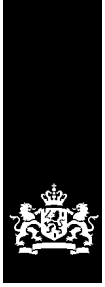


Ministerie van Binnenlandse Zaken en Koninkrijksrelaties

Best practice WUS

Digikoppeliing 3.0

Document versie 1.9

Datum 6 november 2014
Status Definitief

Colofon

Projectnaam	Digikoppeliing 3.0
Versienummer	1.9 Definitief
Organisatie	Servicecentrum Logius Postbus 96810 | 2509 JE Den Haag T 0900 555 4555 servicecentrum@logius.nl
Bijlage(n)	0

Inhoud

Inleiding

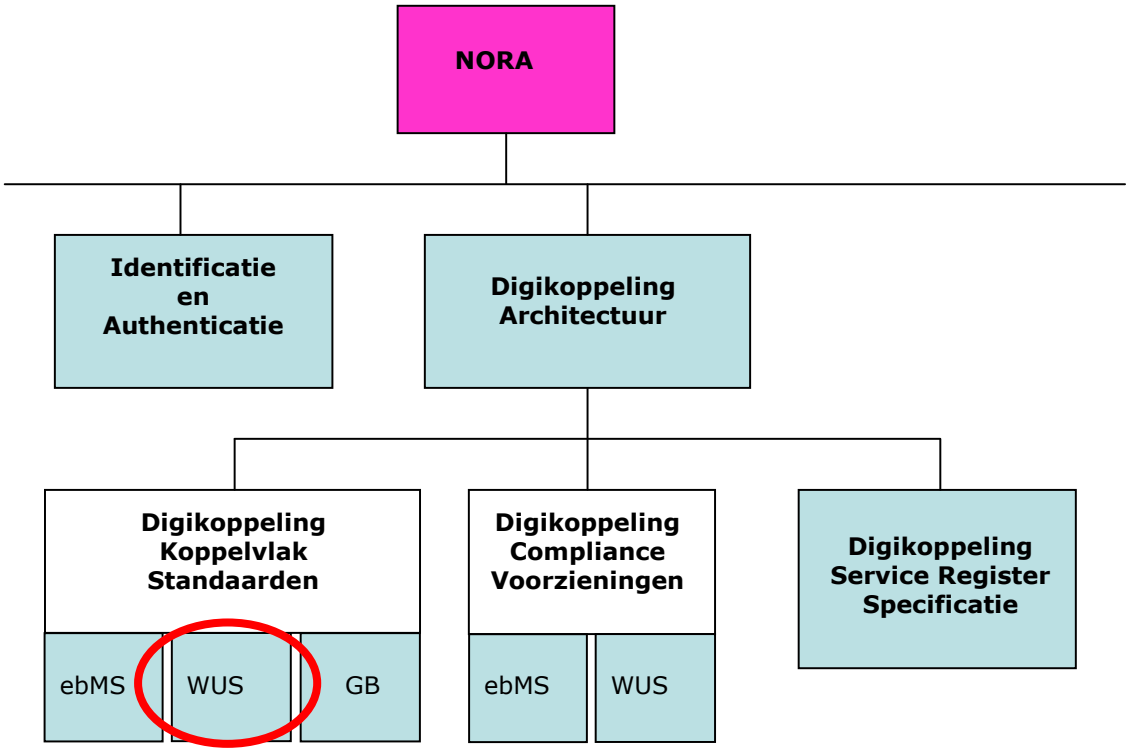
1.1 Doel en doelgroep

Alle Digikoppeling webservices die op WUS gebaseerd zijn, moeten conformeren aan Digikoppeling Koppelvlakstandaard WUS. Dit document is een aanvulling hierop. Het heeft als doel ontwikkelaars te adviseren en te informeren over de huidige werkwijze bij het toepassen van Digikoppeling Koppelvlakstandaard WUS – deze informatie geldt dus alleen voor de WUS-variant.

Het document is bestemd voor ontwikkelaars van webservices, die Digikoppeling toepassen. Het gaat hierbij om zowel service providers als service requesters (clients).

1.2 Opbouw Digikoppeling documentatie

Digikoppeling is beschreven in een set van documenten. Deze set is als volgt opgebouwd:



figuur 1 – Opbouw documentatie Digikoppeling

1.3 Digikoppeling

1.3.1 Doel en scope van Digikoppeling

Voor de Overheid als geheel is interoperabiliteit tussen een groot aantal serviceaanbieders en serviceafnemers van essentieel belang. Die grootschalige interoperabiliteit wordt bereikt door een sterke standaardisatie van het koppelvlak tussen de communicatiepartners.

Deze communicatie vindt plaats in het domein van Digikoppeling, en daarbij worden Digikoppeling koppelvlakstandaarden toegepast. Dat is een zeer beperkte set van standaarden waaruit onder gedefinieerde omstandigheden gekozen kan worden.

Digikoppeling biedt de mogelijkheid om op die sterk gestandaardiseerde wijze berichten uit te wisselen tussen serviceaanbieders (service providers) en serviceafnemers (service requesters of clients). Digikoppeling richt zich (in elk geval voorlopig) uitsluitend op uitwisselingen tussen overheidsorganisaties.

1.3.2 Uitwisseling binnen Digikoppeling

De uitwisseling tussen service providers en - requesters is in drie lagen opgedeeld:

- Inhoud: deze laag bevat afspraken over de inhoud van het uit te wisselen bericht, dus de structuur, semantiek en waardebereiken . Digikoppeling houdt zich niet met de inhoud bezig, "heeft geen boodschap aan de boodschap".
- Logistiek: op deze laag bevinden zich de afspraken betreffende transportprotocollen (HTTP), messaging (SOAP), adressering, beveiliging (authenticatie en encryptie) en betrouwbaarheid. Dit is de laag van Digikoppeling.
- Transport: deze laag verzorgt het daadwerkelijke transport van het bericht.

Digikoppeling richt zich uitsluitend op de logistieke laag. Deze afspraken komen in de koppelvlakstandaarden en andere voorzieningen.

1.4 Opbouw van dit document

Hoofdstuk 1 bevat een aantal algemene inleidende onderwerpen.
Hoofdstuk 2 bevat aanbevelingen, werkwijzes en Best Practices.

1.5 Gehanteerde terminologie: Digikoppeling Glossary

Voor de definities die binnen het Digikoppeling project gehanteerd worden, zie de 'Digikoppeling Glossary' die via Digikoppeling website te vinden is.

2 Werkwijze/Aanbevelingen/Best Practices

2.1 Servicedefinities

Deze paragraaf bevat de aanbevelingen t.a.v. het omgaan met servicedefinities.

WSDL

In WSDL 1.1 is een Authoring Style advies (zie hieronder) opgenomen: "separate the definitions in three documents: data type definitions, abstract definitions, and specific service bindings". Dit advies, met name het apart beschrijven van de "specific service bindings" (WSDL onderdelen Binding en Service) wordt overgenomen.

Onderstaande tekst is een citaat uit de WSDL 1.1 Standaard. Het vormt een door Digikoppeling overgenomen best practice bij het schrijven van duidelijke WSDL's.

> *Wijze van notatie (© W3C Note 15 March 2001, vertaald uit het Engels)*
>
> *"Door gebruik van het importelement wordt het mogelijk, de diverse elementen van een servicedefinitie te scheiden in afzonderlijke documenten, die dan naar behoefte geïmporteerd kunnen worden. Met behulp van deze techniek kun je duidelijker servicedefinities schrijven, omdat de definities gescheiden worden al naar gelang hun abstractieniveau. Het vergroot ook het vermogen om allerlei soorten servicedefinities nogmaals te gebruiken. Het gevolg is, dat WSDL documenten die op deze manier gestructureerd zijn, gemakkelijker te gebruiken en te onderhouden zijn. In het 2e voorbeeld hieronder is te zien hoe deze manier van schrijven gebruikt wordt om de service in het 1e voorbeeld* te definiëren. Hier scheiden we de definities in drie documenten: gegevenstype definities, abstracte definities en specifieke service bindings. Natuurlijk is het gebruik van dit mechanisme niet beperkt tot de definities uit het voorbeeld, dat alleen taalelementen gebruikt die in deze specificatie gedefinieerd zijn. Andere typen definities, die op meerdere taalextensies gebaseerd zijn, kunnen op dezelfde manier gecodeerd en hergebruikt worden."*

* Het voorbeeld waar deze tekst van de WSDL 1.1 standaard naar verwijst wordt is niet in overeenstemming met het Digikoppeling Koppelvlakstandaard WUS profiel (zie voor details CRF 12). Om een indruk te krijgen van de opdeling (authoring style) van een WSDL wordt verwezen naar Bijlage 1 van Digikoppeling Koppelvlakstandaard WUS.

Voor de specificatie van zaken die buiten het bereik van de WSDL vallen (TLS, WS-Security en WS-ReliableMessaging) wordt aanbevolen om in de WSDL van een service "documentation elements" (<wsdl:documentation> of <!-- xxx -->) op te nemen die de eisen ten aanzien van metadata verwoorden, of een verwijzing naar betreffende documenten bevat.

Karakterset en codering
Voor communicatie binnen het Digikoppeling WUS kanaal wordt de UCS karakterset (ISO/IEC 10646) gebruikt. Deze karakterset omvat (is superset van) de set Unicode, Latin (ISO/IEC 8859-x) en de GBA karakterset.

Bij berichtenverkeer speelt de gebruikte karakterset een belangrijke rol. Een serviceafnemer kan in de vraag aanroep karakters gebruikt hebben die niet door de serviceaanbieder ondersteund worden. Voor goede communicatie is het dus belangrijk dat hiervoor afspraken gelden. UCS is de meest uitgebreide karakterset. Toepassen van UTF-8 zorgt er voor dat efficiënt omgegaan wordt met het aantal bytes in het bericht.

Versie aanduiding
Er zijn een aantal elementen waaraan een versie aanduiding moet worden toegevoegd. Dit zijn:
WSDL/namespace
WSDL/Servicenaam
WSDL/PortType
WSDL/Type(s) (XSD) namespace

Er zijn een aantal manieren om de versie van een service aan te duiden. De meest gangbare zijn “Major.Minor”, “Enkelvoudige versie” (bijv V1) en “YYYY/MM”.

Het voorstel is om voor zowel de XSD als de WSDL de Enkelvoudige versie aanduiding te gebruiken.

Waarom gebruiken we geen major.minor? Er zijn verschillende mogelijkheden om Minor wijzigingen backward compatible te houden, deze worden echter als erg omslachtig beschouwd en/of ze vereisen speciale tooling ondersteuning. Daarom wordt voorgesteld geen onderscheid tussen major en minor te maken, en dus alleen met enkelvoudige versies te werken. Dit heeft als resultaat dat de WSDL en XSD namespace dus alleen de “Enkelvoudige” aanduiding, zoals _v1 krijgt.

Een aantal voorbeelden:
WSDL namespace “http://wus.osb.gbo.overheid.nl/wsdl/compliance-v1”
XSD namespace “http://
wus.osb.gbo.overheid.nl/xsd/compliance/xsd/compliance-v1”.
Servicenaam “OSBComplianceService_v1”
PortType “IOSBComplianceService_v1”

De aanduiding YYYY/MM slaat ook op een enkelvoudige versie, d.w.z. zonder onderscheid tussen major en minor. Die aanduiding kan dus ook gebruikt worden. Het lijkt echter aan te bevelen om versies van webservices aan te duiden met (oplopende) versienummers, omdat communicatie daarover iets eenduidiger is.
WB005
Voor het onderscheid tussen test- en productieservices heeft het de voorkeur dat deze twee op aparte machines komen met een eigen DNS naam (en dus met verschillende PKI overheid certificaten).

Aan de locatie (uri) van de service is daardoor te zien of het om een productie- of een testservice gaat.

XSD

Gebruik van document/literal wrapped style. In Digikoppeling Koppelvlakstandaard WUS staat de bij voorschrift WW003 dat bij de document literal style de body maar 1 element mag bevatten. Het wordt sterk aangeraden dat dit element de operatie naam bevat voor een bepaald bericht. Deze wordt dus door de xsd beschreven en bevat een beschrijving van de payload. Door deze methode te gebruiken wordt de interoperabiliteit verhoogd, met name tussen Microsoft en andere omgevingen. (zie http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/)

Wsdl definition

```
…
<types>
    <schema>
        <element name="myMethod">
         <complexType>
             <sequence>
               <element name="x" type="xsd:int"/>
                     <element name="y" type="xsd:float"/>
             </sequence>
         </complexType>
        </element>
        <element name="myMethodResponse">
          <complexType/>
        </element>
    </schema>
</types>
<message name="myMethodRequest">
    <part name="parameters" element="myMethod"/>
</message>
<message name="empty">
    <part name="parameters" element="myMethodResponse"/>
</message>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>
…
```

bericht:

```
…
soap:envelope>
    <soap:body>
        <myMethod>
           <x>5</x>
           <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
….
```

Namespaces
Met betrekking tot het verkrijgen van eenduidigheid in de WSDL bestanden, wordt sterk aangeraden dat xml namespace prefixen volgens de WSDL 1.1 specificatie gebruikt worden.
(http://www.w3.org/TR/wsdl)"1.2 Notational Conventions").

Het heeft de voorkeur dat een namespace wordt opgebouwd op basis van de domeinnaam van de web service.

Naamgeving conventies
Over het algemeen moet de service naam een goede weerspiegeling zijn van de context waarin de service wordt gebruikt. De operatienamen die door deze service ondersteund worden, moeten passen binnen de context van de service en overdadige lange tekststrings moet worden voorkomen.

Contract First vs Contract Last

Het ontwikkelen van webservices kan grofweg ingedeeld worden in twee categorieën, namelijk:

- contract first
- contract last

Met 'contract' wordt gewezen op het WSDL contract dat de webservice definieert. Nagenoeg alle webservice toolkits ondersteunen beide manieren van ontwikkeling.
Bij een contract first approach wordt in feite gestart met het handmatig samenstellen van het WSDL. Gebaseerd op het WSDL wordt vervolgens door toolkits de code gegenereerd. De ontwikkelaar maakt gebruik van de genereerde code om vervolgens een implementatie te schrijven. Hierbij maakt de ontwikkelaar gebruik van technieken als interfaces en overerving.
Bij Contract last approach is dit proces net andersom. Een ontwikkelaar begint met het ontwikkelen van de logica en het schrijven de code. Daarna voegt de ontwikkelaar meta informatie toe aan de code waarmee bepaalt wordt hoe de WSDL eruit ziet. Meta informatie verschilt per toolkit/framework hoe dit gedaan wordt. Binnen de Java wereld wordt dit vaak gedaan met jaxws annotations. Bij het opstarten van de applicatie wordt vervolgens door de toolkit bepaald welke meta informatie het tot zijn beschikking heeft en zal vervolgens op runtime een WSDL publiceren. Beide manieren hebben hun voor en nadelen. Ervaring leert wel dat bij trajecten waarbij strikte eisen worden gesteld met betrekking tot de WSDL dat een contract first approach uiteindelijk de meest robuuste manier is. Doordat een WSDL volledig handmatig wordt samengesteld, is dit ook de meest flexibele en toolkit-onafhankelijke manier van werken.

2.2 Foutafhandeling

Bij berichtenuitwisseling tussen een service requester en service provider kunnen fouten optreden. Het is van belang dat er nagedacht wordt over het nut van het communiceren van een bepaalde foutmelding. De beschrijving van een of andere interne fout bij een service provider zal in het algemeen niet interessant zijn voor een requester; het terugmelden van de specifieke oorzaak heeft dan geen zin.

Als de oorzaak van de fout bij de service requester ligt, dan dient de foutmelding er op gericht te zijn dat de service requester op basis van de foutmelding de fout kan achterhalen en actie ondernemen.

Voor communicatie in het Digikoppeling domein volgens de Koppelvlakstandaard WUS zijn vier verschillende fouttypen te onderkennen: protocolfouten, transportfouten, functionele fouten en technische fouten.

Protocol- en transportfouten (dus TLS of HTTP fouten).
Protocol- en transportfouten zijn in het algemeen in het protocol gedefinieerd. Wijzigingen daarin - dus aanpassing van standaard software - zijn niet wenselijk. Protocol- en transportfouten worden daarom niet beschreven. De hier beschreven foutmeldingen hebben betrekking op situaties waarin het requestbericht door de web service is ontvangen. Deze kan het echter niet goed verwerken en stuurt daarom een foutmelding terug.

Functionele fouten
Functionele fouten zijn in het kader van Digikoppeling moeilijk te standaardiseren. Deze zullen voor veel organisaties verschillen en ook het communiceren van de foutmelding zal niet altijd eenduidig zijn. Dit is weliswaar iets wat om aandacht vraagt, maar het valt buiten de scope van Digikoppeling.

Technische fouten
Voor technische foutmeldingen kan een standaard bericht gedefinieerd worden. In de SOAP specificatie is de SOAP Fault beschreven die je hiervoor goed kunt gebruiken.

Communiceren van een fout via een SOAP Fault heeft een aantal voordelen:
Uitzonderingen op een consistente manier afgehandeld worden;
De SOAPFault wordt beschreven in de SOAP specificatie;
De verschillende elementen waaruit een SOAP Fault is opgebouwd biedt de mogelijkheid tot het communiceren van uitgebreide informatie;
De FaultCode kan aanduiden of de fout was veroorzaakt door Client of Server.

Een aantal nadelen zijn:
Soapfaults kunnen geen binding (HTTP) gerelateerde fout communiceren. In dat geval wordt over het onderliggende protocol de fout gecommuniceerd
Bij een SOAPFault bericht mag geen additionele data toegevoegd worden

Het ‘detail’ element van de SOAP Fault is bedoeld om applicatie specifieke foutmeldingen te communiceren die gerelateerd zijn aan het SOAP ‘Body’ element. Het moet aanwezig zijn in de SOAP Fault indien de fout werd veroorzaakt door het ‘Body’ element. Indien er geen ‘detail’ element in de SOAP Fault aanwezig is, dan betekent dit dat de fout niet is ontstaan bij het verwerken van het ‘body’ element.

Voor een web service in het Digikoppeling domein moeten foutmeldingen gedefinieerd worden. Neem die foutmeldingen op in de publicatie van de service in het Digikoppeling Service Register.

2.3 WS-Addressing

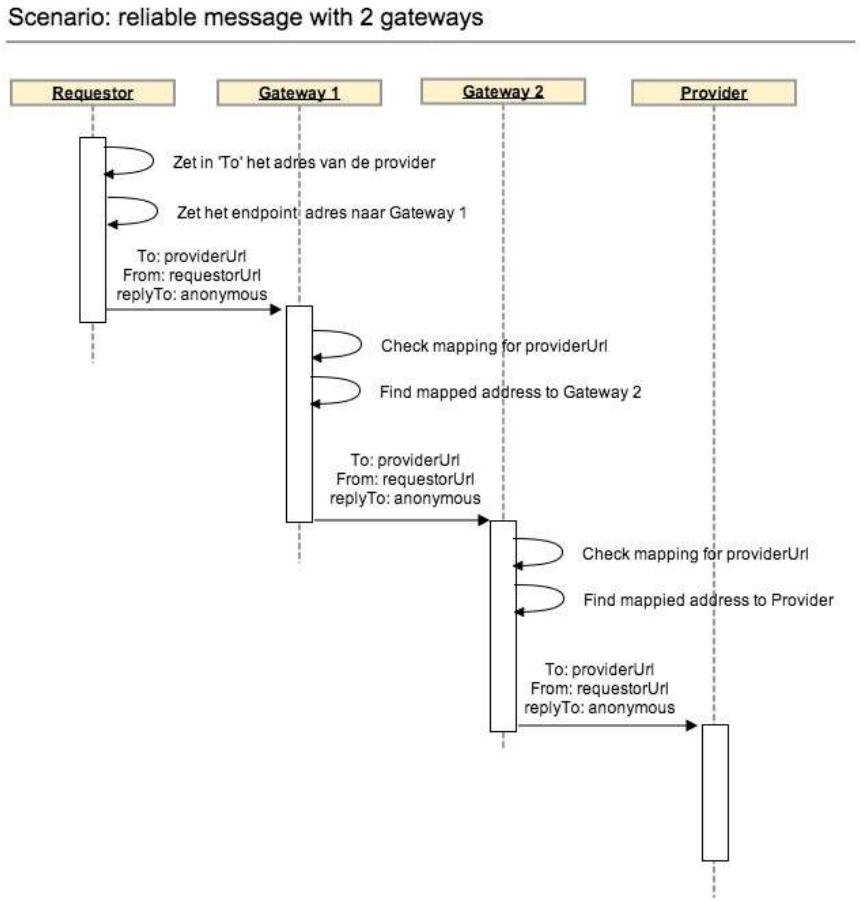
Maak gebruik van MessageID voor asynchroon verkeer

Binnen WS-Addressing wordt de wsa:MessageID gebruikt om een bericht uniek te definiëren. Dit veld is verplicht binnen de specificatie. De meeste frameworks/toolkits genereren daarom standaard een unieke messageID voor elk bericht indien deze niet is meegegeven door de applicatie.
Hoewel dit in de meeste gevallen prima werkt, is het aan te raden om gebruik van de MessageID en deze via de applicatie te laten bepalen. Met name bij asynchroon berichten verkeer, heeft het beheren van een MessageID verschillende voordelen. Bijvoorbeeld in de scenario dat een asynchroon bericht wordt verstuurd door een andere interne applicatie proces, kan door gebruik te maken van het interne proces nummer als onderdeel van de messageID, een correlatie over verschillende processen tot stand gebracht worden. Een foutbericht dat asynchroon binnenkomt kan via de relatesTo eenvoudig bepaalt worden welke interne applicatie proces een fout heeft veroorzaakt.
Daarnaast zou het formaat van een messageID ook gebruikt kunnen worden om naast een unieke waarde ook gedeeltelijk aan te vullen met een logische waarde. Indien bijvoorbeeld gewerkt wordt met een interactie waarbij meerdere berichten uitgewisseld worden voor 1 conversatie, kan het correleren versimpeld worden door een conversationID te verwerken in de messageID.
Voorkeur is om consistentie in de opbouw van de messageID aan te houden. De volgende opbouw heeft de voorkeur: "CUSTOM@UUID@URI " of "CUSTOM@GUID@URI". UUID of GUID volgens http://www.ietf.org/rfc/rfc4122.txt
URI is een anyURI volgens http://www.w3.org/2001/XMLSchema
De URI kan de domeinnaam zijn van Digikoppeling messagehandler of de web service namespace.

Routeren van berichten over meerdere intermediairs

WS-Addressing biedt de mogelijkheid om via vaste metadata informatie berichten te voorzien van routeer informatie. Hiervoor gebruikt men met name het 'TO' adres. Het is aan te raden om in het TO adres, het beoogde eindadres op te nemen. De endpoint die het bericht ontvangt kan door middel van de waarde van TO adres bepalen hoe het bericht doorgezet wordt. Intern dient de intermediair een mapping tabel bij te houden naar wie een bericht doorgestuurd moet worden afhankelijk van de TO waarde van het bericht. Dit kan dus naar de eindbestemming zijn, of weer naar een andere intermediair. De mapping inrichting dient vooraf afgesproken en ingericht te zijn.
Het is geen optie om het TO adres continu te herschrijven bij elke intermediair, waarin de TO adres de waarde krijgt waar het naar toe moet gaan. Dit is namelijk niet mogelijk, als het bericht ondertekend is want de TO waarde is onderdeel van de ondertekening en mag dus niet gewijzigd worden.

Ter verduidelijking, de volgende sequence diagram:



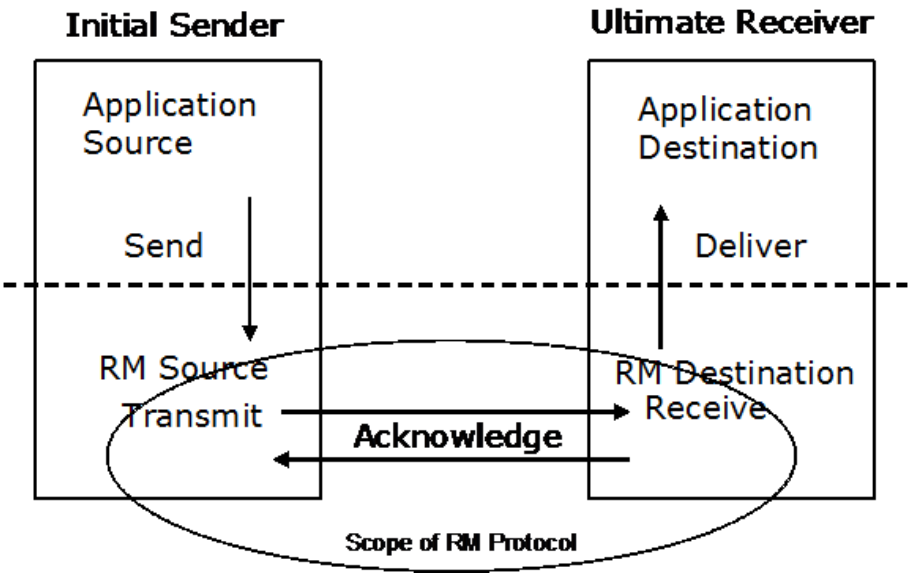
Zoals in het diagram getoond wordt, blijft de addressing informatie gelijk tijdens de hele verzending. Indien het bericht ondertekend en versleuteld is, hoeven gateway 1 en gateway 2 het bericht niet te valideren of ontcijferen. Zolang de addressing informatie niet veranderd wordt, is de meegestuurde 'signature' nog steeds valide. De gateway componenten lezen enkel de To address uit, om te bepalen waar het bericht naartoe gestuurd moet worden.
Voor een terugmelding, wordt in feite de rollen van requestor en provider omgedraaid omdat de provider een nieuwe bericht verstuurd naar de requestor. Om te bepalen waar het bericht naartoe gestuurd moet worden, kan de provider gebruik maken van het from adres dat mee is gestuurd; maar ook op basis van informatie in het bericht kan op business/applicatie-niveau bepaald worden waar een vervolg-bericht naar toegestuurd moet worden.

Afhandeling From in combinatie met een proxy/gateway

Een from adres geeft aan waar het bericht vandaan is gekomen. Dit adres wordt vervolgens gebruikt om te bepalen waar het bericht vandaan is gekomen en eventueel om terugmeldingen uit te voeren. In complexe infrastructuur oplossingen waarbij gebruik is gemaakt van een reverse proxy kan dit voor bepaalde complicaties zorgen (zie http://en.wikipedia.org/wiki/Reverse_proxy). Een server kan namelijk niet het from adres als endpoint gebruiken omdat deze moet wijzen naar de interne proxy. Extern verkeer dient vaak in dergelijke situaties altijd over

de proxy te gaan. Om dit probleem op te lossen dient ook hier gebruik gemaakt te worden van een endpoint mapping functionalitet.

2.4 WS-ReliableMessaging



Bovenstaande diagram toont aan dat 4 componenten nodig zijn voor een WS-RM communicatie. Namelijk:

1. AS: Application Source
2. RMS: Reliable Messaging Source
3. RMD: Reliable Messaging Destination
4. AD: Application Destination

Tooling en applicaties die WS-RM ondersteuning bieden, starten in runtime principe de RMS en RMD op. De AS en AD zijn de componenten met de daadwerkelijke logica implementatie. Bij het gebruik van een ESB is het retour proces ook gelijk de AD implementatie.
De bovenstaande 4 afkortingen worden in het document verder gebruikt waar nodig om te refereren naar een specifieke WS-RM component.

WS-Reliable Messaging is een standaard om een succesvolle bericht aflevering te garanderen tussen twee endpoints. Het omschrijft een protocol waar d.m.v. van een message id in combinatie met een sequentie nummer bepaald kan worden of een bericht is ontvangen. Indien een bericht niet ontvangen is, zal het automatisch opnieuw verstuurd worden naar de bestemming.
Voor het gebruik van WS-RM zijn een aantal best practices/interessante topics verzameld die implementatie partijen kunnen raadplegen. De punten zijn opgedeeld in een aantal logische categorieën, namelijk:

- Environment setup
- Service ontwikkeling
- Deployment
- Beheer

2.4.1 Environment setup

Firewall setup met betrekking tot WS-RM lifecycle berichten

Voor het ontwikkelen van een client die gebruik moet maken van een WS-RM enabled services, is het mogelijk om een publieke endpoint te gebruiken voor asynchrone WS-RM lifecycle berichten die vanuit de destination gestuurd kunnen worden. Lifecycle berichten worden transparant van de applicatie verstuurd tussen de RMS en de RMD om zaken als acknowledgements en resubmits te ondersteunen.
Bij het aanmaken van de CreateSequence bericht wordt door de RMS een AcksTo adres meegegeven. Dit adres bepaald gedeeltelijk hoe lifecycle berichten worden behandeld. Indien AcksTo element de waarde http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous bevat zullen lifecycle berichten vanuit de RMD enkel via de back channel teruggestuurd worden (via piggybacking). Piggybacking is een techniek om header informatie mee te laten liften op andere response berichten zodat er niet een aparte retour bericht nodig is. Indien een adres wordt meegegeven via de AcksTo kan een RMS aanvullend zelf lifecycle berichten sturen naar de RMD. Hier zal dus expliciet rekening meegehouden moeten worden tijdens het inrichten van de omgeving en het deployment van de endpoints. Als een RMD namelijk berichten terug kan sturen naar de RMS dan moet ook de firewall poort voor retour berichten opengezet worden. Mede door deze extra complexiteit raden wij af om een publieke acksTo endpoint te gebruiken.

Loadbalancer inzet voor geclusterde omgevingen

Indien de WS-RM service op een geclusterde omgeving draait dient er nagedacht te worden over het routeren van requests naar de betreffende service. Een round-robin algoritme in de loadbalancer (waarbij berichten automatisch verdeeld worden onder alle nodes) kan alleen gebruikt worden indien identifiers in een gedeelde storage opgeslagen zijn. Elke bericht is namelijk voorzien van een identifier en een sequence nummer om de reliability te ondersteunen. Indien de identifiers niet gedeeld zijn over de applicatie servers (via een database of gedeelde memory oplossingen als Terracotta) zal een sticky session oplossing noodzakelijk zijn. Bij een sticky session oplossing zullen alle berichten gedurende een sessie van een client naar dezelfde node gestuurd worden. Let wel op dat bij een node uitval de context kwijt is van de specifieke client.
Clustering dat niet afhankelijk is van sticky session kan de load gelijkmatig verdelen over de beschikbare nodes en levert een hogere betrouwbaarheid (namelijk automatisch failover zonder data verlies). Hoewel het vaak meer effort kost om clustering zonder sticky session werkend te krijgen, is dit een meer wenselijke opzet.

TLS setup

Voor alle Digikoppeling profielen geldt dat de communicatie over een tweezijdig TLS verbinding dient te gaan. Let op: het is belangrijk om er zeker van te zijn dat de verbinding correct opgezet is. Zonder een correcte verbinding, komen de calls niet binnen in de applicatie en zal ook geen foutafhandeling in de applicatie plaats vinden.
Alvorens een omgeving daadwerkelijk gebruikt wordt om een service produceren ofwel consumeren, is het aan te raden om via de beheerder de tweezijdig TLS verbinding te controleren.

2.4.2 Service ontwikkeling

Reliable protocol vs reliable solution

WSRM is een reliable protocol. Dit houdt in dat de focus ligt bij communicatie over de lijn. Het geeft geen antwoord op een betrouwbare verwerking binnen een applicatie. Hiervoor dienen andere maatregelen op het gebied van applicatie logica of infrastructuur inrichting meegenomen te worden. Wat een reliable protocol je biedt is dat een bericht daadwerkelijk betrouwbaar van A naar B is gestuurd en ook ontvangen is. Het zorgt voor zaken als het retransmitten van berichten van A naar B indien B niet binnen de gestelde tijd een bevestiging heeft gestuurd van ontvangst. Het kan ervoor zorgen dat berichten in volgorde en ontdubbeld worden aangeleverd aan de applicatie. Eenmaal aangekomen bij de applicatie, is het wat betreft de protocol succesvol afgehandeld. Hoe fouten binnen de applicatie vervolgens opgelost moeten worden valt buiten de scope van het protocol.
Welke problemen lost een betrouwbare protocol dan op?

1. bericht verlies over het netwerk
2. bericht verlies als systemen tijdelijk niet bereikbaar zijn

Het risico van bericht verlies wordt door WSRM opgelost door te werken met bevestigingen. Indien er geen bevestiging ontvangen is, worden berichten automatisch opnieuw verstuurd. Bij onverwachts systeem uitval van zowel de client als de server is het dus van belang dat de identifiers en sequences een systeem crash kunnen overleven. Een persistent storage voor WSRM is dus zeer aan te raden voor reliable verkeer. De meeste toolkits/frameworks onderkennen het belang van deze storage en bieden vaak ondersteuning voor verschillende databases.
Een reliable protocol is feitelijk een onderdeel van een reliable solution. Ook op het gebied van betrouwbaarheid geldt dat een ketting zo sterk/reliable is als de zwakste schakel. Een sterk Digikoppeling vereist daarom ook een sterke infrastructuur.

WS-RM acknowledgements
WS-RM ondersteunt twee technieken om acknowledgements te ontvangen, namelijk:

- een actieve terugkoppeling van acknowledgement berichten via een publieke AcksTo adres
- een passieve terugkoppeling van acknowledgement berichten door mee te liften op response berichten (piggybacking)

Het gebruik van een publieke AcksTo adres binnen Digikoppeling zorgt voor extra complexiteit en wordt daarom sterk afgeraden. Zie onder het kopje "Piggybacking voorkeur boven dynamic public endpoint" voor de achtergrond hiervan.

Zoals bekend stuurt WS-RM acknowledgement informatie terug naar de RMS zodat de RMS kan bepalen dat een bericht verwerkt is. Het is belangrijk om te weten wat het precies betekent als een bericht acknowledgement ontvangen wordt. Een bericht wordt acknowledged als het bericht door de RMD is doorgegeven aan de AD. Wat de AD in de verwerking heeft vervolgens geen invloed op de acknowledgement.

Bijvoorbeeld: een AD heeft als implementatie dat een ontvangen bericht wordt bekeken en wordt op een JMS queue geplaatst. De RMD ontvangt

bericht 1 en stuurt deze door naar de AD. De AD probeert het bericht te plaatsen op een JMS maar die blijkt niet online te zijn. De AD geeft dus een exceptie. De RMS ontvangt dan toch een acknowledgement omdat het bericht door RMD is afgeleverd aan de AD.

Een belangrijk consequentie van deze werking is bijvoorbeeld als 3 berichten worden verstuurd waarbij het inOrder delivery insurance van kracht is waarbij bericht 2 niet aan komt. Het gevolg is dat hoewel bericht 3 'over the wire' wel is aangekomen bij de RMD, toch niet acknowledged wordt omdat deze niet wordt afgeleverd aan de AD. Zowel bericht 2 als 3 worden dan opnieuw verstuurd door de RMS.

Het is zeer aan te raden om de AD implementatie zo licht mogelijk te houden en min mogelijk logica te geven. De hoofdtaak van dit component zou idealiter enkel het bericht persisteren zodat een ander asynchroon proces de daadwerkelijke verwerking kan doen.In het geval van een ESB betekent dit dat het routeer proces zich zou moeten beperken tot het opslaan van het bericht en vervolgens een ander asynchroon proces start.

WSRM met non-repudiation ondersteuning

Met de introductie van melding functionaliteit binnen WUS is de vraag van non-repudiation ontstaan. Bij bevragingen is deze vraag minder relevant gezien een client altijd wacht op een antwoord van de server en tevens worden over het algemeen geen mutaties uitgevoerd in bevragingen. Voor meldingen daarentegen is het vaak wel van belang dat zowel de versturende partij als de ontvangende partij onweerlegbaar kan bewijzen dat een bericht correct ontvangen/verstuurd is. Met correct wordt gewezen op niet alleen te weten dat een bericht ontvangen/verstuurd is maar ook dat de content onveranderd is gedurende het communicatie proces. WSRM specificatie is gericht op betrouwbare aflevering van berichten over de lijn maar heeft geen antwoord op onweerlegbaarheid. Onweerlegbaarheid is namelijk vaak een security eis waardoor het antwoord logischer wijs ook in de WS-Security specificatie is verwerkt. Als ontvangende partij kan door middel van een signature waarde bepaald worden of een bericht onveranderd is verzonden door een vertrouwd persoon. De verzendende partij verwacht dan een response terug met een signature van de ontvanger inclusief de originele signature die verstuurd was. Door de orginele signature mee te geven in een signatureConfirmation, kan de versturende partij aantonen dat het bericht onveranderd ontvangen is. Indien non-repudation een vereiste is, is het aan te raden de signatureConfirmations gecombineerd met de response signature vast te leggen ter naslag. Dit kan variëren van een simpele log bestand tot een database (afhankelijk van de omstandigheden).

Gebruik van delivery Insurance

Delivery insurance binnen WSRM wordt gebruikt om het gedrag van bericht aanlevering te bepalen. Binnen Digikoppeling passen we de mogelijkheid toe om elk bericht met zekerheid één keer te laten aankomen (èn niet vaker) en optioneel in volgorde van verzending. Opmerkelijk is dat deze configuratie feitelijk niet de communicatie over de lijn beïnvloed maar met name betrekking heeft op de communicatie tussen de RMD en de AD. Over het netwerk kunnen bijvoorbeeld best meerdere berichten uit volgorde verstuurd worden, maar de RMD hersteld dit voordat berichten aan de AD doorgegeven worden.

Het gebruik van deze configuratie heeft overigens wel grote invloed op de werking van WSRM. Met name het gebruik van 'InOrder' configuratie dient expliciet aandacht aan besteedt te worden.
InOrder garandeert namelijk dat berichten op volgorde van versturen ook op volgorde verwerkt wordt door de server. Bijvoorbeeld als bericht nummer 2 door netwerk issues niet aankomt bij de server maar bericht nummer 3 wel, zal de RMD er voor zorgen dat bericht nummer 3 niet doorgezet wordt naar de AD. In plaats daarvan zal het wachten op een retransmit van bericht nummer 2 alvorens door te gaan met de volgende berichten. Dit kan dus in potentie betekenen dat bericht nummer 2 de gehele berichtenstroom blokkeert en voor performance problemen kan zorgen. Op volgende berichten hebben namelijk geen acknowledgements ontvangen dus zullen ook opnieuw verstuurt worden door de client.
Een ander belangrijk punt is dat de InOrder betrekking heeft op de volgorde binnen dezelfde sequence. Indien er 15 berichten verstuurd worden vanuit de client waarvan vanaf bericht 11 een nieuwe sequence gestart wordt, kan het in potentie gebeuren dat bericht 11 eerder verwerkt wordt in de applicatie dan bericht 10. Bericht 11 is namelijk verstuurd met een andere sequence en heeft dus geen relatie voor WSRM tot bericht 10. Indien het van cruciaal belang is dat alle berichten in volgorde ontvangen worden, moet binnen de applicatie ervoor gezorgd worden dat er geen nieuwe sequence gestart kan worden als de oude sequence nog actief is en berichten nog niet acknowledged zijn.
InOrder moet altijd gecombineerd worden met ExactlyOnce.
Het gebruik van AtMostOnce is niet toegestaan binnen Digikoppeling omdat dit mogelijk de betrouwbaarheid van de berichtenverkeer aantast. De WSRM specificatie definieert namelijk het volgende voor AtMostOnce:

'The RM Source MAY retry transmission of unacknowledged messages, but is NOT REQUIRED to do so.'

Dit houdt in dat het implementatie afhankelijk is of retransmission berichten worden verstuurt als AtMostOnce is aangezet. In het geval van een bericht verlies door netwerk uitval, kan het dus zo zijn dat het bericht niet nogmaals verstuurd wordt.

Piggybacking voorkeur boven dynamic public endpoint

Om voor WSRM een reliable message verkeer te kunnen realiseren moet buiten het daadwerkelijke bericht nog extra lifecycle management informatie zoals CreateSequence gecommuniceerd worden tussen de RMS en de RMD. De communicatie tussen deze twee punten kan zowel direct als indirect:

- Bij direct verkeer verstuurt een RMS een eigen lifecycle bericht naar de RMD (en omgekeerd). CreateSequence is bijvoorbeeld een bericht dat de RMS (onafhankelijk van de Source Application) naar de RMD verstuurt om een sequence te starten. Om direct lifecycle berichten vanuit de RMD terug naar de RMS te kunnen sturen, moet een publieke endpoint beschikbaar gesteld worden. Let op: voor deze endpoint moet een TLS verbinding gemaakt worden conform Digikoppeling specificaties.
- Naast het directe verkeer maakt WSRM voor retour-verkeer van RMD naar RMS gebruik van indirect verkeer door extra header informatie mee te geven met andere berichten. Door 'mee te liften' op deze berichten kan het netwerk verkeer beperkt worden

om de overhead te minimaliseren. Deze vorm van communicatie wordt aangeduid als piggybacking.

Wij adviseren om enkel gebruik te maken van de piggybacking techniek om de volgende redenen:

- Het is eenvoudiger op te zetten en te configureren
- Er hoeft geen tweezijdig TLS verbinding terug gelegd te worden voor acknowledgement (performance winst)
- Piggybacking headers liften mee op een response van een client call wat effecienter is
- Niet alle frameworks/tooling ondersteunen een publieke acksTo punt waardoor interoperabiliteits issues kunnen ontstaan

2.4.3 Deployment

WSRM heeft met betrekking tot deployment geen directe afhankelijkheden. Het gebruik van WSRM wordt door toolkits/frameworks vaak onder water gerealiseerd door de RMS en RMD. Hierdoor zijn geen aparte best practices met betrekking tot het gebruik van WSRM voor deployments.
Een klein kanttekening bij het gebruik van publieke endpoints is dat de applicatie server de juiste truststore tot zijn beschikking heeft. Het kan per applicatie server verschillen welke truststore als default gebruikt wordt. Soms wordt de default JVM truststore gebruik en in sommige gevallen heeft de applicatie server zijn eigen truststore locatie waar de certificaten ingeladen dienen te worden.

2.4.4 Beheer

Performance overhead WSRM

Zoals beschreven brengt het gebruik van WSRM een zekere hoeveelheid overhead met zich mee. Namelijk:

- netwerk verkeer overhead (lifecycle berichten en retransmits)
- resource allocatie overhead (bij houden van actieve sequences)

Netwerk verkeer overhead wordt al geminimaliseerd door het gebruik van piggybacking. Met name de retransmit configuratie in combinatie met een heavy load kan van grote invloed zijn op de performance van de applicatie. Beheer dient hier rekening mee te houden tijdens het bepalen van de netwerk en resource capaciteit van de server. Het is aan te raden om load tests en capaciteits tests uit te voeren om het gedrag van de applicatie te bepalen.

Monitoring van WSRM berichten

Monitoring in productie omgevingen is altijd een belangrijk aspect om beheer mogelijk te maken. Door het instellen van alerts bij het bereiken van bepaalde thresholds is het voor beheer mogelijk om problemen voortijdig te voorkomen.
Bij het gebruik van WSRM kunnen de hoeveelheid berichten in een korte tijd snel oplopen door het gebruik van retransmits. Indien bijvoorbeeld een service tijdens een piekbelasting niet meer te bereiken is, zal gedurende de downtime van de server continu retransmits verstuurd

worden. Indien de frequentie van retransmits hoog ingesteld is, kan dit mogelijk voor problemen zorgen. Het is dus belangrijk om bij downtime van systemen zo snel mogelijk acties worden ondernomen.

2.5 WS-Policies

WS-Policies is onderdeel van de WS-* specificaties en is sinds 2007 is versie 1.5 final. De doelstelling van WS-Policies is om een framework aan te bieden waarmee domein specifieke definities in een WSDL opgenomen kunnen worden. Gebaseerd op deze framework kunnen specifieke WS-* standaarden extensies definiëren op een eenduidig manier. De extensies worden dan in de policy elementen toegevoegd zodat ze vast gelegd zijn in de WSDL. Frameworks/toolkits kunnen tijdens het inlezen van de WSDL dan al bepalen hoe met de policy elementen omgegaan moet worden en kunnen hiervoor al de voorbereidingen treffen. Bijvoorbeeld: indien op WSDL niveau al aangegeven is dat een bericht gesigned moet worden, kan een framework zoals Apache CXF al de benodigde interceptor registreren die het signing proces afhandeld. Dit gebeurd dan transparant voor de ontwikkelaar.

Hoewel de standaarden reeds geruime tijd zijn uitgewerkt en gefinaliseerd, is in de praktijk de implementatie ondersteuning nog wisselvallig. Hierdoor is gekozen om het gebruik van policies niet binnen de koppelvlakstandaard op te nemen maar optioneel te houden. De meerwaarde van het gebruik van policies is daarentegen dusdanig evident dat het binnen de best practices is opgenomen.

Het gebruik van policies kunnen verschillende zaken zowel versimpelen als verduidelijken in de ontwikkelfase maar zijn niet noodzakelijk om ingezet te worden. In de praktijk blijkt wel dat zaken als signing en encryption zonder policies tijdens het ontwikkelen voor de nodige complexiteit zorgt. Het is dus te overwegen om bij het publiceren van een service in het serviceregister zowel een WSDL versie zonder policies (deze is verplicht) als met policies (deze is optioneel) mee te nemen. Implementatie partijen kunnen dan afhankelijk van het gebruikte framework/toolkits bepalen of zij de policy variant kunnen inzetten of niet. Uiteraard is het onderhouden van meerdere WSDL varianten voor dezelfde service niet wenselijk. Daarentegen kan de impact daarvan beperkt worden door te werken met een externe policy file dat in de WSDL wordt geimporteerd en vervolgens met policy references aan specifieke methodes te koppelen. Door de policies extern te houden, is het zelfs mogelijk om verschillende policy varianten te maken indien een veel gebruikte framework een policy niet interoperabel ondersteund. De kans op implementatie fouten zou hiermee verlaagd kunnen worden wat zal resulteren in een sneller adoptieproces. De baten die hiermee behaald kunnen worden zijn hoogstwaarschijnlijk hoger dan de kosten die ermee gemoeid zijn.

Voorbeeld: Partij A besluit om een melding dienst aan te bieden waar andere partijen zich kunnen abonneren en meldingen kunnen ontvangen. Dit houdt in dat alle partijen de service moeten implementeren om de meldingen te kunnen ontvangen. Indien er 1000 verschillende partijen zijn waarbij het bekend is dat 50% gebruik maakt van Oracle Webcenter technologie en 30% gebruik maakt van Microsoft technologie. Dan zou de case om een policy attachment aan te bieden voor beide technologieën dat getest is op een correcte werking zeer goed te maken zijn. Door het

inlezen van de WSDL met de bijbehorende policy attachments staan alle webservice instellingen zoals signing onderdelen gelijk goed.

De handleiding voor het Digikoppeling Service Register beschrijft hoe publicatie van WSDL-varianten kan plaatsvinden.