Discussiedocument RESTful APIs

Versie 1.1 – juni 2016

Contents

1. Inleiding

Aanleiding onderzoek

Voor allerhande problemen en toepassingen zijn er steeds meer programma's en app's die je helpen. Je huis zorgt met 'slimme' apparaten dat de temperatuur aangenaam is. Je navigatiesysteem past routes aan op basis van actuele verkeersinformatie. En via Instagram kan je foto's direct delen via andere platforms. Hiervoor is het nodig dat informatie snel en eenvoudig wordt uitgewisseld. Dit vraagt om efficiënte koppelingen tussen systemen. Deze koppelingen tussen systemen worden vaak via APIs gerealiseerd gebaseerd op het REST architectuurprincipe.

Een Application Programming Interface (API) is een combinatie van technische bestanden, documentatie en andere ondersteuning die helpen bij het aanroepen van externe applicaties. Een API wordt gepubliceerd door de softwareontwikkelaar zodat andere ontwikkelaars weten hoe de software te koppelen aan de eigen software.[1] Het is daarmee geen standaard, maar meer een handleiding die kan worden gebruikt voor een machine tot machine koppeling. REST is geen standaard maar een architectuurconcept voor het aanspreken van webservices. In de praktijk meestal toegepast met standaarden als HTTP voor de transport en XML of JSON voor de berichtinhoud.

Hoe om te gaan met APIs en welke standaarden of principe te gebruiken is een discussie die regelmatig terugkomt bij het Forum. Opmerkingen die dan langskomen zijn dat RESTful APIs de rol van standaarden vervangen of dat de standaarden op de lijst 'ouderwets' zijn en met name gericht op de standaarden SOAP / WSDL. Het REST principe voor informatie uitwisseling wordt ook vaak tegenover de SOAP / WSDL standaarden geplaatst. Waarbij REST staat voor snelheid, gebruiksvriendelijkheid en innovatie en SOAP / WSDL voor formaliteit, betrouwbaarheid en complexiteit. Een API kan overigens beide ondersteunen.

Dat deze discussie ook het Forum raakt is overigens niet zo gek omdat het direct aansluit op vraagstukken over interoperabiliteit en gegevensuitwisseling. Een API zorgt immers voor interoperabiliteit en reguleert gegevensuitwisseling. Daarnaast zijn in de praktijk ook niet alle standaarden van onze lijst goed toegerust op het REST principe. Zie daarvoor bijvoorbeeld het recente onderzoek van de Software Improvement Group naar de STUF-standaard.[2] Om dus meer inzicht te krijgen op de ontwikkeling rondom RESTful APIs en de impact hiervan op de standaarden die op lijst staan is deze notitie opgesteld.

Vraagstelling

Dit document is geschreven om meer inzicht te krijgen op wat RESTful APIs zijn, wat de impact is op gegevensuitwisseling en hoe deze ontwikkeling zich verhoudt tot de standaarden van de 'pas toe of leg uit'-lijst. Het is daarmee meer een introductie in de materie waarbij wordt ingegaan op de volgende vragen:

[1] Een goed voorbeeld hiervan is bol.com (https://developers.bol.com/). Ook worden veel APIs aangeboden op platforms als Github of programmableweb.
[2] https://www.sig.eu/nl/over-sig/publicaties/analyse-van-de-stuf-bg-standaard/

- Wat is REST en wat zijn APIs waarbij REST vaak wordt toegepast?
- Is er sprake van een trend op het gebied van REST en het gebruik van APIs?
- Wat zijn de verschillen tussen REST en SOAP/WSDL?
- Wat is de invloed van REST en APIs op gegevensuitwisseling tussen overheidsorganisaties en bedrijven, tussen overheidsorganisaties en burgers en tussen overheidsorganisaties onderling?
- Wat is de invloed van REST en APIs op standaarden voor gegevensuitwisseling in het algemeen?
- Wat is de invloed van REST en APIs op de lijst met open standaarden en de standaarden die daar op staan, zijn er bijvoorbeeld conflicten?
- Welke standaarden (die nog niet op de PTOLU lijst staan) die veel voorkomen bij REST en APIs zijn interessant voor een mogelijke aanvulling op de lijst?

Aanbevelingen uit het onderzoek

Met name daar waar veel digitale diensten met elkaar samenwerken en informatie op een makkelijke en toegankelijke manier willen delen zijn RESTful APIs zeer geschikt. Van deze techniek is veel kennis in de markt, het is snel en laagdrempelig te implementeren. Dit betekent niet dat de overheid volledig over moet naar REST. SOAP/WSDL kent ook voordelen en zeer legitieme toepassingen. Daar waar betrouwbaarheid en formaliteit het belangrijkst is, is berichtuitwisseling via SOAP en WSDL te prefereren. De standaarden op de lijst zijn over het algemeen te combineren met de REST principes maar zijn wel veelal XML standaarden die vaak gebruikt worden in combinatie met SOAP, dat kan betekenen dat er complexe structuren worden voorgeschreven die niet flexibel zijn voor REST. Het verdient dan ook de aanbevelingen dat het Forum aandacht heeft voor deze vernieuwende ontwikkeling en bij het verplichten van standaarden meer oog heeft voor de praktisch implementatie van de standaarden in de praktijk. Daarom zijn onderstaande aanbevelingen geformuleerd die het Forum kan oppakken.

1. Toetsen Standaarden: Het Forum kan op eigen initiatief de standaarden Oauth en Odata toetsen voor opname op de lijst met standaarden. (*Deze aanbeveling is overgenomen*)
2. Kwaliteitstoets: enkele standaarden van de lijst zouden getoetst kunnen worden in hoeverre ze aansluiten en geschikt zijn voor REST. Dit zou samen met de beheerorganisaties moeten worden opgepakt en worden geanalyseerd. (*Deze aanbeveling is niet overgenomen, dit is meer een taak van de beheerorganisatie*)
3. Handreiking RESTful APIs: Het Forum heeft eerder handreikingen gepubliceerd over WEB of APP en Authenticatieniveaus. APIs sluiten nauw aan op het uitwisselen van gegevens en interoperabiliteit. Het ligt dan ook voor de hand om ook over dit thema een handreiking te publiceren en in te gaan op hoe overheden RESTful API moeten publiceren. Wel zal eerst onderzocht moeten of hier behoefte aan is. (*Deze aanbeveling is overgenomen*)

Over het onderzoek

Het onderzoek is geschreven door Frank Terpstra van Enable-U in samenwerking met Lancelot Schellevis van Bureau Forum Standaardisatie. Frank is integratiespecialist en houdt zich veel bezig met het koppelen van ICT systemen met name binnen de overheid, In zijn dagelijkse werk is hij veel betrokken bij standaarden als Digikoppeling, de Geo-standaarden

en StUF. Lancelot werkt als adviseur standaardisatie bij het Bureau Forum Standaardisatie en is betrokken bij het vaststellen van standaarden voor de overheid en signalering van interoperabiliteitsproblemen, hij heeft het onderzoek gereviewed en begeleid.

De notitie kan gelezen worden als een toelichting op termen en ontwikkelingen naar APIs en REST, het is daarmee een achtergronddocument. Het document is opgesteld vanuit de expertopinie en niet op basis van interviews. Daarnaast is in het onderzoek ook de ruimte genomen om vraagstukken te adresseren die het Forum zou kunnen oppakken of onderzoeken.

Het onderzoek is door Lancelot Schellevis gepresenteerd in de Forumvergadering van 15 maart 2016 en door Frank Terpstra in het GAB-overleg (Gemeenschappelijk Afspraken Berichten) van 13 april 2016.

Leeswijzer

In bovenstaande inleiding is de basis van het onderzoek uiteengezet. In de volgende hoofdstukken zal, na de begrippenlijst, dieper worden ingegaan op de achtergrond van de ontwikkeling naar Restful APIs. Ook wordt de relatie tussen REST en SOAP toegelicht en kijken we naar de invloed van REST op gegevensuitwisseling met name die van de overheid. Tot slot komt de impact van RESTful APIs op de lijst met standaarden van het Forum Standaardisatie aan bod en de aanbevelingen die het Forum zou kunnen oppakken.

2. Begrippenlijst

Er zijn een aantal begrippen(vaak afkortingen) die in dit document veel voorkomen:

Begrip	Definitie
API	Application Programming Interface: De combinatie van technische bestanden, documentatie en andere ondersteuning. APIs helpen programmeurs bij het aanroepen van externe applicaties/functies die zodoende in een website of platform geïntegreerd worden. APIs zijn meestal gebaseerd op één of meerdere webservice(s).[3]
Webservice	Een Webservice is de implementatie op een webserver voor een machine to machine koppeling gecombineerd met een applicatie/database die de inhoud van de webservice levert. Daarmee zijn ze specifieker dan een API, een webservice is de technische implementatie van (een deel van) een API. Webservices zijn bijna altijd op REST of op SOAP/WSDL standaarden gebaseerd. Door webservices is het bijvoorbeeld mogelijk om zonder menselijke tussenkomst een kamer in een hotel te boeken of een vlucht bij een luchtvaartmaatschappij terwijl de webservice in een andere applicatie/website is geïntegreerd.
SOAP	Simple Object Access Protocol (SOAP) is een standaard (een “envelop”) voor elektronische berichten. De standaard voor de envelop is op XML gebaseerd maar stelt geen eisen aan de inhoud van de envelop. Je kan er dus van alles mee versturen, meestal is de inhoud XML maar het kan ook JSON een JPG plaatje of nog iets anders zijn. Het stelt ook geen eisen aan het transport protocol, meestal is dit HTTP maar het kan ook iets anders zijn.
WSDL	WebService Description Language (WSDL) is een formaat waarmee gedefinieerd wordt wat voor soort berichten een webservice kan ontvangen en versturen. Het wordt gebruikt om systemen al dan niet automatisch te configureren voor en aanroepen van een webservice. Het is een op XML gebaseerd formaat, en beschrijft bijna altijd SOAP over HTTP uitwisselingen. Het kan ook gebruikt worden voor het beschrijven van andere uitwisselingen.
SOAP/WSDL	In dit document duiden we met SOAP/WSDL twee sets standaarden aan voor het communiceren met SOAP over HTTP. De standaarden van W3C[4] voor het uitwisselen van SOAP berichten over HTTP gedefinieerd in een WSDL ook bekend als WS-* en de OASIS standaard voor SOAP berichten over HTTP volgens de ebMS standaard[5].
REST	REST staat voor REpresentational State Transfer, het is geen standaard of een protocol maar een architectuur concept voor het aanspreken van webservices waarbij met name ‘bruikbaarheid’ centraal staat. In de praktijk meestal

[3] Voorbeelden van APIs zijn er in overvloed te vinden bij grote platforms. Google: https://developers.google.com/products/ facebook: https://developers.facebook.com/docs/apis-and-sdks linkedin: https://developer.linkedin.com/ twitter: https://dev.twitter.com/rest/public

[4] https://www.w3.org/2002/ws/

[5] ebMS is een op SOAP gebaseerde standaard voor webservices van OASIS en gebruikt geen WSDL. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-msg

	toegepast met de http 1.1 standaard voor transport en XML of JSON voor berichtinhoud. REST en RESTful is overigens hetzelfde en worden door elkaar gebruikt.
XML	eXtensible Markup Language. XML is een formaat om gestructureerde gegevens in op te slaan en gegevens te versturen over het internet. Het wordt ook gebruikt voor de inhoud van webservice berichten. XML is gericht op de mogelijkheid tot validatie en leesbaarheid door mensen. XML zelf is een open formaat met een heel simpele syntax, dat geen specifieke elementen voorschrijft. Specifieke structuren van XML documenten kan je met XML Schema(XSD) beschrijven. Met XML Schema kan je bijvoorbeeld vastleggen dat een <persoon> precies één geboortejaar heeft. XML kent een zelfbeschrijvende niet compacte notitie bijvoorbeeld: <persoon> <naam>Jan</naam> <geboren>1983</geboren> </persoon>

JSON	JavaScript Object Notation, een formaat om net zoals XML gegevens op te slaan en te versturen. JSON kan ook gebruikt worden voor de inhoud van webservice berichten en is met name gericht op efficiënt programmeren. Kent een compacte notatie bijvoorbeeld:
	{
	“naam”: Jan,
	“geboren”: 1983
	}

Wat goed om te onthouden is dat:

- Een API gebaseerd kan zijn op ofwel webservice volgens een REST architectuur danwel op webservice gebaseerd op SOAP/WSDL standaarden.
- REST en SOAP/WSDL beide gebruikt kunnen worden voor de implementatie van een webservice.
- REST en SOAP kunnen als uitwisselformaat allebei zowel JSON als XML gebruiken (of iets geheel anders).
- REST architectuur zegt niets over de logistiek, iets wat juist de kern van SOAP WSDL is. REST implementaties maken hier geen gebruik van, of passen één of meerdere standaarden uit de logistieke laag toe, of lossen logistieke vraagstukken op een ander niveau op.

3. Achtergrond RESTful APIs

REST is niet nieuw, eind jaren 90 is REST bedacht en in 2000 gepubliceerd. Ook de SOAP en WSDL standaarden zijn halverwege 2000 als standaard vastgesteld. Het zijn technieken die parallel zijn ontwikkeld en al jaren naast elkaar bestaan. Toch wordt vaak gezegd dat SOAP ouderwets is. Dit komt met name omdat in de jaren 2000-2008 SOAP/WSDL het meest werd gebruikt voor APIs en de laatste jaren (2008-heden) REST populair is geworden.[6] Dit is voor een groot deel te verklaren door de opkomst van social media en de toenemende integratie van webservices in allerhande online diensten. Eenvoud, bruikbaarheid en schaalbaarheid is daarbij belangrijk omdat veel programmeurs 'elkaars' producten gebruiken en RESTful APIs zijn daarvoor goed geschikt. Zodoende is er gemakkelijker interoperabiliteit te realiseren tussen verschillende databases en online diensten.

Voorbeeld 1
Via de RESTful Open API van BOL.com kunnen andere winkels die via het web producten verkopen aansluiten op het platform van BOL.com. Hiermee is de eigen productendatabase onderdeel geworden van de catalogus van BOL en kunnen ze producten aanbied van het BOL platform. Een soort van Shop in Shop.

Onderzoeksbureau Gartner[7] ziet een duidelijke trend naar RESTful APIs en verklaart de populariteit als volgt. In het begin van de populariteit van internet (tot ongeveer 2008) was er maar één kanaal belangrijk. De menselijke gebruiker met een webbrowser. Sindsdien zijn er veel meer kanalen ontstaan waarnaar je als organisatie je diensten wil ontsluiten:

- Mobiele devices(telefoons tablets);
- the internet of things(slimme themostaten, tvs autos);
- social media platforms als twitter, facebook;
- wearables (iwatch, fitbit, google glass).

Gebruik van RESTful APIs zie je vooral terugkomen bij platforms (denk aan google, facebook, linkedin, uber). Zij hebben zowel contact met hun gebruikers, maar ontsluiten hun platform ook naar anderen. Ook de beweging naar cloudcomputing draagt bij aan het gebruik van API. Diensten worden steeds meer in de cloud aangeboden (en afgenomen) waarbij het nodig kan zijn om deze te integreren in andere producten. Voor toepasbaarheid en schaalbaarheid past een RESTful API heel goed.

Voorbeeld 2
Het ministerie van Binnenlandse Zaken heeft een Vacature-API gebaseerd op JSON voor het ophalen en filteren van het vacatures van de websites WerkenvoorNederland.nl, WerkenbijdeOverheid.nl en Mobiliteitsbank.nl. Deze vacatures zijn zo ook te verwerken in andere vacature platforms en realiseren daarmee interoperabiliteit tussen vacaturesites.

Het is namelijk ondoenlijk geworden om je te bemoeien met hoe de userinterface op al die kanalen eruit ziet. Terwijl het voor een organisatie steeds belangrijker is om zijn unieke resources simpel te ontsluiten en

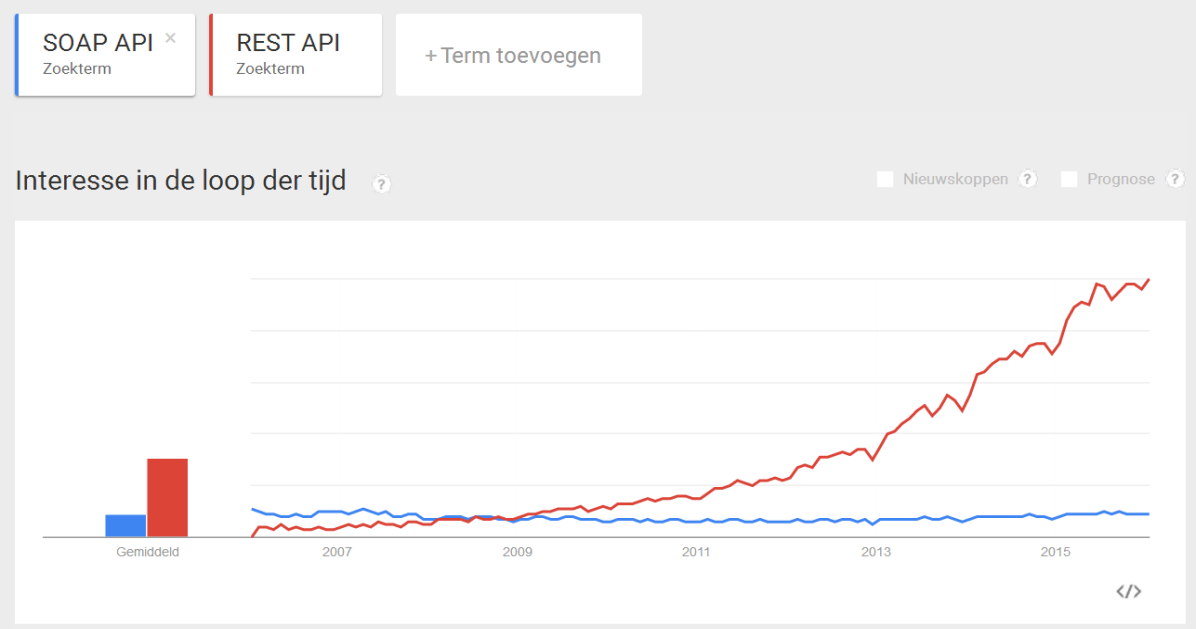
[6] http://royal.pingdom.com/2010/10/15/rest-in-peace-soap/
[7] Zie de stukken en presentaties over API management van Poalo Malinverno op www.gartner.com helaas achter een betaalmuur.

ontwikkelaars van andere platforms hier (laagdrempelig) op te laten aansluiten. Andere ontwikkelaars ontwikkelen daarmee de userinterface/userexperience gebruik makend van verschillende diensten. Als organisatie wil je er natuurlijk wel voor zorgen dat zoveel mogelijk gebruik gemaakt wordt van jouw API. Als anderen deze API hergebruiken krijg je namelijk ook hun gebruikers of klanten erbij en heb je een breder bereik.

Hergebruik kan je realiseren door het programmeurs van andere partijen zo makkelijk mogelijk te maken, hiervoor is REST goed geschikt. Overigens vallen niet alle usecases voor communicatie tussen systemen hieronder. Daar waar nauwkeurigheid(validatie van berichten) het belangrijkst is of waarbij je weet wie de gebruiker is en er een formele relatie mee hebt, kan SOAP/WSDL beter geschikt zijn.

Trends

Er is ook een trend dat naar verhouding steeds meer REST APIs worden gebruikt, kijkend naar de populariteit van zoekopdrachten via GOOGLE Trends.[8]



De interesse in SOAP APIs is redelijk stabiel, maar hieruit valt ook af te lezen dat de API ontwikkeling de laatste jaren met name gekoppeld is aan REST. Als we kijken naar zoekopdrachten m.b.t. REST en SOAP alleen is het verschil minder groot. Maar ook daar is te zien dat er meer interesse is voor REST dan voor SOAP.

Voorbeeld 3
Het Rijksmuseum publiceert zijn eigen API op de rijksmuseum collectie te ontsluiten voor bijvoorbeeld in webtoepassingen of apps. Deze API is voor iedereen te gebruiken en ondersteunt JSON en XML formaten. Daarnaast hebben ze ook een API gebaseerd op het OAI protocol een formaat die met name in de culturele sector wordt gebruikt.

[8] https://www.google.nl/trends/. Test uitgevoerd op 9 februari 2016. Selectie loopt van januari 2006 tot januari 2016. De interesse worden tegenover elkaar weggezet. Het hoeft dus niet zo te zijn dat het absoluut aantal zoekopdrachten naar SOAP is gedaald.

Nu geeft google trends alleen de interesse van mensen weer en is het geen garantie voor daadwerkelijk gebruik. De interesse is echter onmiskenbaar.

Er wordt steeds meer gebruik gemaakt van REST en dit is ook terug te zien in de ondersteuning voor REST in standaard software producten. Maar ook programmeurs hebben steeds meer kennis van REST ten opzichte van SOAP/WSDL. En bestaande inhoudelijke standaarden maken zichzelf geschikt om via een REST interactiepatroon uitgewisseld te worden. Een standaard als CMIS kent bijvoorbeeld een uitwisselingsprofiel via SOAP/WSDL en één via REST. Door deze marktontwikkeling is het aantrekkelijk voor bestaande op SOAP/WSDL gebaseerde uitwisselingsstandaarden om dezelfde inhoud via een RESTful API uit te wisselen en voor XML schema's om zich onafhankelijk van de techniek te ontwikkelen. Voor deze bestaande standaarden blijft het semantiek/applicatie gedeelte, het formaat van de berichtinhoud hetzelfde. Echter ze kunnen overgaan op REST als transportmethode. Dit is met name aantrekkelijk als geen gebruik wordt gemaakt van de extra mogelijkheden die SOAP/WSDL biedt ten opzichte van REST of als hiervoor alternatieven gevonden kunnen worden.

Alvorens te kijken naar de standaarden van de 'pas toe of leg uit'-lijst gaan we in het volgende hoofdstuk dieper in op de techniek en kijken we naar de verschillen tussen REST en SOAP/WSDL. Eerst zullen de technieken afzonderlijk worden toegelicht. Vervolgens worden de voor- en nadelen met elkaar vergeleken. In Bijlage A en B zijn de technieken in meer detail uitgelegd.

4. Gegevensuitwisseling: REST & SOAP/WSDL

REST

REST architectuur

REST is een architectuurconcept dat eind jaren 90 parallel met HTTP 1.1 is ontwikkeld door Roy Thomas Fielding en gepubliceerd in 2000[9]. De basis van REST is het interactie patroon van het web, geen toeval gezien de nauwe band met HTTP. Het beschrijft hoe interactie tussen componenten plaatsvindt in een gedistribueerd systeem. In de taal van het www: je hebt een webbrowser en een webserver, hoe praten die met elkaar? REST volgt dit zelfde interactie patroon van browser en webserver maar je client is geen persoon met een webbrowser maar een machine. In de praktijk wordt REST bijna altijd geïmplementeerd op basis van HTTP, maar dat is niet noodzakelijk.

Voorbeeld 4: De client en server

Ter illustratie van de begrippen client en server, nemen we het NHR (handelsregister). Een applicatie van een gemeente heeft informatie van een bedrijf nodig. Het KvK kan deze leveren. De gemeente stuurt een KvK nummer naar de NHR webservice van de KvK en krijgt van de webservice informatie over het betreffende bedrijf terug. De gemeente is in deze uitwisseling de client, zij stellen namelijk de vraag. De KvK is de server, zij worden namelijk aangesproken en leveren de informatie.

De belangrijke eigenschappen van de REST architectuur zijn:

1. Prestaties (zorg ervoor dat machines snel met elkaar praten)
2. Schaalbaarheid (zorg ervoor dat het blijft werken bij veel gebruik(ers))
3. Simpele interfaces (de communicatie tussen componenten is eenvoudig en overzichtelijk)

In bijlage A is nader uitgelegd wat de werking is van REST.

REST in combinatie met het meestal gebruikte uitwisselingsformaat JSON is uitermate efficiënt. Het kent kleine headers[10] en ook JSON zelf gebruikt relatief weinig ruimte. Dit zorgt samen met een aantal andere architectuur kenmerken voor lage benodigde bandbreedte. Op basis van JSON kan snel code gegenereerd[11] worden voor het opstellen van berichten. Voor het aanroepen van een webservice is meer nodig, een standaard voor het adverteren van een webservice (meta data over hoe een webservice aangeroepen moet

[9] *Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine. This chapter introduced the Representational State Transfer (REST) architectural style for distributed hypermedia systems. REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems.*

[10] Een header beschrijft de eigenschappen van een bestand dat wordt uitgewisseld, je kan het zien als de envelop van een bericht.

[11] Op basis van technische documenten is het mogelijk om automatisch programmeer code te genereren zodat deze code bijvoorbeeld gebruikt kan worden om een webservice correct aan te spreken vanuit een applicatie.

worden) kan dan gebruikt worden. Vaak is dat bij REST niet nodig omdat de aanroep van een webservice zo simpel is dat codegeneratie overbodig is.[12]

Wereldwijd zie je dat het REST principe het meest wordt toegepast daar waar men zo snel mogelijk zoveel mogelijk gebruikers wil benaderen, voorbeelden zijn:

- Bij het aanbieden van open data
- Voor het aansluiten op sociale media platforms
- In toepassingen gerelateerd aan Internet of things (smart cities)
- In Apps voor mobiele devices (telefoons, tablets)

RESTful APIs

RESTful APIs zijn APIs gemaakt op basis van een REST architectuur. Het zijn dus eigenlijk implementaties van de REST architectuur. Een implementatie van een API kan gezien worden als het op een server aanbieden van één of meerdere webservices op basis van een gespecificeerd formaat (bijv. XML of JSON). Bij REST wordt er dan gebruik gemaakt van een combinatie van HTTP voor transport en JSON of XML voor de inhoud van berichten en URIs voor het refereren naar resources. Hoewel het doel van REST is dat APIs zoveel mogelijk zelfdocumenterend zijn wordt over het algemeen ook documentatie meegeleverd met een API. Platforms(denk google, twitter, facebook, linkedin)[13] die RESTful APIs aanbieden leveren daarnaast extra voorzieningen aan programmeurs die met hun API aan de slag willen. Deze platforms hebben vaak ook API portals waarin informatie over de API verzameld is. Ook zijn er API platforms waarin iedereen zijn API kan publiceren. Partijen kunnen zichzelf aanmelden op de portal en middels zelfbediening aansluiten. Verder zijn op deze portals discussie fora waar gebruikers elkaar kunnen helpen en FAQs om veelgestelde vragen te beantwoorden.

Wat is het effect van REST op een webservice implementatie?

Door gebruik te maken van REST architectuur principes wordt een implementatie van een webservice in de eerste plaats ‘snel’. In een REST implementatie zit namelijk weinig overhead en door het gebruik van ‘caching’[14] en het onzichtbaar kunnen toepassen van loadbalancers[15] e.d. kunnen webservices enorm snel antwoorden teruggeven. Een REST interface is er op gericht om clients met zo min mogelijk moeite aan te sluiten. Doordat de interface en berichten zoveel mogelijk zichzelf beschrijven kunnen programmeurs er snel mee aan de slag, bovendien is het voor een server relatief eenvoudig om nieuwe formaten aan te bieden als daar vanuit de clients vraag naar is. Zo kan snel ingespeeld worden op nieuwe trends in uitwisselformaten. Denk aan het parallel ondersteunen van JSON, XML en RDF(linked data) formaten.

---

[12] Overigens is voor het genereren van code voor databasestructuren JSON minder geschikt, het kent (nog) geen vastgestelde standaard voor schema’s.

[13] Google: https://developers.google.com/products/ facebook: https://developers.facebook.com/docs/apis-and-sdks linkedin: https://developer.linkedin.com/ twitter: https://dev.twitter.com/rest/public

[14] Een server kan aangeven of de antwoorden die hij geeft cacheable zijn. Wat betekent dat veelvoorkomende vragen worden ‘onthouden’ en vanuit de cache worden beantwoord zonder ze opnieuw op te halen. Dit verhoogd de prestaties en schaalbaarheid en de server meer capaciteit heeft voor moeilijkere vragen.

[15] Loadbalancing wordt bij servers gebruikt om een groter aanbod aan te kunnen. Een load balancer is dan een virtuele server die verbonden is met de fysieke servers en verdeeld verzoeken gelijkmatig over alle beschikbare servers. Op deze manier wordt de verwerkingstijd van een verzoek verminderd.

SOAP/WSDL

In de praktijk wordt REST vaak geplaatst tegenover SOAP/WSDL. SOAP/WSDL zijn standaarden voor gegevensuitwisseling die vaak binnen meer formele omgevingen worden toegepast zo ook binnen de overheid. Er zijn dan ook verschillende XML standaarden geschreven met oog op SOAP en WSDL. Aangezien REST een architectuur patroon is en SOAP/WSDL een set van standaarden, is de vergelijking niet triviaal. Daarom kijken we naar wat de typische interactiepatronen zijn die met SOAP/WSDL worden geïmplementeerd.

Interactie patronen SOAP/WSDL

De kern van SOAP/WSDL zit in de logistiek van berichtenuitwisseling. Het formaliseert de logistiek expliciet terwijl REST juist niets over deze logistiek zegt. Door te focussen op de interactiepatronen (kenmerken) voor logistiek die SOAP/WSDL middels standaarden oplost is het mogelijk om goed aan te geven waar de verschillen liggen tussen REST en SOAP. De interactiepatronen binnen SOAP/WSDL zijn:

- Bericht validatie: klopt de berichtinhoud met wat is afgesproken;
- Reliable messaging: garantie dat een bericht aankomt;
- Beveiligd transport: zorgen dat het verkeer niet wordt afgeluisterd;
- Onweerlegbaarheid: is een bericht intact en afkomstig van de bron;
- Adressering/routering: een bericht handig versturen via tussenstations;
- Berichtversleuteling: zorgen dat alleen de ontvanger het bericht kan lezen;
- Adverteren webservice implementatie: maakt technische documenten beschikbaar voor ontwikkelaars die aansluitingen realiseren;
- Codegeneratie: programmeercode genereren op basis van technische documenten.

In bijlage B worden deze interactiepatronen in relatie tot SOAP/WSDL in detail toegelicht.

Toepassing in de praktijk SOAP/WSDL

De nadruk ligt bij SOAP/WSDL op het heel precies definiëren van de inhoud van berichten en het (in een aantal gevallen) verder formaliseren van interactiepatronen met een nadruk op betrouwbaar transport. Met het toepassen van SOAP/WSDL kan je heel precies uitdrukken hoe de inhoud van een bericht er wel of niet uit mag zien en tijdens het gebruik controleren of beide partijen (zender en ontvanger) zich hieraan houden. Het voordeel hiervan is dat het voor ontwikkelaars relatief eenvoudig wordt om complexe data structuren in hun (Client) software te implementeren. Daarnaast hoeft de ontwikkelaar van de Client en server software niet na te denken over de betrouwbaarheid van het transport. Dit wordt op protocol niveau voor ze opgelost (zie ook bijlage B).

Hierdoor is SOAP/WSDL wel een standaard met meer overhead en de interfaces zijn complexer wat leid tot grotere headers van berichten. Ook het binnen SOAP meest toegepaste uitwisselformaat XML kent meer overhead en een relatief grote berichtinhoud omdat het erop isgericht niet alleen door machines maar ook door mensen leesbaar te zijn. Het is daardoor minder efficiënt. De implementatie van SOAP en WSDL vereisen dan ook een

relatief grote en complexe software stack[16] en is dus minder lichtgewicht als op REST gebaseerde applicaties.

Bij SOAP/WSDL is het de WSDL die goede ondersteuning biedt om code te genereren voor het aanroepen van een webservice. Op basis van het meestal gebruikte XML is het daarnaast ook goed mogelijk code te genereren voor het aanmaken van berichten maar ook databasestructuren. De programmeur hoeft maar op een knop te drukken en een groot deel van zijn werk voor het aanleveren van berichten staat in het juiste formaat net als het opslaan daarvan ook automatisch wordt gedaan. Dit is snel en minder foutgevoelig dan "ambachtelijk" programmeren. Kanttekening is dat SOAP/WSDL vaker bij complexe informatiestructuren wordt gebruikt. Code generatie is dan theoretisch mogelijk maar praktisch vaak niet goed bruikbaar.

Voorbeeld 5: Digipoort
Op basis van technische documenten is het mogelijk om programmeercode te genereren die in een applicatie gebouwd worden. Zo kan je code genereren om een webservice correct aan te spreken vanuit een willekeurige applicatie of op basis van een schema databasestructuren genereren. Digipoort is hier een voorbeeld van. Deze laat bedrijven gegevens aanleveren aan de overheid. Digipoort publiceert hiervoor WSDLs (webservice description language) en XSDs(schemas). Op basis van de WSDL kan een ontwikkelaar het aanroepen van digipoort in een applicatie inbouwen. Op basis van de XSDs kan hij een database structuur genereren waarin hij gegevens van zijn applicatie opslaat voordat hij ze verzendt en kan hij makkelijk correcte berichten samenstellen om te verzenden.

Internationaal gezien zie je dat SOAP/WSDL vooral wordt toegepast daar waar beide communicatie partners een (zeer) formele relatie met elkaar hebben. Ook bovenstaand voorbeeld over Digipoort illustreert dat. Voorbeelden waar veel gebruik wordt gemaakt van SOAP/WSDL zijn.

- Auto fabrikanten en toeleveranciers
- Financiële instellingen onderling
- Telecom providers onderling
- Gezondheidszorg tussen instellingen
- Tussen verschillende overheid organisaties nationaal en internationaal

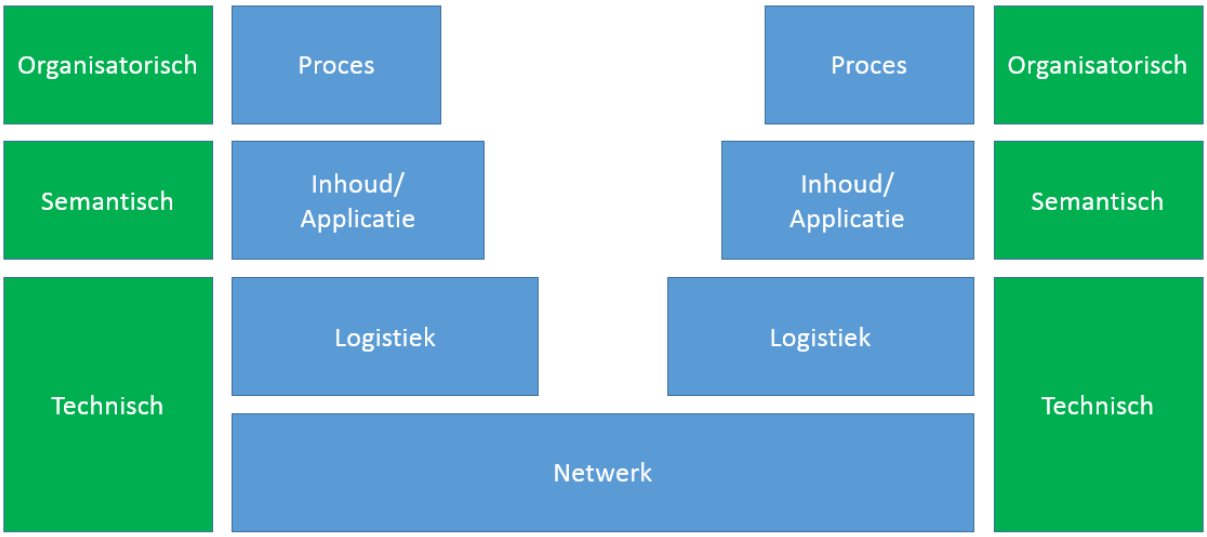
Dat de standaard daar veel gebruikt wordt heeft met name te maken met: de 'formaliteit' van de keten en dat het relatief eenvoudiger is om eenzijdig afspraken op te leggen, de betrouwbaarheid van de gegevens die worden uitgewisseld en legacy van systemen (eerder is al voorgesorteerd op SOAP/WSDL).

[16] Een software stack is een verzameling programma's of applicaties die met elkaar samenwerken om een gecombineerde opdracht te realiseren. In de regel is hoe complexer de opdracht hoe meer applicaties bij een opdracht betrokken zijn,

Context van berichtuitwisseling

Voor het opzetten van een berichtuitwisseling tussen twee partijen moeten op vele niveaus afspraken worden gemaakt. In de overheid wordt vaak de “half pipe” gebruikt om deze schematisch weer te geven. Hierin wordt onderscheid gemaakt tussen afspraken op:

- Procesniveau: welke informatie wordt tussen entiteiten en personen uitgewisseld, wordt ook wel aangeduid als organisatorisch;
- Inhoud/applicatieniveau: welke informatie wordt door applicaties verwerkt en in welk formaat staat dit, wordt ook wel aangeduid als semantisch;
- Logistiekniveau: het adresseren naar en afleveren van berichten op de juiste server, ook bekend als technisch;
- Netwerkniveau: de verbinding tussen uitwisselende machines in een netwerk, ook bekend als technisch.



Figuur 1 Geeft de afspraakniveaus voor berichtuitwisseling weer

In de onderstaande tabel hebben we een aantal van de belangrijkste begrippen uit dit document geplaatst in de context van de “half pipe” om een idee te geven op welk niveau ze spelen.

Begrip	**Niveau(s)**
API	Proces, Inhoud/Applicatie, Logistiek, Netwerk
SOAP/WSDL	Logistiek, Netwerk
REST	Inhoud/Applicatie, Netwerk
Webservice	Inhoud/Applicatie, Logistiek, Netwerk
XML	Inhoud/Applicatie
JSON	Inhoud/Applicatie

REST architectuur zegt dus niets over de logistieke laag, iets wat juist de kern van SOAP WSDL is. REST implementaties maken hier geen gebruik van, of passen toch één of meerdere standaarden uit de logistieke laag toe of lossen logistieke vraagstukken op een ander niveau op.

Verschillen REST met SOAP/WSDL gebaseerde interactie

Veel bestaande overheidsstandaarden zijn gebaseerd op SOAP en WSDL zoals StUF, Digikoppeling WUS en SuwiML. Of alleen SOAP zoals Digikoppeling ebMS en JAB/JUBES. Hieronder bekijken we hoe REST zich hiervan onderscheidt op basis van een aantal thema's waarin de belangrijkste eigenschappen van REST en SOAP/WSDL terugkomen. Eerst kijken we naar formele relaties, een sterk punt van SOAP/WSDL. Daarna naar prestaties, een sterk punt van REST. Vervolgens naar Code generatie waar beide voor en nadelen hebben en tot slot naar beveiliging. Dit is samengevat in onderstaand schema. Meer uitleg over de precieze betekenis van de begrippen en vergelijkingspunten staat in bijlage A en B.

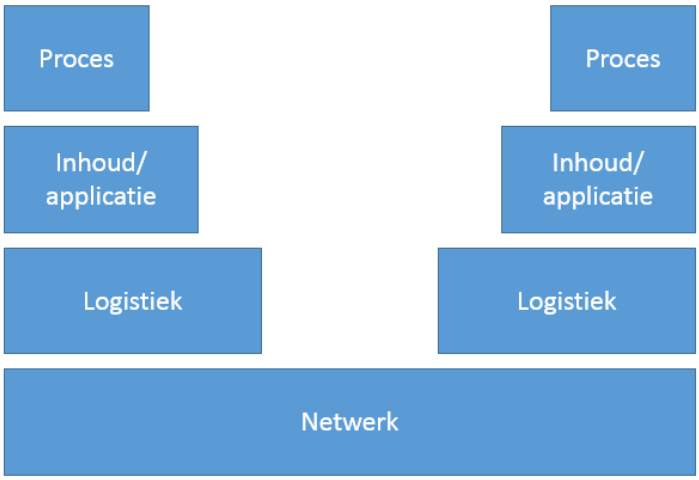
Eigenschappen	**REST**	**SOAP/WSDL**
Formele relaties		
Bericht validatie	Is mogelijk indien de inhoud van het bericht (XML)schema ondersteunt. Vaak wordt JSON gebruikt waarbij dit (nog) niet kan.	Is een uitgangspunt van de standaarden. XML wordt bijna altijd gebruikt waarbij validatie middels XML schema(XSD) mogelijk is. Het is wel mogelijk een bericht formaat te versturen dat geen schema kent
Adverteren webservice	Niet altijd nodig bij REST, kan wel middels WSDL[17], WADL, OpenAPI/Swagger en een aantal leverancier specifieke formaten. Echter geen onderdeel van REST en ondersteuning in software is niet altijd aanwezig.	Kent eenduidige breed ondersteunde standaard maar alleen middels WSDL.
Reliable messaging	Biedt de REST architectuur geen mogelijkheden voor, dient op applicatie niveau opgelost te worden.	Wordt ondersteund middels interoperabele standaarden in de logistieke laag.
Onweerlegbaarheid	Zegt de REST architectuur niets over, dient op applicatie niveau opgelost te worden.	Wordt ondersteund middels interoperabele standaarden(WS-Security, WS-I basic security profile 1.0) in de logistieke laag.
Adressering/routering	Wordt alleen ondersteund op netwerk niveau middels HTTP. Het kennen van de URLs van machines is altijd vereist	Zijn standaarden voor op logistiek niveau(o.a. ws-adressing), vereist slechts het kennen van identiteit van de uiteindelijke machine, niet de URL.
Prestaties		
bandbreedte	Gebruikt weinig bandbreedte door caching en simpele interfaces en het meestal toepassen van compacte uitwisselformaten als JSON	Gebruikt relatief veel bandbreedte door het ontbreken van caching, mogelijkheid voor complexe interfaces en het bijna altijd toepassen van het niet compacte uitwissel formaat XML

[17] Klinkt raar maar toch waar: WSDL een een beschrijvend metadata formaat en met WSDL 2.0 kan je ook REST webservices beschrijven

Response tijd	Zeer snel door simpele berichten en gebruik caching en simpele interfaces	Langzamer door complexere berichten en ontbreken caching en mogelijkheid voor complexere interfaces
Stateless	Een REST implementatie is altijd stateless, dit wordt vereist door de architectuur	Een SOAP/WSDL implementatie kan stateless zijn, maar is dit vaak niet. Voor reliable messaging is bijvoorbeeld bijhouden van state noodzakelijk. Dit gaat ten koste van de responsetijden
Schaalbaarheid	Zeer goed schaalbaar door toepassen layered system en caching	Goed schaalbaar door toepassen layered system
Code generatie		
webservice	Code voor een webservice kan gegenereerd worden als er gebruik wordt gemaakt van een standaard voor adverteren van de webservice. Dit ondersteunt het aanroepen van de webservice met berichten. Op basis van het meestal toegepaste JSON kan ook code gegenereerd worden die helpt bij het opstellen van berichten maar niet het aanroepen van de webservice.	Code generatie voor een webservice kan op basis van een WSDL altijd plaatsvinden, zowel voor het aanroepen van de webservice als het genereren van berichten. Bij soms voorkomende zeer complexe data structuren is de bruikbaarheid echter beperkt.
database	Genereren van een database schema kan als er gebruik wordt gemaakt van een uitwisselformaat met schema zoals XML. Dit is meestal **niet** het geval	Genereren van een database schema kan als er gebruik wordt gemaakt van een uitwisselformaat met schema zoals XML. Dit is meestal **wel** het geval
Beveiliging		
Definitie	Zegt de REST architectuur niets over moet apart geregeld worden. Kan in sommige van de standaarden voor het adverteren van webservices (zoals WSDL) wel meegenomen worden. Dit betekent niet dat uitwisseling volgens REST onveilig is	Kan in groot detail opgenomen worden in WSDLs zijn aanvullende standaarden voor als WS-Policy
interoperabiliteit	Werkt goed bij veel toegepaste profielen voor beveiliging van transport als PKI certificaten, Oauth en SAML. Het interoperabel krijgen van ondertekenen en versleutelen van berichtinhoud is lastiger omdat hier veel vrijheden in bestaan.	Alle mogelijke profielen, van veel gebruikt tot obscuur, zijn tot in detail middels WS-security te beschrijven en daarmee sneller interoperabel te implementeren. De meest voorkomende profielen staan in WS-I basic security profile en zijn door alle grote leveranciers onderling op interoperabiliteit getest.

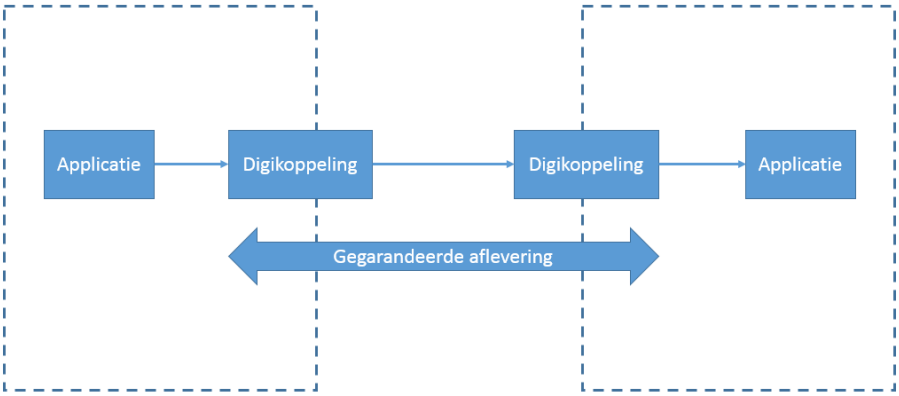
Logistiek in SOAP/WSDL en REST

Groot verschil tussen SOAP/WSDL en REST is dat REST geen logistieke laag kent, het slaat deze laag als ware over. SOAP/WSDL kent vele mogelijkheden in de logistieke laag die allemaal ondersteund worden door de protocolstandaarden (ws-security,WS-policy, WS-reliable messaging etc…). Beveiliging met PKI certificaten (PKIOverheid), SAMLtokens (DIGID, E-Herkenning Idensys), gegarandeerde aflevering en onweerlegbaarheid (Digikoppeling reliable messaging) zijn allemaal in standaarden gevat bij SOAP/WSDL. Aangezien REST zelf geen standaard is en niets over de logistiek zegt zijn deze functionaliteiten lastiger te regelen met REST.
Dit komt ook doordat SOAP/WSDL geen eisen stelt aan “state”(REST is expliciet stateless). Een server die een SOAP/WSDL webservice aanbiedt kan dus ook stateful zijn en precies bijhouden waar een client mee bezig is tijdens een sessie. Het bijhouden van state is bijvoorbeeld van belang bij het implementeren van gegarandeerde aflevering van berichten. Een functionaliteit die binnen de overheid vaak gevraagd wordt. Bij een RESTful implementatie kan dit niet op logistiek(protocol) niveau worden afgedwongen, maar wel op applicatie niveau.



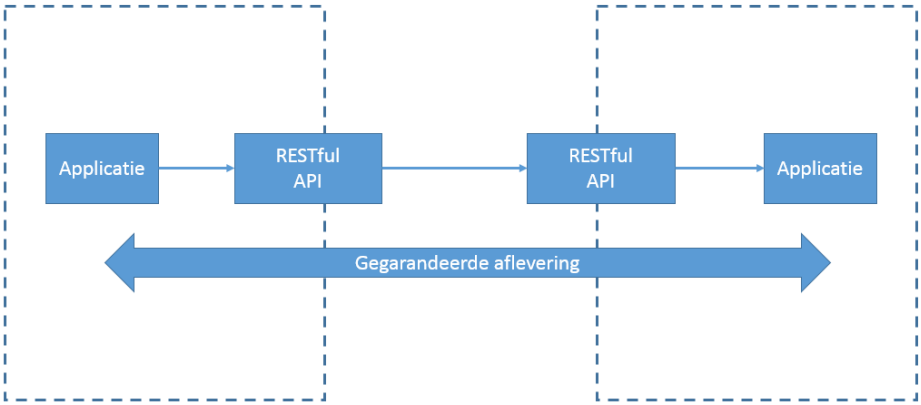
Figuur 2 De verschillende niveaus waarop afspraken moeten worden gemaakt voor gegevens uitwisseling

Het regelen op applicatieniveau heeft voor en nadelen. Voordeel is dat het voor de hele keten is geregeld (van applicatie tot applicatie) en niet tot de deurmat van de organisatie (tot de digikoppeling adapter) zoals bij Digikoppeling vaak het geval is.



Figuur 3 gegarandeerde aflevering bij Digikoppeling

Bij Digikoppeling is er geen garantie dat een bericht ook bij de applicatie is afgeleverd, daarvoor dienen ofwel binnen de organisatie maatregelen getroffen te worden of er moet ook op applicatieniveau een implementatie van gegarandeerde aflevering komen. Hetgeen dubbelop is.



Figuur 4 gegarandeerde aflevering bij RESTful APIs

Het nadeel is dat het toepassen van beveiliging op RESTful webservices bewerkelijker is omdat REST zelf geen standaard is en er per implementatie afspraken over gemaakt dienen te worden. Gelukkig zijn er een aantal beveiligingsstandaarden die veel met REST gebruikt worden als Oauth, SAML, TLS en PKI. Deze worden toegepast voor de beveiliging van het transport. Voor beveiliging van de berichtinhoud zijn er veel mogelijkheden, dit is lastiger interoperabel te krijgen bij REST. Er zijn ook methoden waarmee de beveiliging van RESTful APIs formeel beschreven kan worden:

- OpenAPI/Swagger
- WSDL 2.0 (ironisch maar waar)
- WADL
- Heel aantal specifiek voor één vendor

Ondersteuning hiervoor verschilt per applicatie en technology stack en er is niet een methode die door iedereen ondersteund wordt zoals dat bij SOAP wel geldt met WSDL. SOAP/WSDL kent bij WS-I[18] interoperabiliteitsprofielen waarin de mogelijke keuzes voor o.a. beveiliging worden terug gebracht tot configuraties die softwareleveranciers onderling getest hebben[19]. Voor REST is dit er niet, vooral ook omdat REST meestal niet toegepast wordt bij scenario's met complexe logistiek en beveiliging.

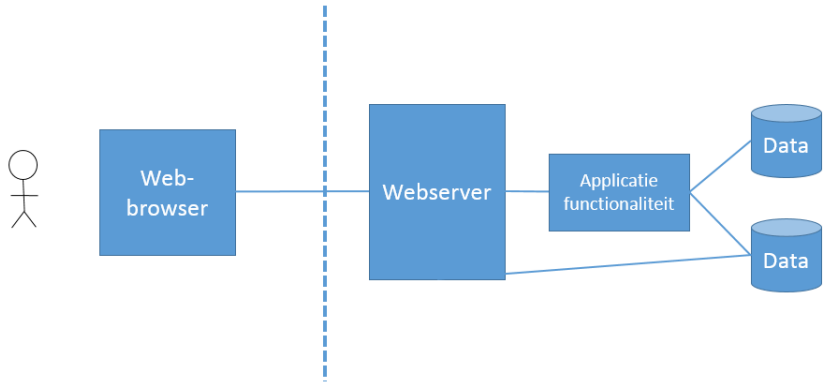
[18] http://www.ws-i.org/
[19] Ook ebMS kent interoperabiliteits testen door Drummond. Hier testen leveranciers hun implementaties tegen elkaar.

5. RESTful APIs en de overheid

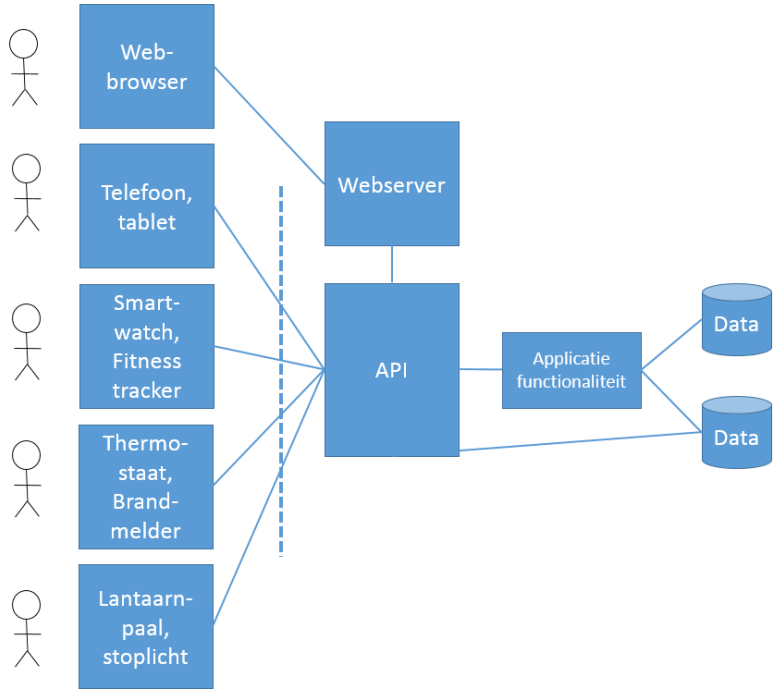
inleiding
Welke invloed heeft nu de opkomst van RESTful APIs? Mensen verwachten dat er steeds meer als vanzelfsprekend geregeld is. Voor ieder mogelijk probleem is er een handige app om je te helpen. Je huis zorgt ervoor dat met slimme apparaten de temperatuur aangenaam is en je telefoon vertelt wanneer je moet vertrekken om op tijd op je werk te zijn. Snelle en efficiënte koppelingen zijn nodig en dit wordt mogelijk via RESTful APIs. Er wordt steeds meer gebruik gemaakt van REST en dit zie je terug in ondersteuning voor REST in standaard software producten. Ook bij programmeurs zie je steeds meer kennis van REST ten opzichte van SOAP/WSDL. Door deze marktontwikkelingen kan het aantrekkelijk zijn om voor bestaande SOAP/WSDL uitwisselingsstandaarden om dezelfde inhoud via een RESTful API te gaan uitwisselen. De standaard CMIS kent bijvoorbeeld een uitwisselingsprofiel via SOAP/WSDL en één via REST.

Nederlandse overheid

Voor Nederlandse burgers en bedrijven, maar ook ambtenaren (medewerkers) betekent de opkomst van RESTful APIs dat ze hetzelfde van de overheid verlangen als wat ze gewend zijn van (commerciële) platforms. Deze platforms zijn slim in het creëren van nieuwe diensten door het koppelen van verschillende APIs. Daarnaast wilt men informatie en diensten van de overheid snel en plaatsonafhankelijk gebruiken op alle kanalen die ze gewend zijn. Dus niet alleen in hun browser, maar ook via apps op hun tablet, op hun smartTV, hun slimme thermostaat of iWatch. Ze zijn inmiddels gewend het internet via allemaal nieuwe kanalen te benaderen.



Figuur 5 communicatie met de gebruiker vroeger: alleen via de browser



Figuur 6 communicatie met gebruikers nu: vele kanalen

> ***Voorbeeld 6: Vergunningaanvraag 1***
> *Als je slimme thermostaat het weerbericht kan integreren bij het aangenaam houden van je huis zonder dat je er iets voor hoeft de doen, waarom kan een bouwbedrijf je dan niet een dakkapel aanbieden inclusief volledig afgehandelde vergunning aanvraag? Met behulp van een RESTFUL API kan dit proces efficiënt en snel ingeregeld worden.*

Om goed aan te geven waar voor de overheid de invloed van REST van belang is kijken we eerst naar de interactiepatronen van gegevensuitwisseling op macro niveau. Dit zijn de mogelijke toepassingsgebieden.

Interactie patronen binnen de overheid

We bekijken hieronder de interactiepatronen op macro niveau. Dus niet zozeer hoe twee specifieke machines met elkaar interacteren, maar hoe de verhoudingen in informatieketens liggen. Dit doen we zodat beter inzichtelijk is voor welke type interactiepatronen Restful APIs een grote toegevoegde waarde hebben. In de interactiepatronen binnen de overheid is er een glijdende schaal te zien in de verhouding tussen communicerende partijen. Van communicatie met vele onbekende gebruikers, naar communicatie met vele bekende gebruikers, naar herhalende keten processen tot unieke ketenprocessen. Het aantal betrokken gebruikers wordt hierbij langzamerhand steeds kleiner.

Communicatie naar vele onbekende gebruikers

De overheid heeft een dienst die zij naar vele gebruikers wil ontsluiten, vaak zijn niet alle potentiële gebruikers zijn van tevoren bekend. Voorbeeld bij uitstek is het ontsluiten van open data naar burgers, bedrijven en overheden. Het potentieel aan gebruikers is enorm en ze zijn bij publicatie lang niet allemaal in beeld. Er is geen formele relatie tussen gebruiker en

aanbieder (overheid), het doel is juist dat zoveel mogelijk partijen hergebruik maken van de eigen informatie waarbij gebruikersvriendelijkheid belangrijk is. Hier past een RESTful API dus zeer goed.

> ***Voorbeeld 7: Smart Cities en Open Data***
> *In het kader van smart cities, delen gemeenten veel van hun data als opendata en zoeken de samenwerking op met app developers van commerciële partijen in de hoop dat op basis van open data bezoekers en inwoners van de stad op nieuwe en slimme manieren geholpen kunnen worden. De data wordt 'as-is' aangeboden en het doel is zoveel mogelijk gebruikers bereiken. Je hebt bij deze interactie patronen potentieel te maken met miljoenen gebruikers(alle burgers en bedrijven in Nederland) en tienduizenden programmeurs die voor hen applicaties ontwikkelen.*

Communicatie naar vele bekende gebruikers
Het lijkt op het eerste interactiepatroon met als verschil dat het van belang is dat je als aanbiedende partij weet wie je afnemers zijn. Dit kan zijn om dat er beperkingen liggen op de informatie die de server aanbiedt (persoonsgegevens uit de BRP) of bijvoorbeeld omdat het nodig is om gebruik bij te houden ten behoeve van financiering. Doel is nog steeds dat je gegevens door zoveel mogelijk partijen gebruikt worden. Door meer gebruik is er meer potentieel voor kwaliteitsverbetering en kostenreductie. Gebruikers verlangen echter ook kwaliteit en SLA's worden daarom afgesloten. Hiervoor is een formele relatie nodig, gebruikers van basisregistraties bijvoorbeeld hebben een formele relatie met de houder landelijke voorziening, er worden aansluitvoorwaarden overeengekomen en gebruikers zijn bijvoorbeeld verplicht tot terugmelden. Belangrijke mutaties uit registraties(bijvoorbeeld Digilevering) moeten gegarandeerd afgeleverd worden bij de gebruiker. Ook in relaties met bedrijven kan het van belang zijn een API aan te bieden waarbij je weet wie de andere partij is. Denk aan de gegevens die bedrijven via Digipoort aan de overheid aanleveren, ook hier weet je als overheid niet altijd precies hoeveel afnemers je kan bereiken maar je wil er zoveel mogelijk. Dit type uitwisseling vraagt ook om technische maatregelen zoals: Beveiliging (versleuteling van berichten, authenticatie, autorisatie) en gegarandeerde aflevering.

Je hebt bij dit interactiepatroon te maken met potentieel honderdduizenden gebruikers (alle bedrijven, overheden en semi overheden). Deze zullen worden bediend door duizenden programmeurs. Qua het bereiken van gebruikers en gebruiksvriendelijkheid past REST nog steeds goed en beveiligde verbindingen zijn ook via REST mogelijk. Indien er teveel complexe interacties (beveiliging, gegarandeerde aflevering etc…) bij elkaar worden toegepast neemt de geschiktheid echter af en is de ondersteuning voor complexe interacties die SOAP/WSDL biedt de betere oplossing.

Herhalende ketenprocessen
Bij ketenprocessen vindt uitwisselingen niet tussen twee partijen plaats maar is er een keten van meer dan twee partijen (applicaties en/of organisaties) die gezamenlijk één uitwisseling verzorgen. De overheid kent ketenprocessen binnen tal van domeinen die op meerdere plekken in het land worden geïmplementeerd. Denk hierbij aan de implementatie van de uitvoeringsprocessen van gemeenten. Deze worden in Nederland 400 keer neergezet. Hierbij is er een softwaremarkt met 150+ pakket leveranciers die schakels uit de keten leveren die

vervolgens met andere leveranciers moeten samenwerken. Voor de overheid als opdrachtgever is het van belang dat er APIs zijn die door zo veel mogelijk leveranciers ondersteund worden. Door APIs laagdrempelig te maken kunnen meer leveranciers deelnemen wat monopolies kan voorkomen. Dit bevordert de leveranciersonafhankelijkheid.

Ketenprocessen kennen daarnaast vaak een wat gelijkwaardiger relatie tussen de partijen die met elkaar communiceren. Alle partijen zijn vaak zowel client als server. De relatie tussen alle partijen in de keten is ook een zeer formele. Gezamenlijk zijn ze verantwoordelijk voor het resultaat, maar als er iets mis gaat is het belangrijk om aan te kunnen wijzen waar in de keten de verantwoordelijkheid ligt. Dit vraag om afspraken over datgene dat tussen ketenpartijen wordt uitgewisseld. Bijvoorbeeld over beveiliging, gegarandeerde aflevering en validatie van berichten. Ook kan hier naast synchrone communicatie ook a-synchrone communicatie om de hoek kijken. Wanneer in een ketenproces een antwoord weken op zich kan laten wachten houd je hier geen verbinding voor open maar stuur je een a-synchroon antwoord. In een herhalend ketenproces kan je te maken hebben met tienduizenden gebruikers (vooral medewerkers van decentrale overheden) en zeker duizend programmeurs die actief zijn in de gemeentelijke softwarepakket markt. Voor het makkelijk maken voor programmeurs is het aanbieden van RESTful APIs in deze ketens zeer geschikt en wenselijk, wanneer de interacties te complex (er moeten (te)veel aanvullende afspraken worden gemaakt) neemt de wensbaarheid echter af.

Unieke keten processen
Dit zijn ketenprocessen die slechts één keer voorkomen bijvoorbeeld tussen een aantal centrale overheden. Hiervoor is geen ‘pakket’ in de markt waarmee deze ketens worden geïmplementeerd. Men maakt gebruik van internationale Common Of The Shelf software en/of van in opdracht gebouwde maatwerk oplossingen. De allergrootste unieke keten in de Nederlandse overheid is de loonaangifte keten waarin belastingdienst CBS en UWV samenwerken. Net als bij de herhalende keten processen hebben ketenpartijen hierbij een zeer formele relatie met elkaar. Eén uniek ketenproces hoeft soms maar enkele tientalen direct betrokkenen te kennen vooral bij centrale overheden(bij echt grote ketens als de loonaangifte is dit natuurlijk veel meer). Zij worden bediend door maximaal tientallen programmeurs. REST zou je hier toepassen vanwege praktische overwegingen: mogelijk betere ondersteuning in standaard software of vanwege kennis in de markt. Echter qua interactie past SOAP/WSDL hier ook heel goed zeker als er zeer formele relaties tussen ketenpartners zijn.

Invloed REST op interactiepatronen van de overheid
Ten behoeve van de gebruiksvriendelijkheid, interoperabiliteit en leveranciers toegankelijkheid is het gewenst dat de Nederlandse overheid in haar communicatie met vele bekende en onbekende gebruikers(vooral burgers en bedrijven) APIs aanbiedt (REST of SOAP APIs). Deze APIs zijn pas echt waardevol als ze zoveel mogelijk worden aangesproken door derde (commerciële) partijen die zodoende (extra) dienstverlening kunnen bieden aan gebruikers (burgers en bedrijven). In zulke gevallen past dus het aanbieden van RESTful APIs zodat het de programmeurs van innovatieve toepassingen zo makkelijk mogelijk wordt gemaakt. Het gaat dan zowel om de interactiepatronen waarbij je veel onbekende gebruikers wil bereiken (data.overheid.nl, CBS, Geo-informatie) als de patronen waar je veel ‘bekende’ wilt bereiken (Digipoort, ondernemersdossier, zaaksystemen).

Bij herhalende ketenprocessen kan je ook lessen leren van RESTful APIs. Je hebt hier vaak een 'gesloten' proces met een grote hoeveelheid programmeurs die met dezelfde standaarden, programma's en APIs bezig zijn. Door het hen makkelijker te maken creëer je meer concurrentie vooral op de complexere systemen. In dit geval is het doel dat meer concurrentie leidt naar betere en goedkopere implementaties. Je hoeft hier niet direct volledige RESTful APIs voor te ontwikkelen, er zijn ook hele goede redenen om in deze usecase op SOAP/WSDL APIs te zitten om bijvoorbeeld:

- Gedetailleerd kunnen beschrijven van uitgewisselde informatie;
- Het kunnen valideren van uitgewisselde informatie;
- In het protocol ingebouwde beveiligingsmechanismen;
- Sluit vaak aan bij het nu werkende bestaande aanbod binnen de overheid.

Bij het aanbieden van APIs is het vooral van belang om het programmeurs makkelijk te maken. Dit kan ook door kant en klare bibliotheken aan te bieden en/of voorbeeld implementaties welke programmeurs makkelijk in hun eigen code kunnen verwerken. Een voorbeeld is het bevragen van gegevensmagazijnen bij gemeenten. De bestaande API liet zoveel vrijheden dat het moeilijk was voor programmeurs deze op een gegevensmagazijn te implementeren. KING werkt nu samen met gemeente Den Haag aan een API die simpeler is te implementeren voor zowel client als server. Dit wordt bereikt door alleen die informatie uit te wisselen die echt nodig is en niet meer dan dat.

Voor unieke ketenprocessen kunnen RESTful APIs ook kansen bieden. De internationale 'common of the shelf' software die daarin wordt gebruikt zal vaak meer en betere ondersteuning voor REST hebben. Bij maatwerk oplossingen voor een unieke keten heb je vaak niet te maken met legacy standaarden dus is het makkelijker om over te stappen op een nieuwe uitwissel methode en voorkom je sneller een 'lock in' situatie.

Standaarden PTOLU

Om te illustreren wat de invloed van REST is op de lijst met standaarden van het Forum, gaan we dieper in op een aantal standaarden in detail. Voor de overige standaarden doen we dit op een hoog abstractie niveau.

Digikoppeling

Bij Digikoppeling staat betrouwbaarheid en formaliteit van de gegevensuitwisseling voorop, de standaard is dan ook gebaseerd op SOAP en vereist (in alle profielen) tweezijdige authenticatie. Het toepassingsgebied van Digikoppeling kan wel zorgen voor een conflict met het toepassen van RESTful APIs. Immers volgens het huidige werkingsgebied zou bijvoorbeeld ook open data uitgewisseld moeten worden tussen overheden onderling. Digikoppeling zou dus niet verplicht moeten worden bij het uitwisselen van open data of in ketens daar waar authenticatie en betrouwbaarheid minder relevant is dan snelheid en efficiëntie. Een aanscherping van het werkingsgebied van Digikoppeling zou helpen.

StUF

StUF is in de eerste plaats een uitwisselingsformaat (inhoud van de envelop) voor transport is er een zogeheten protocol binding. De huidige StUF protocol bindingen zijn op basis van SOAP, echter een uitbreiding van deze protocolbindingen naar REST zou heel goed mogelijk

zijn en tast de essentie van de standaard niet aan. De StUF standaard is al bezig met het lering trekken uit de trend voor RESTful APIs. Zo wordt er gewerkt aan het aanbieden van StUF in JSON als ook het definiëren van scherpere webservices in samenwerking met gemeente Den Haag. Een RESTful API zal niet overal mogelijk zijn of zin hebben, maar daar waar er voordeel te behalen is wordt het mogelijk.

Geostandaarden
De geostandaarden (NEN3610) zijn gebaseerd op de OGC standaarden welke in de basis min of meer een REST architectuur volgen. Geostandaarden gebruiken tot nu toe vooral XML of plaatjes als bericht inhoud. Vooral de gebruikte XML standaarden zijn relatief complex en voor simpele toepassingen overkill. Er wordt gewerkt aan het toepassen van echte REST services op basis van GeoJSON om simpele toepassingen te ondersteunen.

SAML
De authenticatiestandaard SAML(toegepast in DigiD en E-Herkenning) kan met zowel SOAP/WSDL als REST worden toegepast. De standaard ondervindt niet direct invloed van de opkomst van REST, maar is wel op XML gebaseerd en zwaarder om te implementeren. SAML wordt in de praktijk met name toegepast in een 'bedrijfsomgeving' voor toegang via websites en via SOAP services. Oauth is een alternatief voor SAML en mogelijk een kandidaat voor de lijst. De standaard is zeker voor RESTful APIs een waardevolle toevoeging en wordt voornamelijk gebruikt voor autorisatie van internetbronnen. SURF heeft een goede verkenning gedaan van de mogelijkheden van Oauth ten opzichte van SAML https://blog.surf.nl/wp-content/uploads/2013/04/SURFnet-OpenID-Connect-1.1-.pdf

Hieronder geven we de relatie weer die REST mogelijk heeft met een standaard. Dit betekent dat wanneer een standaard mogelijk in de toekomst zou moeten veranderen door de opkomst van REST dat er dan invloed is op de standaard. Let op als een standaard met het REST principe kan samenwerken dan heeft de opkomst daarvan geen invloed: de standaard hoeft dus niet te veranderen. Vooral standaarden die SOAP/WSDL gebruiken of een uitwisseling kennen die op REST lijkt zullen invloed kunnen ondervinden. Onderstaande PTOLU lijst is gebaseerd op de versie van begin februari 2016.

Standaard	Invloed REST	Uitleg
Aquo-standaard	Ja	Gegevensverzameling, -vastlegging en -uitwisseling voor het beheer van waterkeringen, oppervlaktewater en afvalwaterzuivering. Dit zou ook via een RESTful API kunnen en met JSON formaten. Overigens worden er ook bestandsformaten via JSON aangeboden
BWB	Ja, gebruikt SOAP/WSDL/ XML	Elektronische verwijzing naar (delen van) geconsolideerde wetten en regelingen met het doel om deze met anderen te delen. De BWB webservices is gebaseerd op WSDL en SOAP.
CMIS	Ja, ondersteunt al REST	Het toegankelijk maken van ongestructureerde gegevens in content CMS'enDMS'en. Oorspronkelijk op SOAP gebaseerd, maar er zijn drie protocolbindingen gedefinieerd. 1 voor SOAP, 1 voor Atompub (RESTful XML) en 1 voor JSON. Overigens moet voor CMIS altijd aanvullende afspraken worden gemaakt .
Digikoppeling	Ja, uitleg hierboven	Geautomatiseerde gegevensuitwisseling tussen informatiesystemen voor sectoroverstijgend berichtenverkeer,

DKIM	Nee	Het faciliteren van het vaststellen van organisatorische herkomst van e-mail. Geen specifieke relatie met REST.
DNSSEC	Nee	Het registreren en in DNS publiceren van internet-domeinnamen ('signing'). Hierdoor is validatie van het domeinnaam mogelijk. Geen specifieke relatie met REST.
E-portfolio NL	Nee	Het uitwisselen van informatie over de ontwikkelingsvoortgang van een individu, Is wel gebaseerd op XML en de IMS specificatie. Deze kan echter via Restful principe of SOAP worden uitgewisseld.
ECLI	Nee	Identificatie ter citatie van gepubliceerde rechterlijke uitspraken. Zegt niks over de protocolbindingen.
EML_NL	Nee	De definitie en uitwisseling van kandidaatgegevens en uitslaggegevens bij verkiezingen welke onder de Nederlandse Kieswet vallen. Uitwisseling kan alleen als XML bestand en niet als JSON, maar zegt niks over de hoe het uitgewisseld moet worden.
Geo Standaarden	Ja, zie uitleg hierboven	Uitwisseling van geografische informatie tussen organisaties, waarbij de ruimtelijke dimensie van significant belang is
IFC	Ja,	Uitwisseling in het kader van bouwwerkinformatiemodellen. IFC kent al een RESTful API
IPv6 en IPv4	Nee	Voor communicatie van computernetwerken over organisatiegrenzen heen tussen organisaties, individuele eindgebruikers, apparaten, diensten en sensoren. Zegt niks over REST of SOAP.
JCDR	Nee	Identificatie van geconsolideerde decentrale regelgeving en een gestandaardiseerde manier om hiernaar elektronisch te verwijzen zegt niks over REST of SOAP.
NEN-ISO/IEC 27001 / 2	Nee	Specificeren van eisen voor het vaststellen, implementeren, uitvoeren, controleren, beoordelen, bijhouden en verbeteren van een gedocumenteerd Information Security Management System. Is een organisatorische standaard en zegt in principe niks over de techniek.
NL LOM	Nee	Metadatering van content die ontsloten wordt ten behoeve van educatieve doeleinden. Maakt wel gebruik van XML schema's, maar zegt niks over het aanbieden en uitwisselen.
NTA 9040	Ja,	De standaard is van toepassing voor overheden die expliciet met ondernemingen hebben afgesproken een Ondernemingsdossier in te zetten voor de informatie-uitwisseling met ondernemingen. De standaard heeft een relatie, schrijft namelijk expliciet een WSDL voor.
OAI-PMH	Ja	Het vraaggestuurd aanbieden en ophalen van verzamelingen metadata uit bibliotheken met (digitale) documenten of andere objecten. Een protocol gebaseerd op het REST principe.
ODF 1.2	Nee	Uitwisseling van reviseerbare documenten en zegt niks over de manier van uitwisselen.
OWMS	Ja	Metadateren van publieke overheidsinformatie op internet. Is syntax neutraal en net zoals OAI-PMH gebaseerd op de internationale Dublin Core standaard. Sluit goed aan op LinkedData wat goed samen gaat met het RESTful API.
PDF 1.7, A1 en A2	Nee	Het uitwisselen, publiceren en archiveren van niet- of beperkt-reviseerbare documenten. Zegt niks over de manier van uitwisselen.

SAML	Ja	Federatieve (web)browser-based single-sign-on (SSO) en single-sign-off. Dat wil zeggen dat een gebruiker na eenmalig inloggen via zijn browser toegang krijgt tot verschillende diensten. Voor de relatie zie uitleg hierboven.
Semantisch model e-factureren	Nee	De verzending en ontvangst van elektronische facturen door organisaties die deelnemen aan het economisch verkeer in Nederland. Is techniek neutraal.
SETU-standaard	Ja	Is een XML standaard voor elektronische berichtenuitwisseling rondom de bemiddeling/inhuur van flexibele arbeidskrachten. Zegt niet iets over de manier van uitwisselen, kan dus zowel via een RESTful API als via SOAP verbindingen.
SIKB0101 & SIKB0102	Ja	Uitwisselen van onderzoeksgegevens over de milieuhygiënische kwaliteit van de bodem. Voor SIKB0101 is er een specifiek SOAP webservice voorgeschreven, geen gebaseerd op REST.
SKOS	Ja	Het in een gestructureerde vorm op het Web publiek beschikbaar stellen van gegevenswoordenboeken. Is een typisch standaard die goed aansluit op REST, zijn ook RESTful APIs voor SKOS.
SPF	Nee	Het controleren of een e-mailserver gerechtigd is om namens een domeinnaam e-mail te mogen verzenden. Geen specifieke relatie met REST.
STOSAG	Ja	Digitaal container- en pasmanagement voor afval en grondstoffen, maakt gebruik van SOAP/WSDL en XML.
StUF	Ja	Uitwisseling en bevraging van basisgegevens die behoren tot een aantal wettelijk vastgestelde basisregistraties, zoals Personen (GBA), Adressen (BRA), Gebouwen (BGA), Kadaster (BRK), Nieuw Handelsregister (NHR) en Waarde Onroerende Zaken (WOZ). Zie uitleg hierboven.
TLS	Nee	Het met behulp van certificaten beveiligen van de verbinding (op de transportlaag) tussen client- en serversystemen of tussen serversystemen onderling. Geen specifieke relatie met REST.
VISI	Ja	Formele communicatie tussen partijen in de bouwsector, zowel grond-weg en waterbouw, de burger & utiliteitsbouw als de installatiebranche. Heeft richtlijnen met betrekking tot uitwisseling gebaseerd op SOAP.
WDO Datamodel	Misschien	Gegevensuitwisseling tussen het bedrijven dien bij grensoverschrijdend goederenverkeer betrokken zijn. Gaat via XML berichten, maar niks terug te vinden over SOAP of RESTful.
Webrichtlijnen	Nee	Voor de toegankelijkheid van webgebaseerde informatie-, interactie-, transactie- en participatiediensten. Geen specifieke relatie met SOAP of REST
XBRL en Dimensions	Ja	Elektronisch verkeer dat te kenmerken is als verantwoordingsverkeer waarin financiële informatie de kern vormt. Gaat over XML berichten en wordt met name gebruikt in combinatie met SOAP.

Figuur 7 PTOLU lijst februari 2016 met aangeven welke standaarden invloed kunnen ondervinden van de opkomst van REST

6. Welke standaarden missen er op PTOLU voor RESTful APIs?

Veel van bovenstaande standaarden kunnen zowel in combinatie met SOAP als RESTful APIs gebruikt worden of hebben niet direct een relatie. Op de lijst worden dan ook niet veel standaarden gemist die typisch gebruikt worden bij REST. Veel van de op XML gebaseerde standaarden zouden in veel gevallen op een REST manier uitgewisseld kunnen worden. Hierbij zou het van meerwaarde zijn als beheerders niet alleen kijken naar SOAP bindingen, maar ook kijken naar of hun standaard goed bruikbaar zijn binnen RESTful APIs. Dit is in dit onderzoek niet specifiek gecheckt, maar wat je wel zit is dat er bij verschillende standaarden wel uitgelegd wordt hoe deze te verwerken met SOAP, maar dat er niks gezegd wordt over REST of andere formaten als JSON. Bij het toetsen van standaarden op bruikbaarheid en gebruiksvriendelijkheid zou bijvoorbeeld goed meegenomen kunnen worden of de voorgeschreven XML code geschikt is voor RESTful APIs. Zoals bijvoorbeeld ook is gedaan in het onderzoek naar de StUF standaard.

Veel standaarden kunnen in principe samenwerken met RESTful APIs er zijn echter enkele standaarden die binnen RESTful APIs ook veel gebruikt worden. Dit zijn:

- HTTP 1.1 (staat op de lijst met aanbevolen standaarden)
- JSON (aanbevolen standaard, zie ook begrippenlijst)
- XML (aanbevolen standaard, zie ook begrippenlijst)
- URI (aanbevolen standaard)

Andere standaarden die relevant zijn en terug en getoetst zouden kunnen worden voor de lijst met standaarden zijn:

- Oauth: een authenticatie standaard
- OData: een standaard voor het beschikbaarstellen van data middels REST
- Atompub: een op XML gebaseerd alternatief voor JSON bij REST services

Over Oauth
Oauth is autorisatiestandaard en een alternatief voor SAML en mogelijk een kandidaat voor de lijst en zal zeker voor REST principes een waardevolle toevoeging vormen. Het is voor telefoons, tablets, wearables, en internet of things apparaten een veel vaker gebruikte en beter ondersteunde beveiligingsstandaard. Juist voor de kanalen waar REST sterk is heb je dus Oauth nodig.

Voorbeeld 8: Vergunningsaanvraag 2
Vanuit dienstverlening kan het voor een bouwbedrijf interessant zijn om de bouw van bijvoorbeeld een dakkapel inclusief vergunningaanvraag aan te bieden. Zo is het bouwbedrijf voor de klant de ingang om alles voor de dakkapel te regelen. Voor een bouwbedrijf is dit interessant als hij kan aansluiten op een (niet bestaande) API van het OmgevingsLoket Online. Het bouwbedrijf kan de aanvraag dan via een RESTful API dat beveiligd is via Oauth doorzetten naar het OLO. Daar wordt het automatisch afgehandeld en weer teruggekoppeld.

Over OData
OData is een OASIS standaard[20] voor het toepassen van RESTful APIs en geeft aan hoe je daar o.a. Atompub en JSON kan gebruiken. Het biedt dus het voordeel dat het makkelijker wordt op basis van deze standaard interoperabele REST implementaties te maken. Het geeft een eenduidige manier van het toepassen van REST. Het is gericht op het ontsluiten van data en daarmee (bij de overheid) voor met name open data zeer interessant. Wel moet goed uitgekeken worden dat een keus voor OData het gebruik van linked open data standaarden als SPARQL niet uitsluit.

Over Atompub
Atompub is een bij RESTful APIs populair uitwisselingsformaat net als JSON. Atompub is op xml gebaseerd en komt voort uit de wereld van webfeeds. Het wordt o.a. veel toegepast door bloggers om hun artikelen te verspreiden. Websites kunnen de blogs via Atompub automatisch oppakken en herpubliceren(syndicatie). Er zijn echter ook vele andere soorten van data die geschikt zijn om op deze manier te delen. Voor de lijst met standaarden is deze niet direct relevant om te toetsen.

7. Conclusies en Aanbevelingen
In dit document hebben we inzichtelijk gemaakt dat REST en RESTful APIs een onmiskenbare trend is met betrekking tot gegevensuitwisseling. Met name daar waar veel digitale diensten met elkaar samenwerken en die informatie op een makkelijke en toegankelijke manier willen delen zijn RESTful APIs zeer geschikt. Van deze techniek is veel kennis in de markt, het is snel en laagdrempelig te implementeren. Dit betekent niet dat de overheid volledig over moet naar REST, SOAP/WSDL kent ook voordelen en zeer legitieme toepassingen. Daar waar betrouwbaarheid en formaliteit het belangrijkst is, is berichtuitwisseling via SOAP en WSDL te prefereren.

De standaarden op de lijst zijn over het algemeen te combineren met de REST principes, maar hoeven daar niet specifiek voor geschreven te zijn. Het zijn veelal XML standaarden die vaak gebruikt worden in combinatie met SOAP, dat kan betekenen dat er in de XML veel complexe structuren worden voorgeschreven die niet flexibel zijn voor REST. Dat betekent niet direct dat de lijst het gebruik van RESTful APIs in de weg zit. Maar je ziet wel dat bestaande standaarden op de lijst zoals StUF en de geostandaarden al proberen om REST principes te verwerken. Door het Forum zou bijvoorbeeld bij de beoordeling van standaarden op de lijst ook meegenomen kunnen worden of ze geschikt zijn voor RESTful APIs. Een andere optie is om ook bij andere standaarden die invloed ondervinden van REST, toepassingsgebieden aan te scherpen, zoals bij Digikoppeling het geval is.

APIs richten zich met name op de softwareontwikkelaar en op de interactie met burgers en bedrijven via andere kanalen dan websites. Het is een andere manier om gegevens te delen en interoperabiliteit te bewerkstelligen. Een API biedt als het ware een kant en klaar pakket aan de software-ontwikkelaar waarmee informatie is te hergebruiken en te integreren is in zijn toepassing. De lijst met standaarden heeft vooralsnog onvoldoende aandacht voor deze nieuwe vormen van interactie en interoperabiliteit. Om als Forum beter op deze nieuwe

[20] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=odata

ontwikkeling en trend in te spelen is het aan te raden om in ieder geval standaarden als Oauth en OData te toetsen voor opname op de lijst om zodoende beter aan te haken bij deze ontwikkeling. Een andere meerwaarde is om te onderzoeken of er behoefte is aan een handleiding voor het publiceren van APIs door overheden. In principe wordt dit nu aan de 'markt' zelf overgelaten en zijn er verschillende standaardisatieorganisaties die het zelf uitzoeken. Zo zijn er op het gebied van open data verschillende APIs beschikbaar. Deze zijn echter niet uniform en is er niet zoiets als een interoperabel profiel. Dit maakt ze lastiger bruikbaar en toegankelijk.

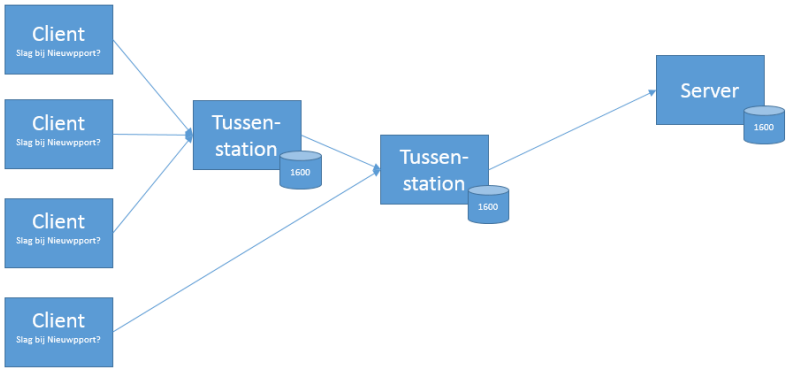
Aanbevelingen:
Als een standaard goed aansluit op de REST principes dan zegt dit iets over de toekomstvastheid en gebruiksvriendelijkheid van een standaard. Het verdient dan ook de aanbevelingen dat het Forum aandacht heeft voor deze vernieuwende ontwikkeling en bij het verplichten van standaarden meer oog heeft voor de praktisch implementatie van de standaarden in de praktijk. Daarom zijn onderstaande aanbevelingen geformuleerd die het Forum kan oppakken.

1. Toetsen Standaarden: Het Forum kan op eigen initiatief de standaarden Oauth en Odata toetsen voor opname op de lijst met standaarden. (*Deze aanbeveling is overgenomen*)
2. Kwaliteitstoets: enkele standaarden van de lijst zouden getoetst kunnen worden in hoeverre ze aansluiten en geschikt zijn voor REST. Dit zou samen met de beheerorganisaties moeten worden opgepakt en worden geanalyseerd. (*Deze aanbeveling is niet overgenomen, dit is meer een taak van de beheerorganisatie*)
3. Handreiking RESTful APIs: Het Forum heeft eerder handreikingen gepubliceerd over WEB of APP en Authenticatieniveaus. APIs sluiten nauw aan op het uitwisselen van gegevens en interoperabiliteit. Het ligt dan ook voor de hand om ook over dit thema een handreiking te publiceren en in te gaan op hoe overheden RESTful API moeten publiceren. Wel zal eerst onderzocht moeten of hier behoefte aan is. (*Deze aanbeveling is overgenomen*)

Bijlage A REST architectuur

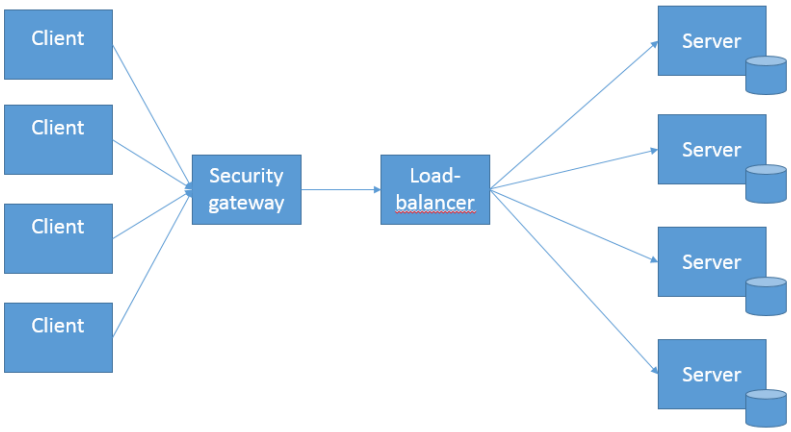
REST bereikt zijn belangrijkste eigenschappen (prestaties, schaalbaarheid en simpele interfaces) door een aantal beperkingen (contraints) op te leggen:

- **Expliciet onderscheid tussen Client en Server**. Zodat beide zich goed op hun taak kunnen richten. De client stelt vragen aan de server en houdt zelf bij waar het mee bezig is, waar in het proces de vraag zich bevindt (state) en wat het aan een gebruiker presenteert (user interface).
  De server beantwoordt de client en houdt zich bezig met dataopslag. Hoe de server data opslaat hoeft de client niet te weten, alleen dat hij doet en dat het op de server op te vragen is. Deze duidelijke scheiding verhoogt de schaalbaarheid. Je kan Eenvoudig het aantal clients en servers uitbreiden zonder dat dit grote impact heeft op de prestaties.
- **Stateless:** De server houdt niets bij over waar de verschillende clients waar het mee communiceert mee bezig zijn. De server hoeft alleen de vragen van het huidige moment te beantwoorden en verder nergens rekening mee te houden. Dit bevordert de prestaties.
- **Cacheable:** Bij antwoorden die een server geeft kan het expliciet aangeven of deze cacheable zijn. Veelvoorkomende vragen kunnen zo veel sneller uit cache beantwoord worden. Dit verhoogt de prestaties en schaalbaarheid enorm.



In bovenstaand plaatje wordt caching geïllustreerd. Stel een heleboel clients willen weten wanneer de slag bij nieuwpoort was. Het antwoord is iedere keer hetzelfde en daardoor zeer geschikt voor caching. De eerste keer dat de vraag wordt gesteld wordt het antwoord bij de server opgehaald. Daarna kunnen tussenstations die (voor de gebruiker niet zichtbaar) gebruikt worden in het transport dit antwoord opslaan in hun eigen cache. Ze kunnen de volgende clients die dezelfde vraag stellen daarmee veel sneller antwoord geven. De server wordt daarbij ontzien zodat hij meer capaciteit over heeft om moeilijke vragen snel te verwerken.

- **Layered system:** Een client kan niet zien of hij rechtstreeks met een server praat of dat er tussenpartijen tussen zitten zoals load balancers, (security) gateways etc… welke worden gebruikt om de schaalbaarheid te vergroten en welke prestaties en schaalbaarheid verbeteren door caching toe te passen.



In het plaatje hierboven zie je een layered system uitgewerkt. Iedere van de clients stelt zijn vraag maar heeft geen idee dat er eigenlijk 4 servers, een loadbalancer en een security gateway betrokken zijn. Hij wordt gewoon doorgeleid naar 1 van de 4 servers en krijgt daar antwoord van zonder dat hij iets van de infrastructuur hoeft te weten. Zo kan je de implementatie van een webservice heel makkelijk opschalen naar meer capaciteit.

- **Code on demand (optioneel):** Een server kan naar een client code sturen om lokaal uit te voeren bijvoorbeeld applets, of java script. Dit hoeft niet toegepast te worden (het is optioneel).
- **Uniforme interfaces**
  De interfaces waarmee communicatie tussen componenten plaatsvindt is zoveel mogelijk uniform. Hierbij wordt er gezorgd voor een ontkoppeling tussen client en server. Daarvoor zijn er een aantal ontwerp principes voor interfaces
  - **Identificatie van resources**
    Resources, bijvoorbeeld data worden in berichten teruggegeven als URIs die verwijzen naar de resource zelf. De representatie van resources zoals teruggeven van de server is daarbij volledig onafhankelijk van de interne representatie die een server gebruikt. Dus een server kan data opslaan in een SQL database maar teruggeven aan de cliënt in JSON, XML of een ander formaat dat de Client graag wil hebben. De daadwerkelijk resource en wat de Client krijgt is zo ontkoppeld. Dit zorgt dat er gemakkelijk en flexibel met uitwisselingsformaten omgegaan kan worden.
  - **Manipulatie van resources**
    Een client die in het bezit is van een representatie van een resource(een stukje XML of JSON gekregen van de server) heeft daarmee alle informatie die nodig is om deze resource te manipuleren: veranderen of verwijderen. Een client kan snel en eenvoudig die handelingen verrichten die hij wil verrichten en hoeft hiervoor niet eerst pagina's met documentatie door te spitten waarin toegestane functionaliteit staat beschreven.
  - **Zelf beschrijvende berichten**
    Ieder bericht bevat in zichzelf genoeg informatie zodat duidelijk is hoe dit bericht verwerkt kan worden. Het bericht geeft bijvoorbeeld zelf aan dat het in JSON formaat staat en met een JSON parser verwerkt kan worden.

- **Hypermedia as the engine of application state**
  Het klinkt ingewikkeld maar wat hier wordt bedoeld is simpel. Client machines die met REST interfaces communiceren gedragen zich net zo als iemand die over HTML pagina's (een vorm van hypermedia) navigeert met een browser. Je kan alleen van de ene pagina naar de andere (van de ene state naar de andere) komen door het volgen van een hyperlink. Je neemt als client aan dat de server je alle informatie biedt die je nodig hebt om ergens te komen. Alles is expliciet en je hoeft er niet vanuit te gaan dat er meer mogelijk is dan dat wat je gepresenteerd wordt. Je wordt door de server stap voor stap via hyper links door de mogelijke pagina's(states) geleid.

Bijlage B SOAP/WSDL interactiepatronen

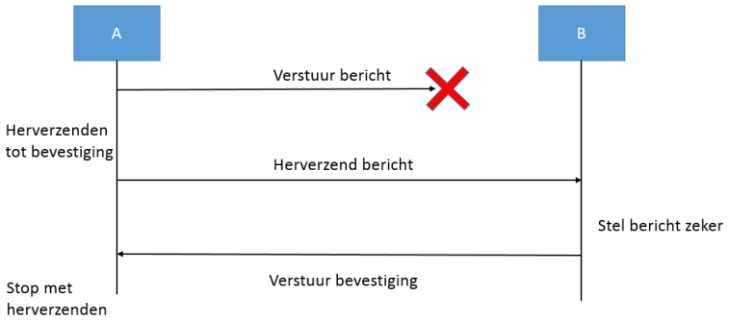
Bericht validatie
Uitwisselingen tussen overheden kunnen uit complexe informatiestructuren bestaan. Het is fijn als van een bericht gecontroleerd kan worden of deze voldoet aan vooraf afgesproken structuren. Hiervoor bestaat een oplossing, met behulp van schema's kan een bericht gecontroleerd worden. Een schema is een door een computer leesbaar metadata formaat dat beschrijft wat wel en niet in een bericht (of document) mag staan. Aan de hand van een schema kan voor verzenden en na ontvangst gecontroleerd worden of het bericht correct is. Zo niet dan kan er een foutmelding worden gegenereerd. Dit komt de kwaliteit en betrouwbaarheid van berichtuitwisselingen ten goede.

Reliable messaging
Een veel voorkomend patroon binnen de overheid is reliable messaging. Hierbij gaat het erom zeker te weten dat een bericht dat je verstuurt is aangekomen. Daarvoor stuurt de ontvanger een bevestiging terug wanneer het bericht ontvangen is.

Dit kan bijvoorbeeld relevant zijn voor het starten van wettelijke termijnen. Ter illustratie op het Omgevingsloket online (OLO)komt een vergunning aanvraag binnen. Deze wordt met reliable messaging doorgestuurd naar het bevoegd gezag, want bij indienen van de aanvraag begint de behandeltermijn te lopen. Het OLO wil dus zeker weten dat de aanvraag bij de behandelaar ligt zodat het OLO niet de schuld kan krijgen als de termijn verstrijkt en er is nog geen antwoord.

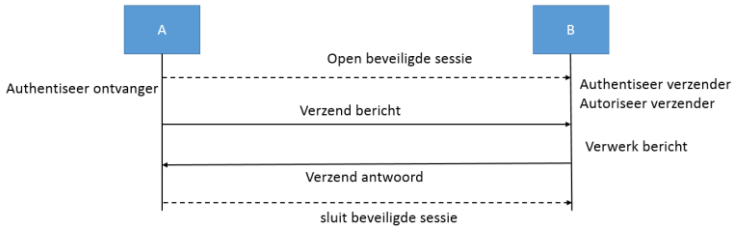
Was er geen reliable messaging toegepast dan had de aanvraag niet aan kunnen komen bij het bevoegd gezag door bijvoorbeeld een netwerkfout. Dan was de termijn verstreken zonder dat het bevoegd gezag ook maar wist dat het iets moest doen. Mocht er dus een fout optreden bij verzending dan blijft de verzendende partij met vooraf afgesproken

tussenpozen het bericht opnieuw aanbieden aan de ontvanger totdat de verzender wel een ontvangstbevestiging binnen heeft.



Beveiliging transport

Er zijn meerdere vormen van beveiliging van transportlaag. Hier geïllustreerd is de bij berichtuitwisseling meest voorkomende tweezijdige beveiliging.



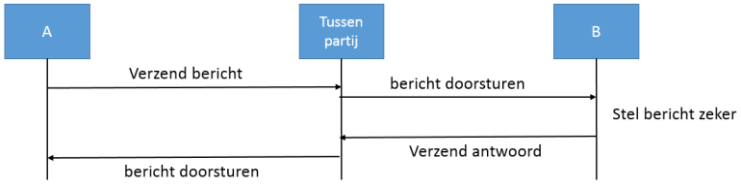
Hierbij wordt een sessie tussen twee partijen beveiligd. Eerst stellen ze elkaars identiteit vast op basis van een authenticatie middel. De aanbieder van de webservice autoriseert de vrager, samen versleutelen ze de sessie zodat de data die heen en weer gaat alleen voor zender en ontvanger te lezen is. Na het heen en weer sturen van informatie wordt de sessie weer gesloten. Deze techniek wordt toegepast zodra er gevoelige gegevens getransporteerd worden. Bijvoorbeeld bij het bevragen van het GBA wil je niet dat er iemand die geen toegang heeft (niet geautoriseerd is) toch meeluistert. Beveiliging van de transportlaag voorkomt dat mensen die het netwerk(internet, diginetwerk) afluisteren de gegevens kunnen lezen.

Onweerlegbaarheid

Deze techniek wordt toegepast om ervoor te zorgen dat de ontvangende partij zeker weet dat een bericht afkomstig is van de authentieke bron: de oorspronkelijke opsteller van het bericht. Bovendien weet de ontvanger dat er niet met het bericht geknoeid is. De oorspronkelijke opsteller van het bericht kan later niet meer ontkennen dat hij het opgesteld heeft.

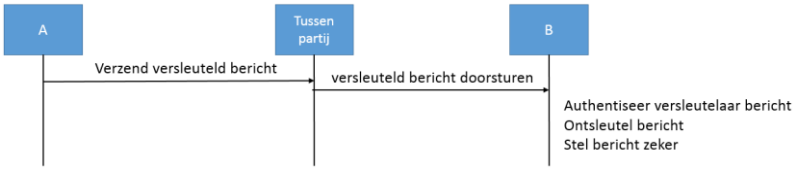
Dit wordt bijvoorbeeld toegepast voor bestemmingsplannen, deze worden in eerste instantie door de gemeente opgesteld en ondertekend en uiteindelijk via ruimtelijke plannen met de rest van de wereld gedeeld. De gebruiker van ruimtelijke plannen kan ervan op aan dat het echt het authentieke bestemmingsplan van de gemeente is.

Adressering/routering
Binnen de overheid komt het vaak voor dat een bericht via een tussenpartij van A naar B wordt gestuurd. Het is eindgebruikers lastig om precies aan te geven langs welke machines een bericht moet reizen om op de plaats van bestemming aan te komen.
Adressering/routering lost dit op door de eindgebruiker een identiteit op te laten geven(adresseren) waar het bericht heen moet en het naar de eerstvolgende machine in de keten te sturen. Deze kan op basis van de identiteit het bericht doorsturen naar een volgende machine(routeren)



Dit komt bijvoorbeeld voor bij het gemeentelijk gegevensknooppunt(GGK), gemeenten sturen facturen naar zorgverzekeraars maar willen niet alle machine locaties van alle zorgverzekeraars bijhouden. Ze sturen een bericht voor een zorgverzekeraar naar het GGK onder vermelding van de zorgverzekeraar. Het GGK houdt voor alle gemeenten eenmalig bij waar per verzekeraar berichten moeten worden afgeleverd en routeert het bericht naar de juiste machine door.

Bericht versleuteling
Wanneer berichten via een tussenpartij verstuurd worden is het soms niet wenselijk dat deze tussenpartij de berichten kan lezen. In dat geval wordt niet alleen de transportlaag versleuteld, maar ook de inhoud van het bericht zelf. De versleuteling werkt zo dat het bericht alleen door de uiteindelijke ontvanger ontsleuteld kan worden. De tussenpartij kan het dus niet lezen, alleen maar doorsturen op basis van niet versleutelde bericht headers.



Dit komt bijvoorbeeld voor in het justitie domein waar de meest gevoelige strafdossiers op deze manier verzonden worden om ieder risico op uitlekken van gegevens uit te sluiten.

Adverteren webservice
We hebben nu een heel aantal interactie patronen voor uitwisseling behandelt, vaak worden die niet op zichzelf gebruikt maar is er sprake van een combinatie. Het samenspel kan lastig te implementeren zijn. Daarom is het voor ontwikkelaars handig als ze geholpen worden bij een aansluiting door meta-data die beschrijft hoe een verbinding precies werkt. Met goede metadata kan een groot deel van de aansluiting geautomatiseerd worden.

Een voorbeeld hiervan is de CPA-creatie voorziening van logius. Deze helpt bij het opzetten van Digikoppeling ebMS verbindingen door op basis van informatie van de aanbieder en afnemer een Collaboration Protocoll Agreement(CPA) te generen. Door deze CPA in zijn software in te lezen kan een ontwikkelaar de aansluiting (gebruikmakend van complexe combinaties van interactiepatronen) automatisch configureren. Dit scheelt veel tijd.

Code generatie

Je kan nog een stap verder gaan dan de automatische configuratie. Het is ook mogelijk om op basis van technische documenten programmeercode te genereren die in een applicatie gebouwd worden. Je kan code genereren om een webservice correct aan te spreken vanuit een applicatie. Je kan ook op basis van een schema (zie validatie) een databasestructuur genereren. Een voorbeeld hiervan is Digipoort, deze laat bedrijven gegevens aanleveren aan de overheid. Digipoort publiceert hiervoor WSDLs(webservice description language) en XSDs(schemas). Op basis van de WSDL kan de ontwikkelaar van het bedrijf het aanroepen van digipoort in een applicatie inbouwen. Op basis van de XSDs kan hij een database structuur genereren waarin hij gegevens van zijn applicatie opslaat voordat hij ze verzend.