.1 Scenarioexempel exemplifierat den i ett scenario med militär prägel.

Avsnitt *3 Web Services* är en exposé över de, för vår modell, viktigaste delarna och standarderna inom Web Services. Speciellt gäller det protokollet SOAP samt dess utbyggnader för säkerhetsfunktioner, WS-Security. Funktionerna för digital signatur och för signaturverifiering är nödvändiga för vår modell.

Avsnitt *4 Samverkans- och interaktionsmodeller* ägnas åt en diskussion av tre olika modeller för hur Web Services kan samverka för att lösa en uppgift. Beroende på tillämpningen kan den ena eller den andra modellen vara att föredra. Den modell som vi bygger vidare på, innebär att de samverkande Web Services anropar varandra i en följd av asynkrona envägsmeddelanden.

I avsnitt *5 Modell för spårning av samverkande Web Services* utvidgas den valda samverkansmodellen. Där beskrivs hur varje Web Service adderar, och signerar, element till en datastruktur som följer med i meddelandekedjan.

Datastrukturen bygger upp ett hierarkiskt träd. Den innehåller bland annat identiteterna hos involverade Web Services.

I avsnitt *6 Skiss över XML-struktur* beskrivs hur datastrukturen XML-kodas och hur den signeras. En implementation är påbörjad, men inte avslutad.

I avsnitt *7 Närliggande aktiviteter* refereras till några andra ansatser till samverkan mellan Web Services. Rapporten avslutas med en summering i *8 Slutord*.

2 Bakgrund

Web Services är ett framväxande implementationskoncept. I [BEN] motiverar vi varför det är relevant att inom projektet ”System av system – säkerhetsfrågeställningar vid hopkopplingar” studera WS (i fortsättningen används förkortningen WS för Web Services). Det tyngsta argumentet är att WS ligger i linje med den tjänsteorienterade arkitektur som har valts som grundarkitektur för FMLS. För att realisera denna vision om ett tjänstebaserat FMLS behövs öppna standarder för samverkan mellan WS, och för samverkan mellan WS och klienter. WS är det sätt som kommersiella aktörer hoppas kommer att bli standarden för hopkoppling av kommersiella system. WS ligger helt klart inom den kommersiella strömfåran. Därmed har WS ett antal önskvärda egenskaper – öppna standarder, mängder av COTS-komponenter, goda möjligheter för samverkan med externa system, m fl. I den negativa vågskålen ligger tveksamheter till säkerhetslösningar. Det är därför nödvändigt att analysera säkerhetsfunktionerna inom WS-standarderna. Det är också angeläget att bygga vidare på grundfunktionerna. I föreliggande rapport beskriver vi en utbyggnad för att åstadkomma en säker och spårbar historik över alla identiteter som samverkat vid lösning av en uppgift. Med detta menar vi att identiteterna för all deltagande WS skall finnas i en datastruktur som är digitalt signerad och vars äkthet därmed kan verifieras. Det skall inte vara möjligt för någon WS att maskera sig i annans namn eller att dölja sin egen medverkan.

WS innehåller inga nya eller speciella säkerhetsfunktioner, utan arbetet går ut på att välja och anpassa lämpliga, befintliga, säkerhetsfunktioner och standardisera hur säkerhetsparametrar skall kommuniceras mellan WS. I [BEN] har vi redogjort för standarder för autentisering och identitetsverifiering. Där ges också en allmän beskrivning av grundkomponenterna inom WS. I föreliggande rapports avsnitt 3 ges en inte lika omfattande beskrivning, varför läsare vid behov hänvisas till exempelvis [BEN] eller [W3Cb].

Slutorden i [BEN] är att WS är en teknik i stark utveckling, men där också mycket återstår att utveckla. Beträffande autentisering över systemgränser har man kommit en bit på väg med autentisering mellan en mänsklig användare och en WS. Beträffande autentisering mellan WS är utvecklingen och standardiseringen i sin linda. Målet med det arbete som redovisas i föreliggande rapport är att ta fram en metod att bygga upp historik och spårning när flera WS samverkar för att lösa en uppgift. Metoden möjliggör bland annat att det på ett säkert och oavvisligt sätt går att spåra identiteterna hos samtliga WS som samverkat, enligt en viss vald samverkansmodell. I avsnitt 4 ”Samverkans- och interaktionsmodeller”, beskrivs egenskaper hos några olika samverkansmodeller. En av dessa utvecklas vidare, och metoden för historik och spårning beskrivs i avsnitt 5 ”Modell för spårning vid samverkande Web Services”. För

att lättare kunna illustrera de abstrakta teorierna om samverkan mellan WS använder vi på flera ställen i rapporten ett konstruerat exempel som följer nedan.

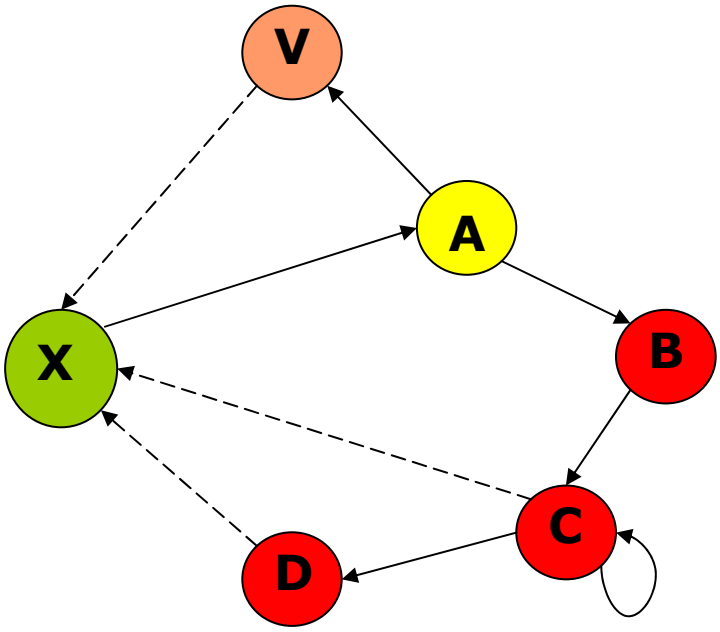
2.1 Scenarioexempel

Vi tänker oss att det finns ett stort antal klienter, en av dem kallas X, som vill ha uppgifter utförda. Det finns ett antal WS, färre än antalet klienter, som kallas A, B, C, ..., U, V. Låt oss anta att X vill ha utfört uppgiften ”Jag, X, behöver lägesbilden för NV-Skåne (angivet i koordinatform)”. Han ställer denna uppgift till A, som kan antas vara en WS som kan styrka X´s behörighet och som har kännedom om vilka tjänster som kan tänkas ge lägesbildsdata. A väljer att slussa ärendet vidare till V och B. V kan vara en luftövervaknings-central och får kanske meddelandet ”X, med adress si och så, behöver veta vilka flygföretag över Kattegat som har kurs mot NV-Skåne. Jag, A, styrker X´s behörighet”. B kan vara en spaningsresurs som kanske får frågan ”Vilka fientliga objekt finns inom området si och så? Meddela detta till X, vars behörighet jag styrker”. B kanske bedömer att han inte har resurser att besvara frågan, utan slussar den vidare till C. C tror sig veta att D har viss relevant information, och formulerar en fråga till D. Dessutom kan C själv ge information. Men det tar en viss tid, eftersom C först måste skicka ut en lokal spaningsresurs, innan C kan skicka sitt svarsbidrag till X.

Det är dessa identiteter – här kallade A, B, ..., V – som skall vara spårbara och omöjliga att manipulera. Särskilt viktigt är det att X, den som initierat uppgiften, kan verifiera identiteterna.

Scenariot kan illustreras med nedanstående figur, som återkommer senare i rapporten:

Figur I: ***Flödet mellan aktörer i scenario.***



3 Web Services

3.1 Allmänt

En försvenskad beteckning vore t ex webbtjänster, eller kanske internettjänster. Begreppet Web Services (WS) är dock så allmänt etablerat att det är meningslöst att använda en försvenskad beteckning. Som tidigare nämnts ges här inte någon uttömmande beskrivning av WS, utan här diskuteras främst de delar som har störst betydelse för samverkan mellan WS.

Det finns väldigt många aktörer inom utvecklingen av WS. Det finns därför skäl att kommentera de viktigaste aktörerna, samt standardiseringsläget. De två leverantörsoberoende sammanslutningar som har störst inverkan på utvecklingen av WS är Organization for the Advancement of Structured Information Standards [OASIS], samt World Wide Web Consortium, W3C, med dess undergrupp med inriktningen WS [W3Ca]. Av dessa arbetar W3C med den tekniska grundstandarden, som får samma status som övriga internetstandarder. OASIS inriktar sig mot tillämpningar, bl a säkerhet i affärstransaktioner. Deras standardförslag är plattforms- och leverantörsoberoende, men har inte samma definitiva status som W3C. Utöver dessa två organisationer finns sammanslutningar med större leverantörsdominans. Deras rekommendationer blir därför mindre definitiva. Ofta överlämnas deras rekommendationer så småningom till OASIS, i förhoppning om att man där skall rösta fram rekommendationen och fastställa den som standard. De två sammanslutningar som vi bedömt ha mest fokus på frågor liknande våra, samverkande WS, är [LIB] (med bla Sun) samt [WSSe] (med bl a IBM).

Ur [BEN] saxas följande sammanfattande beskrivning av WS:

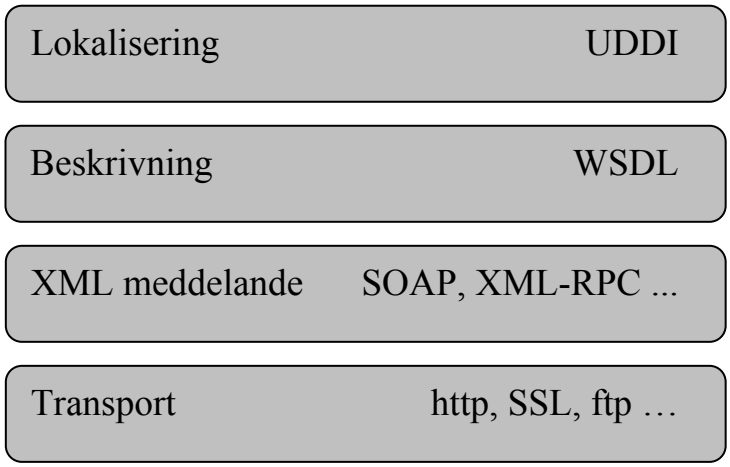
En WS är en programvaruagent som skall kunna anropas av en annan WS (också i flera steg), eller annan programvara. Anropet skall ske via ett meddelande som skall vara uppbyggt med hjälp av XML (se SOAP nedan). Meddelanden skall överföras via något standardiserat internetprotokoll.

Vidare skall varje WS ha en, globalt sett, unik identitet, URI, Uniform Resource Identifier. En WS skall kunna katalogiseras och lokaliseras (discovery) via en katalogtjänst (UDDI) som själv är specificerad med hjälp av XML. Metoderna att anropa en WS, samt typerna av data som ingår i anrops- respektive svarsmeddelanden, skall beskrivas med hjälp av XML (i en WSDL-fil).

Det som konstituerar WS är alltså

- En enhetlig standard, XML, för att bygga upp data och meddelanden. XML används också för att beskriva metadata, d v s beskrivningar av datatyper, objekttyper och -metoder, bindningar etc.

- Fyra stycken protokollsnivåer över TCP/IP-nivån enligt figur II.



Figur II: ***Protokollsnivåer inom Web Services*****.**

Några kommentarer beträffande dessa standardprotokoll:

Kommunikationen nod-nod, mellan två WS respektive mellan klienten X och en WS, sker lämpligen via https, d v s http plus SSL/TLS. Då får man viss säkerhet via kryptering och autentisering. Detta är ju dock enbart på kommunikationsnivå. Autentiseringen följer inte med meddelanden upp på applikationsnivå, och den kan inte verifieras där.

Vi förutsätter att läsaren är bekant med XML. Huvudvikten i rapporten kommer att läggas på XML-utbyggnader som har mest betydelse för samverkande WS. För att inte drunkna i XML´s ”pratighet” kommer vi ibland att använda principskisser, som inte är well-formed XML.

De två övre nivåerna, UDDI respektive WSDL, är standarder som beskriver hur man skall kunna hitta, lokalisera, en WS respektive hur man skall kommunicera med en WS. Grovt förenklat beskrivs UDDI (Universal Description, Discovery and Integration) ofta som en katalogtjänst, liknande vita och gula sidorna. Ett företag som har utvecklat en WS registrerar sig i en UDDI, där det i standardiserat och maskintolkbart XML-format publiceras data om företaget och dess WS. Nästa nivå inom WS är beskrivningen WSDL (Web Service Definition Language). Detta är en XML-grammatik för att beskriva en WS,

avseende vilka meddelandetyper man kan sända till respektive ta emot från WS, vilka kommunikationsprotokoll och adresser som WS är bunden till etc. När vår modell för samverkan, se avsnitt 6, är implementerad bör en WSDL-fil upprättas. Men innan dess avstår vi från att spekulera över WSDL.

3.2 Digital signatur

De grundläggande säkerhetsfunktionerna inom WS är, liksom vid all informationshantering, den digitala signaturen och kryptering. Digital signatur säkerställer ett meddelandes originalitet och kan verifierbart och obestridligt knyta ihop ett meddelande med dess utfärdare. Det är därför den digitala signaturen som är användbar när det gäller identitetsverifiering, autentisering. Kryptering är grundläggande för åtkomstkontroll och för att säkerställa sekretess.

De grundläggande standarderna för WS-säkerhet anger inte tillvägagångssättet för signering respektive kryptering. De beskriver alltså inte algoritmer, nyckellängder etc utan de beskriver hur man i XML-element paketerar dels de data som signerats respektive krypterats, dels anger vilka metoder man har använt. Det är ju väsentligt att detta görs på ett entydigt sätt så att konsumenten kan verifiera signaturen respektive dekryptera meddelandet. Det bör understrykas att WS-standarderna ligger på applikationsnivå. De kompletteras ofta av standarderna på lägre nivåer, främst TLS/SSL på transportnivå och IPSEC på nätnivå.

WS-standarden för digital signatur är XML-Signature Syntax and Processing, [W3Cd]. Det intuitivt naturliga sättet att komplettera ett meddelande med en XML-Signature är t ex

```
<?xml version='1.0' ?>
<meddelande>
   <dataelement1> … </dataelement1>
   <dataelement2> … </dataelement2>
   <ds:Signature
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      ...
   </ds:Signature>
</meddelande>
```

där det inom elementet `ds:Signature` finns underelement som beskriver vilket, eller vilka, dataelement som har signerats och hur det gått till. Namnrymden `xmlns:ds="http://www.w3.org/2000/09/xmldsig#"` är den som gäller den senaste standarden XML-Signature. Skissen ovan är en av tre specificerade sätt att paketera signaturen. Det kallas enveloped, vilket innebär att signaturen finns inuti meddelandet enligt ovan. Ett annat sätt kallas enveloping, vilket innebär att meddelandet finns inuti signaturen. Bildligt talat skiftar mär-

kena `<meddelande>` och `<ds:Signature>` plats. Det tredje sättet kallas detached. Detta innebär att signaturen är helt frikopplad från dataelementen. Inuti `ds:Signature` finns då referenser i form av URIer till de aktuella datameddelandena, som i princip kan finnas var som helst. I standarden WS-security, se avsnitt zz, anges hur man skall digitalt signera element i SOAP-meddelanden. Det enda sätt som tillåts i WS-security är detached signature, varför vi utgår från att detta sätt används i vår modell.

Nedanstående skiss över element som kan eller måste finnas inom en detached `ds:Signature` är hämtat från [W3Cd].

```
<Signature ID?>
  <SignedInfo>
    <CanonicalizationMethod/>
    <SignatureMethod/>
    (<Reference URI >
      (<Transforms>)?
      <DigestMethod>
      <DigestValue>
    </Reference>)+
  </SignedInfo>
  <SignatureValue>
 (<KeyInfo>)?
 (<Object ID?>)*
</Signature>
```

Kursivering har valts eftersom meddelandet inte är well-formed utan bara en principskiss. Tecknen +, ? respektive * anger att elementet måste förekomma 1 eller flera gånger (+), kan förekomma 0 eller 1 gång (?), respektive kan förekomma 0 eller flera gånger (*).

De två obligatoriska elementen i signaturen är enligt ovan *`SignatureValue`* och *`SignedInfo`*. I *`SignatureValue`* läggs den digitala signaturen. Eftersom denna är ett binärt tal måste den kodas till en teckenrepresentation. Detta skall ske via base64-kodning, inga alternativ är tillåtna. Inom *`SignedInfo`* läggs de underelement som signeras. Observera att det inte bara är dataelement (som refereras via *`Reference URI`*) som signeras, utan också flera andra element.

`CanonicalizationMethod` är ett viktigt obligatoriskt element som anger vilken metod som används för kanonisering. Ett och samma XML-meddelande kan vara utformat på olika sätt och likväl vara valid. Blanktecken och ny rad kan läggas till valbart, vissa element kan komma i godtycklig ordning, namnrymder kan anges explicit eller implicit etc. Signering respektive verifiering måste behandla elementen på exakt samma sätt, annars kan inte signaturen verifieras. Detta kallas kanonisering.

Det obligatoriska *SignatureMethod* anger vilken metod för digital signatur som använts. Metoden är valbar, men för att följa standarden är det ett krav att kunna använda DSS, Digital Signature Standard.

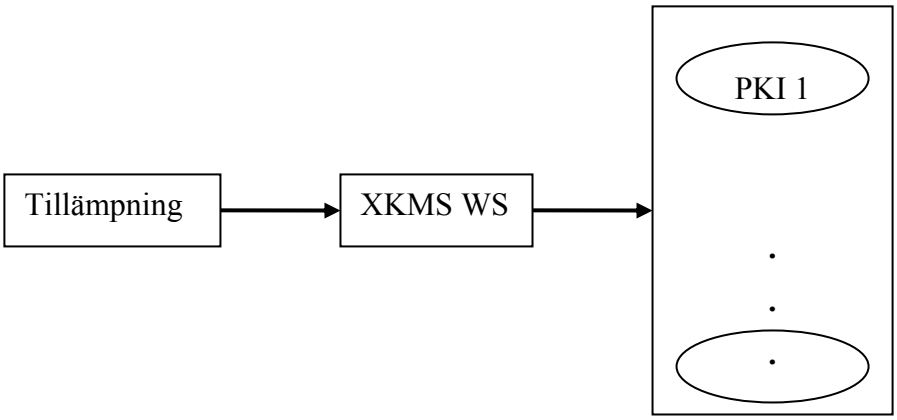
Reference pekar på de dataelement som signeras. Elementet finns en eller flera gånger, man kan alltså i en och samma signatur signera olika delar av ett meddelande. Man kan till och med peka på helt externa data, *URI* kan i princip peka vart som helst. *Transforms* anger eventuella transformationer av det aktuella dataelementet, t ex komprimering. *DigestMethod* anger vilken metod för beräkning av hashvärde som använts. Det är ett krav att kunna använda SHA1. Slutligen finns hashvärdet, base64-kodat, i *DigestValue*. Det är ju detta/dessa värden som, tillsammans med ovan nämnda element, slutligen signeras.

I *KeyInfo* läggs information om den nyckel som konsumenten kan verifiera signaturen med. Om inte elementet anges, måste alltså konsumenten på annat sätt känna till verifieringsnyckeln. De underelement, med datastrukturer, som finns definierade i standarden är `DSAKeyValue`, `RSAKeyValue`, `X509Data`, `PGPData`, `SPKIData`, `MgmtData` och `RetrievalMethod`. Namnen torde vara självförklarande, bortsett från de två sista. `MgmtData` kan användas t ex för att bygga upp en gemensam Diffie-Hellmannyckel. Men standarden rekommenderar andra sätt att göra detta. `RetrievalMethod` används om man behöver peka på externa data, t ex för att konstruera kedjor av certifikat.

Elementet *Object* är valbart. I fallet Enveloping Signature, när dataelementen finns inkapslade inuti *Signature*, läggs dataelementen i *Object*.

3.3 XKMS

XKMS (XML Key Management Specification), är en nära förestående (Candidate Recommendation) [XKMS] standard för en typ av WS som skall tjäna som tjänst gentemot olika PKI-system. PKI-hantering har visat sig vara komplex, och tanken är att XKMS skall avlasta tillämpningarna från de olika momenten.



Figur III: ***Relationen mellan tillämpningsprogram, WS och PKI-lösningar*****.**

Fråge/svarsprotokollen mellan en tillämpning och en XKMS WS består av två delar.

- Key Registration Service, som används när en aktör vill skapa och registrera nycklar och certifikat, vid ändringsbegäran, revokering (ogiltigförklaring), etc
- Key Information Service, som används när man vill fråga efter nycklar och certifikat, vill ha besked om giltighet etc.

Det moment som är viktigast i vår modell är verifiering av signatur för att otvetydigt knyta ihop ett meddelande med en identitet på den som uppger sig ha skapat meddelandet. Vi är alltså beroende av en Key Information Service som binder ihop verifieringsnyckel och identitet. Ur [XKMS] har vi hämtat nedanstående exempel på ett svar, erhållet från XKMS WS.

```
<?xml version="1.0" encoding="utf-8"?>
<LocateResult
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I04cd4f17d0656413d744f55488369264"
      Service="http://test.xmltrustcenter.org/XKMS"
ResultMajor="Success"
      RequestId="I045c66f6c525a9bf3842ecd3466cd422"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <KeyBinding Id="I012f61e1d7b7b9944fe8d954bcb2d946">
    <ds:KeyInfo>
      <ds:KeyValue>
        <ds:RSAKeyValue>
<ds:Modulus>0nIsmR+aVW2egl5MIfOKy4HuMKkk9AZ/IQuDLVPlhzOf
gngjVQCjr8uvmnqtNu8HBupui8LgGthO6U9D0CNT5mbmhIAErRADUMIA
Fsi7LzBarUvNWTqYNEJmcHsAUZdrdcDrkNnG7SzbuJx+GDNiHKVDQggP
BLc1XagW20RMvok=</ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
    <KeyUsage>Signature</KeyUsage>
    <KeyUsage>Encryption</KeyUsage>
    <KeyUsage>Exchange</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2633"
          Identifier="alice@alicecorp.test" />
  </KeyBinding>
</LocateResult>
```

Exemplet visar ett tänkbart svar från en XKMS WS på en förfrågan (som förmodligen skickats i ett meddelande märkt `LocateRequest`) efter en nyckel tillhörande en viss identitet. Inom elementet `KeyBinding` anges en RSAnyckel som binds till `alice@alicecorp.test`. Nyckeln är i detta fall en öppen nyckel som kan användas både för verifiering av signatur, för kryptering och för nyckelutbyte (vilket kan kritiseras, samma nyckel bör inte användas till flera uppgifter). Speciellt är elementet `UseKeyWith Application="urn:ietf:rfc:2633" Identifier="alice@alicecorp.test"` relevant för vår modell. Nyckeln binds till identiteten `alice@alicecorp.test` i tillämpningen e-mail (specificerad i RFC2633). I vårt fall är det nödvändigt att de olika aktörerna har tillgång till giltig nyckel för att verifiera digitala signaturer av data som innehåller spårbarhetsinformationen. Ett naturligt alternativ är att XKMS tillhandahåller verifieringsnycklarna och man kan tänka sig att peka ut vår tillämpning `Application=ngt_inom_FMLS` och `Identifier=ngt_i_stället_för_X,A,B_etc` . Det bör kanske påpekas att kommunikationen mot XKMS måste autentiseras. Antingen genom att vi använder SSL/TLS, eller hellre genom att XML-elementen digitalt signeras av XKMS WS. Vid implementationen av vår modell utgår vi ifrån att vi har till-

gång till någon XKMS WS så att våra identiteter är bundna till verifieringsnycklar.

3.4 SOAP

SOAP är en standard för paketering av XML-meddelanden och bindning av paket(en) till kommunikationsprotokoll. (Det finns också andra sådana standarder – XML-RPC, ebXML m fl).

Den senaste standarden är SOAP V1.2 [W3Cc] från juni -03, namnrymder `xmlns:soap-env="http://www.w3.org/2003/05/soap-envelope"` , `xmlns:soap-encoding="http://www.w3.org/2003/05/soap-encoding"` , `xmlns:soap-rpc="http://www.w3.org/2003/05/soap-rpc"`. I V1.2 påtalar man att SOAP inte skall ha någon uttydning, eftersom protokollet handlar om annat än objektåtkomst. Tidigare har SOAP uttytts som Simple Object Access Protocol. Nedanstående exempel är hämtade från [W3Cc].

Ett SOAP-meddelande skall, enligt reglerna för XML, bestå av ett enda element, ett `Envelope`. Inom detta kan det finnas ett element, `Header`, och det måste finnas ett element, `Body`. Inom såväl `Header` som `Body` får det finnas flera underelement.

Elementet `Body` innehåller de data som skall skickas från producent till konsument. Med konsument menas den slutliga WS som data är avsedd för. Det skall nämligen vara möjligt att data slussas via en serie mellanliggande WS, bl a för brandvägg eller andra kontroller, loggning m m. Elementet `Header` skall komma först. Däri läggs bl a parametrar som rör data i `Body`, t ex digitala signaturer. Likaså måste parametrar avsedda för mellanliggande WS ligga i `Header`. Attributet `role` (olyckligt valt namn) är till för tolkning i mellanliggande WS. Attributet `mustUnderstand` är ett sätt att explicit tvinga fram felhantering, vilket inte är standard.

```
<?xml version="1.0" ?>
 <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
envelope">
   <env:Header>
     <p:oneBlock xmlns:p="http://example.com"
     ...
     ...
     </p:oneBlock>
     <q:anotherBlock xmlns:q="http://example.com"
         env:mustUnderstand="true"
       env:role="http://www.w3.org/2003/05/soap-
envelope/role/next">
     ...
     ...
     </q:anotherBlock>
   </env:Header>
   <env:Body >
```

```
      ...
      ...
    </env:Body>
  </env:Envelope>
```

Envelope skall nu överföras via kommunikationsprotokoll, vanligast http. Man knyter ihop (begreppet binding [W3Cc]) i princip enligt exempel nedan. SOAP-standarden anger två sätt att binda till http, via `POST` som är Request/Response med indata/resultatdata, respektive via `GET` som bara hämtar resultatdata.

```
POST /Reservations HTTP/1.1
Host: travelcompany.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
envelope" >
 <env:Header>
   ...
 </env:Header>
 <env:Body >
     ...
</env:Body>
</env:Envelope>
```

Vår modell för samverkande WS via envägsmeddelanden, avsnitt 4, är tänkt att implementeras med hjälp av SOAP via http. Meddelandena skickas via POST, där svaret bara är en OK-bekräftelse på att meddelandet är mottaget. I SOAP-body finns informationen i de meddelanden som skickas mellan WS, respektive svaret till X. I SOAP-headern finns de statusdata som behövs i vår modell. Dels är det de tillståndsdata som gör det möjligt att bygga upp tillståndsträdet, se avsnitt 5. Dels är det digitala signaturer som gör att spårning och historik över samverkande WS blir säker.

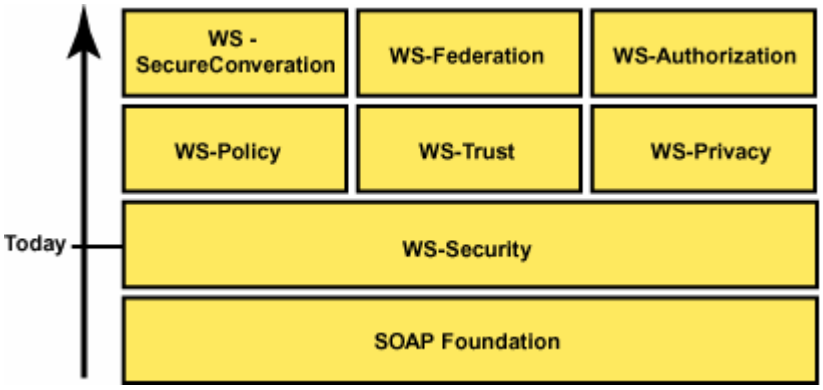
Det finns ett antal förslag till standarder för hur man lägger in element i SOAP-header för olika ändamål. Den viktigaste av dessa är den för säkerhet, WS-Security. Den beskriver bl a hur man lägger in signaturer av SOAP-element. Vi avser att anpasssa oss till WS-Security.

3.5 WS-Security

IBM, Microsoft, Verisign, BEA och RSA har gemensamt tagit fram en specifikation, WS-Security [WSSe]. Den handlar om hur man paketerar in krypterade

och signerade dataelement, jfr XML Encryption respektive XML Signature ovan, i SOAP-meddelanden. Speciellt finns detaljerade scheman för att paketera Kerberos biljetter, X.509-certifikat och SAML-assertions. Specifikationen har överlämnats till OASIS i syfte att standardiseras till en OASIS-standard.

I tillägg till grundspecifikationen WS-Security har de fem företagen gått vidare med utbyggnader för olika WS-funktioner. Nedanstående figur är hämtad från [IBMa]. Notera att utbyggnaderna är under utveckling och föremål för granskning och revision. Today på tidsaxeln antyder standardiseringsläget som det var i april -02. Sedan dess har WS-Security fastställts som standard. Övriga delar är fortfarande förslag.



Figur IV: ***WS-Security med föreslagna utbyggnader***

WS-Policy, hur man beskriver sin WS vad gäller tillåtna attribut, algoritmer, certifikattyper etc.

WS-Trust, vilka typer av tilltro man kan bygga upp. Delar av WS-Trust handlar om certifikathantering och nyckelbindningar, jfr XKMS. Detta illustrerar läget inom WS. Från olika håll finns flera förslag på funktioner och standarder, som överlappar varandra.

WS-Privacy, hur man beskriver vilka sekretesskrav man har. Dessa är sedan tänkta att ingå i WS-Policy och WS-Trust.

WS-SecureConversation, hur man bygger upp hela säkrade sessioner mellan WS, trots att SOAP i grunden är tillståndslöst requets/response. Kan i viss mån liknas vid ”SSL mellan WS”.

WS-Authorization, hur man beskriver åtkomstkontroll. Detta har stora likheter med XACML, eXtensible Access Control Markup Language.

WS-Federation. Här finns några modeller av hantering av identiteter, pseudonymer med mera, över systemgränser.

Vår modell kan man också se som en påbyggnad av WS-security, i syfte att ackumulera tillståndläget vid samverkande WS. Det vore därför naturligt att försöka följa samma sätt för utbyggnad som man gjort i andra fall. De ovan nämnda utbyggnaderna är emellertid främst specifikationer över vilka olika typer av meddelanden som skall utväxlas för att uppnå en viss funktion. WS-Trust beskriver exempelvis olika meddelanden för att överföra bland annat certifikat och nycklar. Sådana meddelanden läggs i SOAP-body, vilket också specifikationerna säger. Data som anger status och tillstånd, som är tanken med vår modell, bör snarare läggas i SOAP-header. Vi har dock lagt våra data i SOAP-body under de första stegen i vår implementation.

WS-Security [WSSe] beskriver hur man i SOAP-header lägger till XML-element som behövs för att få den säkerhet som man vill uppnå enligt sin policy. Vi är mest fokuserade på digitala signaturer. Då säger WS-Security att av de tre typerna av digital signatur som standardiserats i XML-Signature (avsnitt 3.2) så tillåts enbart detached signature. I huvuddrag ser i WS-Security ett SOAP-meddelande med en digital signatur ut enligt:

```
<?xml version="1.0" encoding="utf-8"?>
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
xmlns:ds="...">
<S11:Header>
   <wsse:Security>
      <wsse:BinarySecurityToken
         ValueType="...#X509v3"
         EncodingType="...#Base64Binary"
         wsu:Id="X509Token">
         MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
      </wsse:BinarySecurityToken>
      <ds:Signature>
         <ds:SignedInfo>
            <ds:CanonicalizationMethod Algorithm=
                  "http://www.w3.org/2001/10/xml-exc-c14n#"/>
            <ds:SignatureMethod Algorithm=
                  "http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <ds:Reference URI="#myBody">
                  <ds:Transforms>
                     <ds:Transform Algorithm=
                     "http://www.w3.org/2001/10/xml-exc-c14n#"/>
                  </ds:Transforms>
                  <ds:DigestMethod Algorithm=
                     "http://www.w3.org/2000/09/xmldsig#sha1"/>
                  <ds:DigestValue>EULddytSo1...</ds:DigestValue>
            </ds:Reference>
         </ds:SignedInfo>
         <ds:SignatureValue>
            BL8jdfToEb1l/vXcMZNNjPOV...
         </ds:SignatureValue>
         <ds:KeyInfo>
            <wsse:SecurityTokenReference>
```

```
                    <wsse:Reference URI="#X509Token"/>
                </wsse:SecurityTokenReference>
            </ds:KeyInfo>
        </ds:Signature>
    </wsse:Security>
</S11:Header>
<S11:Body wsu:Id="myBody">
    <tru:StockSymbol xmlns:tru="http://www.fabrikam123.com/payloads">
    QQQ
    </tru:StockSymbol>
</S11:Body>
</S11:Envelope>
```

Exemplet är hämtat ur [WSSe]. Vi har återgett det in extensio för att visa hur ”pratiga” XML-standarderna blir, trots att exemplet inte alls är komplett. Det väsentligaste har vi markerat med färger. Blå färg illustrerar ett SOAP-meddelandes två delar, `Header` repektive `Body`. Grön färg är blocket för WS-Security. Röd färg illustrerar hur man kan referera mellan element med hjälp av identifierare, vilket krävs för detached signature. Referensen `URI="#myBody"` under `<ds:SignedInfo>` säger att det som signeras har identifieraren `wsu:Id="myBody"` (d v s elementet `S11:Body`). Referensen `URI="#X509Token"` under `<ds:KeyInfo>` säger att nyckeln för verifiering finns i elementet med identifieraren `wsu:Id="X509Token"`. I detta fall finns nyckeln i ett X509-certifikat som följer med meddelandet. Vi tänker oss snarare att verifierings-nycklar hämtas via XKMS. Det vi ser framför oss är att data för vår modell paketeras i XML-element som infogas under elementet `<wsse:Security>`. Vissa riktlinjer för hur man infogar XML-element till WS-Security finns i [IBMb].

4 Samverkans- och interaktionsmodeller

Utvecklingen av WS sker gradvis och kontinuerligt. Det första steget var i princip mindre utvidgningar av det traditionellt vanligaste sättet att hämta information via Internet – en mänsklig användare utnyttjar en webbläsare på sin dator, som client gentemot en webbserver, via ett fråge/svar-protokoll. Men visionen om Web Services sträcker sig betydligt längre än så. Man tänker sig att flera WS, helt automatiserat, anropar varandra för att i samverkan lösa en uppgift. Den drivande tillämpningen är e-handel. För att fullfölja ett inköpsärende måste många tjänster samverka. En order skall läggas. Flera olika leverantörer måste kanske anropas för att leverera olika komponenter. Flera ekonomiska transaktioner mellan olika konton måste först reserveras och sedan effektueras i samband med leveransgodkännande. Speditionsfirmor måste involveras m m. Speciellt vill vi understryka att det snarare är regel än undantag att digitala signaturer spelar en väsentlig roll. Det är viktigt att styrka identiteter, att kunna binda identiteter till transaktioner etc.

Samverkande tjänster finns sedan tidigare, men då i mer slutna arkitekturer. Visionen är att den öppna arkitekturen, WS, skall vara tillräckligt kraftfull för att möjliggöra avancerad samverkan mellan tjänster. Bland annat behövs då en standard, beskriven i XML, för de olika begrepp och funktioner som behövs för att entydigt beskriva samverkan. Detta är komplext, och målet kan inte sägas ligga nära, men ett första utkast till standard finns [Chor]. För det vi kallar samverkan har man myntat begreppet Choreography. Minsta byggstenen kallas Interaction, som är när ett meddelande överförs mellan två WS. Nedanstående saxas ur [Chor].

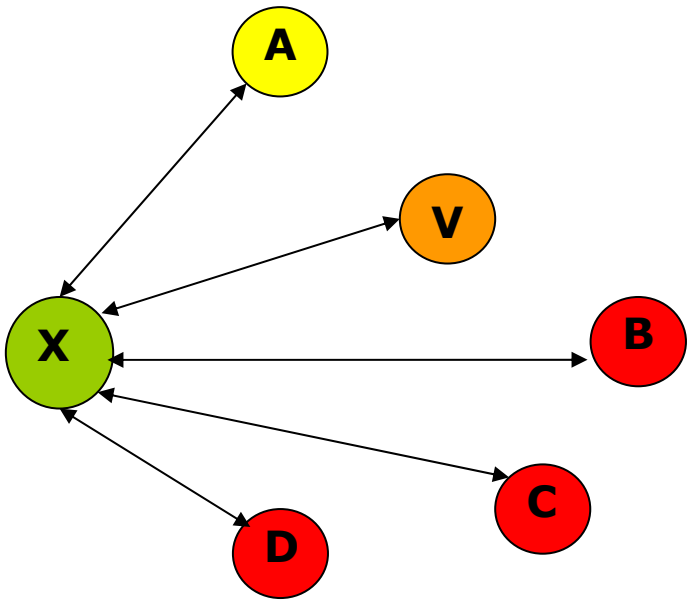
> An Interaction is the basic building block of a Choreography, which results in the exchange of information between parties and possibly the synchronization of their observable information changes and the values of the exchanged information.
>
> An Interaction forms the base atom of the recursive Choreography composition, where multiple Interactions are combined to form a Choreography, which can then be used in different business contexts.
>
> An Interaction is initiated when a party playing the requesting Role sends a request message, through a common Channel, to a party playing the accepting Role that receives the message. The Interaction is continued when the accepting party, sends zero or one response message back to the requesting party that receives the response message. This means an Interaction can be one of two types:
>
> - A One-Way Interaction that involves the exchanging of a single message
> - A Request-Response Interaction when two messages are exchanged

I det följande kommer vi att koncentrera oss på en interaktionsmodell, envägsmeddelanden (som förstås kan utgöra grunden även i tvåvägsmodeller). Först

kommenteras några egenskaper, för- och nackdelar, med tre alternativa modeller.

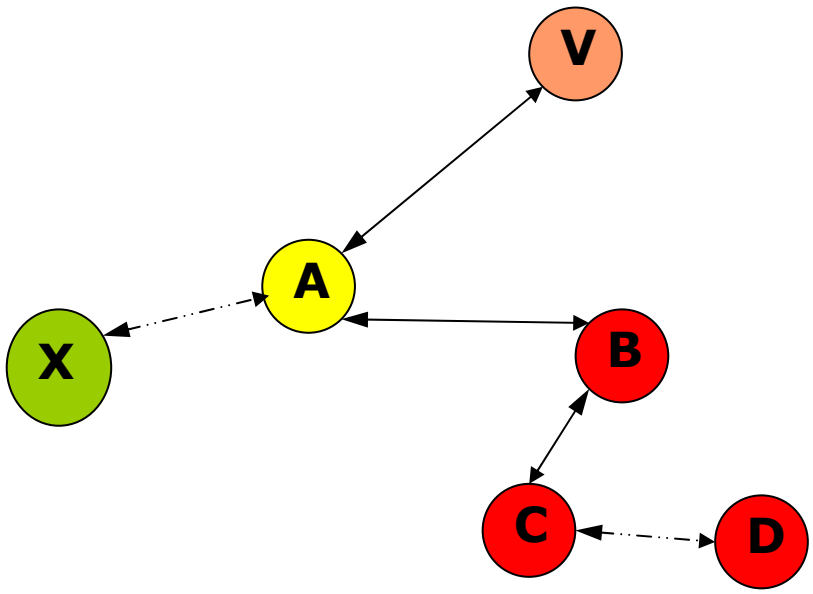
Låt oss också upprepa ett scenario från 2.1, mer militärt än ovanstående om e-handel. Det finns ett stort antal klienter, en av dem kallas X, som vill ha uppgifter utförda. Det finns ett antal WS, färre än antalet klienter, som kallas A, B, C, ..., U, V. Antag att X vill ha utfört uppgiften ”Jag, X, behöver lägesbilden för NV-Skåne (angivet i koordinatform)”. Han ställer denna uppgift till A, som är en WS som kan styrka X´s behörighet och som har kännedom om vilka tjänster som kan tänkas ge lägesbildsdata. A väljer att slussa ärendet vidare till V och B. V kan vara en luftövervakningscentral och får kanske meddelandet ”X, med adress si och så, behöver veta vilka flygföretag över Kattegat som har kurs mot NV-Skåne. Jag, A, styrker X´s behörighet”. B kan vara en spaningsresurs som kanske får frågan ”Vilka fientliga objekt finns inom området si och så? Meddela detta till X, vars behörighet jag styrker”. B kanske bedömer att han inte har resurser att besvara frågan, utan slussar den vidare till C. C tror sig veta att D har viss relevant information, och formulerar en fråga till D. Dessutom kan C själv ge information. Men det tar en viss tid, eftersom C först måste skicka ut en lokal spaningsresurs, innan C kan skicka sitt svarsbidrag till X.

Med detta scenario som exempel beskrivs nedan tre olika interaktionsmodeller, var och en med sina för- respektive nackdelar.



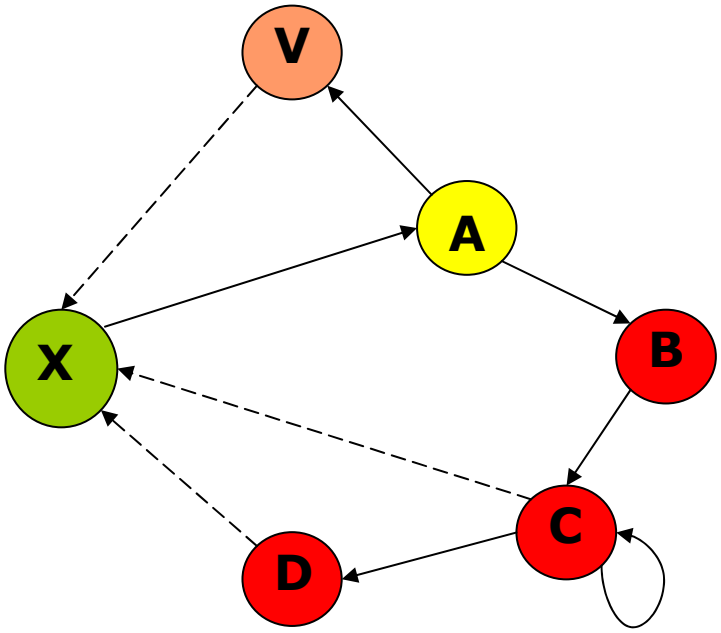
Figur V: ***Enkla meddelanden fråga/svar.***

Den första modellen är den traditionella klient/servermodellen, med en uppsättning enkla fråga/svar mellan X och de olika WS. Allt ansvar rörande vilka WS som skall kontaktas, i vilken ordning det skall ske etc, ligger på X, som har full kontroll över förloppet. X måste ha förmåga att styra interaktionerna, att formulera frågor utgående från svar från föregående WS, samt att baka in information från olika WS, exempelvis behörighetsförsäkran från A. Denna modell är den som har mest stöd från befintliga standarder, t ex kommunikationsprotokollet http.



Figur VI: ***Sammanhållna fråga/svar*****.**

En andra modell är att WS kontaktar nya WS och väntar med att besvara ursprungsfrågan tills man har fått in svar på sina egna frågor. Svaren skickas tillbaka samma väg som frågorna kom. Då avlastas X från bördan att synkronisera och styra interaktionerna. Denna börda hamnar i stället på WS, som måste hålla reda på utestående frågor, hantera fellägen etc. Streckprickningen i figuren anger ställen i scenariot där en anropande WS måste hålla reda på tillståndet i efterföljande led för att kunna sammanställa sitt svar.



Figur VII: ***Asynkrona envägsmeddelanden.***

Den tredje modellen innebär att varje WS skickar enbart envägsmeddelanden till efterföljande WS, respektive svar till X (streckade i figuren). Liksom i modell I kan WS göras helt tillståndslösa. Varje WS bedömer själv vilka åtgärder som skall vidtagas, vem som skall kontaktas med nya meddelanden etc. När en WS har vidtagit åtgärderna kan han i princip glömma bort ärendet. Det är X som har uppgiften att hålla reda på ärendets tillstånd, vilka WS som har involverats, time-out m m. Vissa lägen, när en WS behöver spara sitt eget tillstånd, kan modelleras som att WS skickar ett meddelande till sig själv. Ett sådant exempel finns i figuren när C behöver vänta på sin lokala spaningsresurs.

Det bör påpekas att envägsmeddelanden lämpligen skickas via ett standard tvåvägs kommunikationsprotokoll, t ex http. Man får då direkt ifrån kommunikationslagret en kort kvittens på att meddelandet kommit fram.

I följande tabell sammanfattas grovt några egenskaper hos respektive modell:

Tabell 1, Interaktionsmodeller

	Interaktionsmodeller		
	I, enkla meddelan-den	II, sammanhållna meddelanden	III, asynkrona envägsmeddelanden
WS hanterar till-stånd		x	
X hanterar till-stånd	x		x
X hanterar styrning	x		
WS hanterar styrning		x	x
kommunikation nod-nod	http, https (TLS)		
autentisering och behörighet	X-WS	WS-WS	WS-WS (delvis WS-X, svarsmeddelanden)

En egenskap, som skulle ha kunnat skilja mellan modeller, är den totala mängden kommunikation som åtgår. Intuitivt åtgår det mest kommunikation i modell I. Mängden meddelanden som måste skickas är dock beroende av tillämpningen, och vi avstår från att ha det som särskiljande egenskap.

Det beror på tillämpningen om man värderar en egenskap som positiv eller negativ. Det kan t ex värderas som positivt att X har all kontroll, enligt modell I. Men detta måste då vägas mot högre komplexitet i X. De tre modellerna skulle kunna sammanfattas sålunda:

> Alla tre modellerna kan realiseras med konventionella standardiserade kommunikationsprotokoll – http, eller hellre https (http via SSL/TLS) för att få säker nod-nod kommunikation.
>
> En viktig särskiljande egenskap är att modell II kräver att WS kan hålla reda på andra noders tillstånd. Detta medför en betydligt högre komplexitet. Det är också emot Internets princip, att hålla kommunikationen så tillståndslös som möjligt.

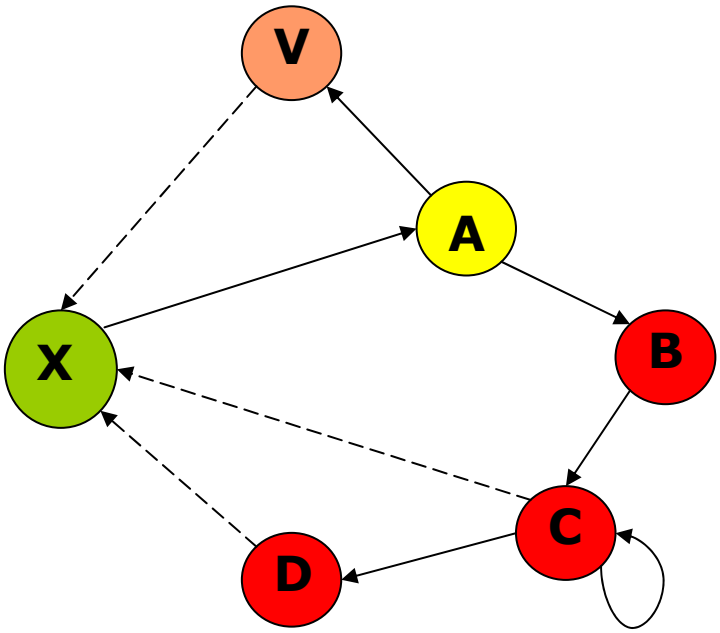
Det är därför modellerna I och III som vi bedömer som mest lämpade för att bygga samverkande system av system. Eftersom modell I lägger all börda på klienten X, är det den som kräver minst av standardiserad infrastruktur. Modell

I kan realiseras med idag befintliga WS-standarder. Modell III är den som är mest naturlig för att bygga upp ett system av samverkande tjänster. Men det finns inga WS-standarder som stödjer modellen. Vi menar att det är viktigt att sådana kommande standarder beaktar åtminstone två aspekter. För det första måste X kunna bygga upp tillståndet för kommunikationen. X måste kunna hålla reda på vilka WS som har blivit tillfrågade, från vilka X kan förvänta sig svar, time-out och fellägen etc. För det andra måste dessa tillståndsdata, via digitala signaturer, kunna verifieras av X. X måste kunna spåra vilka WS som deltagit i ett ärende. En komprometterad WS skall inte kunna dölja sig själv eller på annat sätt förfalska information. I avsnitt 5 beskriver vi en modell som uppfyller dessa båda krav.

5 Modell för spårning vid samverkande Web Services

5.1 Interaktionsmodellen

I avsnitt 4 diskuterade vi tre olika interaktionsmodeller vid samverkande WS. Vi motiverade också varför vi inriktar oss på den tredje modellen. Den är den som kräver minst av den anropande klienten X. I gengäld krävs mer av nyutvecklad standardisering inom WS, inte minst säkerhetsmässigt. Nedan upprepas delar av beskrivningen från avsnitt 4.



Figur VIII: ***Asynkrona envägsmeddelanden.***

Modellen innebär att varje WS skickar enbart envägsmeddelanden till efterföljande WS, respektive svar till X (streckade i figuren). WS kan göras helt tillståndslösa. Varje WS bedömer själv, ofta utgående från data från tidigare WS, vilka åtgärder som skall vidtagas, vem som skall kontaktas med nya meddelanden etc. När en WS har vidtagit åtgärderna kan den i princip glömma bort allt.

Det är X som har uppgiften att hålla reda på tillståndet, vilka WS som har involverats, time-out m m. Detta kan X göra enbart när X får data, vid de streckade anropen i figuren. Vissa lägen, när en WS behöver spara ett eget tillstånd, kan modelleras som att WS skickar ett meddelande till sig själv. Ett sådant exempel finns i figuren när C behöver vänta på sin lokala spaningsresurs, se scenarioexemplet i avsnitt 2.1.

Det bör påpekas att envägsmeddelanden lämpligen skickas via ett standard tvåvägs kommunikationsprotokoll, t ex http. Man får då direkt ifrån kommunikationslagret en kort kvittens på att meddelandet kommit fram.

5.2 Hierarkiskt träd

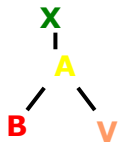
Målet är att bygga upp en datastruktur, kodad i form av XML-element, som möjliggör spårbarhet av vilka WS som är involverade. Låt oss först mynta begreppet ärende. Ett ärende är allt som tillhör den uppgift som X vill att de samverkande WS skall lösa. Det är naturligt att datastrukturen skall innehålla ett ärendenummer, som skall vara en unik identifierare av ärendet. Det är ärendenumret som hjälper X att avgöra vilka svarsmeddelande som hör ihop, och som gör det möjligt för X att bygga upp det hierarkiska träd som avspeglar ärendets tillstånd.

Grundtanken är att datastrukturen följer med meddelandekedjan för ärendet. Varje involverad WS adderar underelement i datastrukturen, som därmed blir strikt växande. Varje WS signerar hela datastrukturen. Därigenom kan X (och vid behov WS) rekursivt verifiera identiteterna för alla som varit involverade i meddelandekedjan.

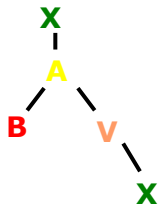
Eftersom målet är att åstadkomma en säker spårning av identiteter är det förstås identiteterna (X, A, B ... i vårt exempel, i verkligheten längre namn) som är de viktigaste elementen som varje WS adderar till datastrukturen. Varje nod adderar identiteten för den som anropat, för sig själv samt för de(n) som anropas. Med de konventioner som angavs i avsnitt 3.2 adderar varje WS elementen <*from*>, <*self*> och <*to*>+. I dessa element anges identitet för anropande WS, egen identitet, respektive identiteter för dem som kommer att anropas.

Delar av datastrukturen når X när X får svar från WS (streckat i figuren). X kan då bygga upp ärendets tillstånd i form av ett hierarkiskt träd.

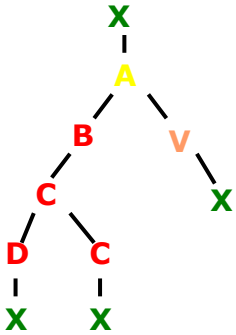
Exempel:



A skickar, till B och V, en datastruktur som i sin hierarkiska form har detta utseende. A signerar hela trädet.



Utseendet på det som V skickar till X. När X bygger upp detta träd ser han att det finns en oavslutad gren, via B. I detta läge kan X förvänta sig att det inkommer fler svar i ärendet. Kriteriet för att ärendet är avslutat är nämligen att alla löv i trädet är X. V signerar hela trädet, inklusive den signatur som A tillförde i föregående steg.



Utseendet på det slutliga, avslutade trädet. Alla löv är X.

Förutom de nämnda identiteterna <*from*>, <*self*> och <*to*>+ kan det också finnas andra element i datastrukturen. Ett naturligt element är tidsstämplar. En tidsstämpel tjänar två syften. Dels har X nytta av den när han skall bedöma hur länge han vill vänta på fler svar. X måste implementera någon time-out som utlöses när svar kommer bort, WS struntar i att svara etc. Dels försvårar tidsstämplar replay-attacker då uppsnappade data, med giltig signatur, används flera gånger.

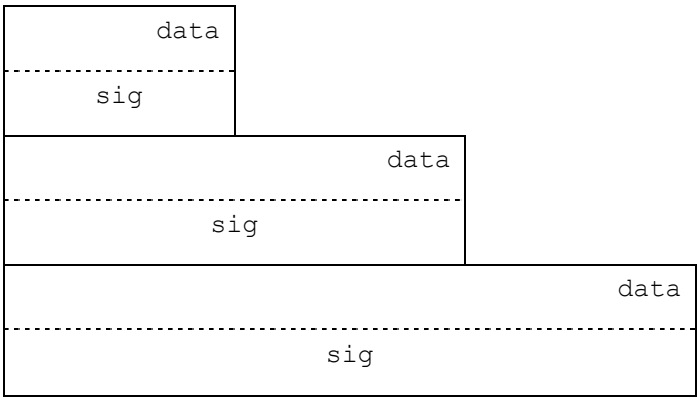
I och med att anroparen signerar vem som anropas (<*to*>+) och den anropade signerar vem som anropat (<*from*>) förhindras olika typer av man-in-the-middle attacker. Detta förstås under förutsättning att inte någon lyckats knäcka någon annans signeringsnyckel. Med en knäckt nyckel kan man alltid maskera sig. Inte heller kan en WS förneka sitt deltagande i kedjan. Man kan inte plocka bort sig själv, eftersom föregående WS har signerat mottagarna <*to*>+. Det går dock inte att hindra en WS från att dölja underaktiviteter. Vilken WS som helst kan ha frågat någon Q om information, utan att uppge detta. Likaså kan mycket väl C i exemplet ”glömma bort” anropet till sig själv.

6 Skiss över XML-struktur

Först en brasklapp. Vi är i färd med att implementera modellen i en WS-miljö, baserad på Java/Linux. Först efter slutförd implementation har vi riktigt klart för oss för- och nackdelar med olika alternativ att XML-koda datastrukturen. Nedanstående skiss är just en skiss, som kan komma att modifieras. Det är också bara en skiss över principen och inte fullständig XML-kod, som blir mycket mer omfattande och ”pratig”.

6.1 XML-skiss

I föregående avsnitt 5 beskrevs modellen. Citerat från 5.2 ”Grundtanken är att datastrukturen följer med meddelandekedjan för ärendet. Varje involverad WS adderar underelement i datastrukturen, som därmed blir strikt växande. Varje WS signerar hela datastrukturen.” Vår ansats är att packa in det som adderas av en WS i block. Ett sätt att grafiskt illustrera detta är enligt figur xx nedan, med tre block inritade. Signaturen `sig` betyder ”signering av allt som finns uppåt”.



Figur IX: ***Blockstrukturen i XML-meddelanden*****.**

WS nummer k, WS_k, skall addera data som den själv skapar, bl a <*from*>, <*self*> och <*to*>+, enligt 5.2. Dessutom skall WS_k signera sina egna data samt data och signaturer från de k-1 tidigare WS. Signaturen skall, enligt avsnitt 3.5,

vara detached. Den skall därför via referenser referera till de data som signeras. Huvudstrukturen för det som adderas av WS_k blir då:

```
<state>
   ...
   ...
   <block ...   ID="uuid_b_{k-1}">
       .....
   </block>
   <block ...   ID="uuid_b_k">
       <ws ...   ID="uuid_ws_k">
           data för WS_k (bl a from, self, to+)
       </ws>
       <signature>
           .....
             <Reference  URI="#uuid_b_0"> ... </Reference>
                 ......
             <Reference  URI="#uuid_b_{k-1}"> ... </Reference>
             <Reference  URI="#uuid_ws_k"> ... </Reference>
           .....
       </signature>
   </block>
</state>
```

`ID` ovan har inget med identiteten för någon WS att göra. `ID` är ett XML-attribut, i form av en identifierare, som andra XML-element skall kunna referera till. Denna identifierare måste vara unik, så att inte två element inom tillämpningen råkar få samma identifierare. Ett sätt att bilda unika identifierare beskrivs i [UUID], Universally Unique IDentifier.

Taggarna `<block ... ID="uuid_b_k"> ... </block>` omsluter det block som WS_k adderar. Blocket består av två delar, tillståndsdata respektive en digital signatur. Tillståndsdata från WS_k samlas inom taggarna `<ws ... ID="uuid_ws_k"> ... </ws>`. Här finns bland annat identiteterna *<from>*, *<self>* och *<to>+,* som de beskrevs i 5.2. Här finns också en tidstämpel av elementet. Här kan också finnas andra data som kan vara väsentliga. Exempelvis vore det tänkbart med referenser, i form av UUIDer, till WS_k´s in- och utdata i SOAP-body.

Det första blocket `<block ... ID="uuid_b_0"> ... </block>` har en särställning. Det skapas av den anropande klienten, X. Förutom identiteter och signatur, analogt med WS, skapar X också data som rör själva ärendet som sådant. En skissartad beskrivning av inledningen av datastrukturen `<state>` skulle kunna vara:

```
<state>
   <block ...   ID="uuid_b_0">
```

```
      <task ID="uuid_tasknumber">
         <client> X_id </client>
         <time> tidstämpel </time>
         .....
      </task>
   </block>
   <block ...   ID="uuid_b1">
      <ws ...   ID="uuid_ws1">
         <from> null </from>
         <self> X_id </self>
         <to> A_id </to>
         m m ....
      </ws>
      <signature>
         .....
           <Reference  URI="#uuid_b0"> ... </Reference>
              ......
           <Reference  URI="#uuid_ws1"> ... </Reference>
         .....
      </signature>
   </block>
   <block ... ID="uuid_b2">
      <ws ...   ID="uuid_ws2">
         <from> X_id </from>
         <self> A_id </self>
         <to> V_id </to>
         <to> B_id </to>
         m m ....
      </ws>
      <signature>
         .....
           <Reference  URI="#uuid_b0"> ... </Reference>
           <Reference  URI="#uuid_b1"> ... </Reference>
           <Reference  URI="#uuid_ws2"> ... </Reference>
         .....
      </signature>
   </block>
   .......
</state>
```

6.2 Implementation

Vår modell för spårning av identiteter, avsnitt 5, är en ny idé. Datastrukturen, avsnitt 6.1, är beskriven skissartad. För att verifiera att en idé är realistisk måste den, åtminstone i sina huvuddelar, implementeras i någon verklig utvecklingsmiljö. Vi har påbörjat en implementation, som vi räknar med att slutföra under 2005. Den skall bland annat resultera i WSDL-filer för modellen och XML-scheman för datastrukturen.

Avsikten, och den stora fördelen, med WS är att de skall vara plattforms- och språkoberoende. Det har också växt upp ett stort antal utvecklingsmiljöer för WS på olika plattformar och i olika språk. Det bör påpekas att utvecklingsmiljöerna oftast är omfattande, med flera lager abstraktioner i syfte att dölja underliggande detaljer för utvecklaren. Det kan därför ofta vara svårt och komplicerat att införa nya funktioner och standarder som inte finns färdiga i miljön.

Vi har valt att implementera i Linux med hjälp av Systinet Server for Java, mera känd som WASP [WASP]. Tonvikten ligger på interaktionsmodellen, samt signering och verifiering av datastrukturen. Vi har inte tillgång till någon XKMS-server som kan binda ihop identiteter med verifieringsnycklar. Vi kommer därför att lägga in verifieringsnycklarna direkt i koden.

7 Närliggande aktiviteter

Som nämndes i inledningen av avsnitt 3 är många organisationer och sammanslutningar aktiva vid utveckling och standardisering av WS. Det är det stora intresset att bygga en öppen infrastruktur för kommersiella tillämpningar, främst e-handel, som är drivande.

I avsnitt 4 diskuterade vi tre modeller för interaktionen dator-dator; I - Enkla meddelanden fråga/svar, II - Sammanhållna fråga/svar, respektive III - Asynkrona envägsmeddelanden. Som nämndes är det i första hand modell I som har varit grunden vid utvecklingen av samverkande WS. Modell I kan delas in i två undermodeller. Den först implementerade byggdes upp av synkrona frågor/svar. Klienten, X, skickar sin fråga till en WS. Sedan gör X inget förrän svaret från WS anlänt, då X skickar fråga till ny WS. Frågorna och svaren är synkroniserade till varandra. Denna modell är inte naturlig i alla tillämpningar. I [ASYN] förtecknas ett dussintal referenser till olika aspekter av asynkrona frågor/svar. Då väntar inte X in svaren i tur och ordning, utan kan ha flera obesvarade frågor ute samtidigt. Det vanligaste är att interaktionen mellan X och en WS byggs upp av två stycken envägsmeddelanden, ett för frågan och ett för svaret. Detta liknar vår modell, men med den stora skillnaden att hela förloppet styrs av X. Referenserna i [ASYN] är på olika nivåer. Grundläggande kommunikation, på samma nivå som http, är exempelvis [BEEP]. På hög nivå, workflow management, d v s Xs styrning av förloppet, är exempelvis [Wf]. Inget finns nämnt om vårt fokus – att spåra och verifiera identiteter hos deltagande WS. De säkerhetsfrågor som nämns rör mest reliability, att vara säker på att ett meddelande kommer fram på rätt sätt.

8 Slutord

Huvudsyftet med denna rapport har varit att beskriva en modell för spårning av identiteter vid samverkande Web Services. Modellen är ny, speciellt i den meningen att den modellerar en typ av samverkan, via asynkrona envägsmeddelanden, som ännu inte så ofta har implementerats. Det finns dock flera tillämpningar när denna samverkansmodell är naturlig. I rapporten används en sådan tänkbar tillämpning som ett scenarioexempel.

Motiveringen, för att över huvud taget studera säkerhetsfrågor avseende Web Services, är att dessa är en stark kandidat för att realisera den tjänsteorienterade arkitektur som har fastställts för FMLS.

Modellen innebär att de meddelanden, som skickas mellan Web Services för att utföra en uppgift, innehåller data med identiteter för samverkande aktörer. Data är signerade på ett sätt som förhindrar deltagarna att agera i annans namn eller att dölja sin medverkan.

Modellen är än så länge i idéstadiet. En implementation är påbörjad, men ej avslutad, för att se om modellen har praktiska brister.

Referenser

[ASYN] Asynchronous Transactions and Web Services
http://xml.coverpages.org/async.html

[BEN] Bengtsson A, Hunstad A & Westerdahl L, "Identitetsverifiering över systemgränser", Användarrapport
FOI-R--1025--SE, November 2003, FOI.

[BEEP] Blocks Extensible Exchange Protocol (BEEP),
http://www.ietf.org/rfc/rfc3080.txt

[CHOR] Web Services Choreography Description Language Version 1.0,
W3C Working Draft 12 October 2004
http://www.w3.org/TR/ws-cdl-10/

[IBMa] Security in a Web Services World: A Proposed Architecture and Roadmap,
http://www-106.ibm.com/developerworks/library/ws-secmap/

[IBMb] Specification: WS-Security Profile for XML-based Tokens,
28 August 2002,
http://www.ibm.com/developerworks/library/ws-sectoken.html

[LIB] LIBERTY ALLIANCE PROJECT WHITE PAPER –
Liberty Alliance & WS-Federation: A Comparative Overview
http:// projectliberty.org/resources/whitepapers/
wsfed-liberty-overview-10-13-03.pdf

[OASIS] Organization for the Advancement of Structured Information Standards
http://www.oasis-open.org/who/

[XKMS] XML Key Management Specification (XKMS 2.0)Version 2.0,
W3C Candidate Recommendation 5 April 2004
http://www.w3.org/TR/xkms2/

[UUID] Network Working Group, P. Leach, Microsoft, M. Mealling, VeriSign, Inc., R. Salz, DataPower Technology, Inc.,
“A UUID URN Namespace”, draft-mealling-uuid-urn-03.txt,
January 2004,
http://www.ietf.org/internet-drafts/draft-mealling-uuid-urn-03.txt

[W3Ca] Web Services Activity
http://www.w3.org/2002/ws/

[W3Cb] Web Services Architecture Requirements, Working Draft 19 August 2002
http://www.w3.org/TR/2002/WD-wsa-reqs-20020819

[W3Cc] SOAP Version 1.2 Part 0: Primer
http://www.w3.org/TR/soap12-part0

[W3Cd] XML-Signature Syntax and Processing
http://www.w3.org/TR/xmldsig-core/

[WASP] Systinet Server for Java,
http://www.systinet.com/products/wasp_jserver/overview

[Wf] XML-Based Workflow and Process Management Standards: XPDL, Wf-XML
http://xml.coverpages.org/wf-xml.html

[WSSe] Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)
http://xml.coverpages.org/WSS-SOAP-MessageSecurityV10-20040315.pdf