Het gebruik van de Schematron validatie-regels.

De schematron validatie regels voor IMOW en IMOP zijn opgebouwd met een bepaalde structuur en methodologie aan de basis. In deze tekst probeer ik uit te leggen waarom dat is, en hoe deze structuur in elkaar zit.

Door

Bert Verhees

Inhoud

Inleiding

Dit artikel hoort bij een github-repository die xml_schematron heet. Deze directory bevat validatie-regels ten behoeve van IMOW en IMOP data-sets.

DEEL I: De Anatomie van de validatieregels-repository

Beschrijving van de volledige github-repository plus enkele opmerkingen over te gebruiken software.

De directory en bestanden van de validatie-regels repository.

De repository bestaat uit een aantal directories en bestanden die op de PC terecht komen als men een “git clone” doet. Sommige directories worden in andere delen van dit document uitgelegd. Dit zijn de directories die per validatie-regel bestaan, en de test-set directory.

De versie-directories.

Als je “git clone” hebt gedaan krijg je 3 versie-directories en een paar zip-bestanden. De ZIP-bestanden bevatten dezelfde bestanden als de directories die hetzelfde heten, maar de volledige sets, terwijl de directories alleen de validatie-set bevatten die bij die versie hoort.

De huidige versie is in de directory met het hoogste versienummer. In die directory:
documentatie: daarin bevindt zich dit bestand, en een Excel-sheet die de validatie-matrix bevat.
src: de omgeving waarin de tpod-validatie directories zijn. Per validatie-regel een subdirectory.
Validaties: hierin de bestanden die samen een volledige test-set vormen, die worden uitgeleverd via de geonuvum website.

De src-directory

In de src-directory zitten enkele subdirectories die niet met Test… beginnen.

Dit zijn:
- saxon9.9.1.5: hierin zitten de saxon-libraries + de hulp-bestanden die van schematron xslt maken. Dit setje bestanden wordt gebruikt vanuit de directory Tests. Ik kom daar later op terug.
- schemas: hierin zitten de xsd-schema’s die worden gebruikt tijdens het ontwikkelen van de validatie-regels. In de test-bestanden zit een verwijzing naar deze schema’s.
- Een hele rits bestanden die met Test_..... beginnen. Per directory wordt een validatie-regel uitgewerkt. In DEEL II ga ik uitleggen hoe dit werkt.
- Tests: In deze directory zitten wat hulpscriptjes om tests vanuit de bash-command-prompt uit te voeren. Deze directory gebruikt de saxon-directory die ik eerder noemde. Er zitten twee bestanden in: test_nummer.sh, moet aangeroepen worden met een validatie-soort/nummer.

Bijvoorbeeld vanuit de tests-directory: ./test_nummer.sh TPOD0230 Ook is het mogelijk om alle tests uit te voeren met ./test_all.sh

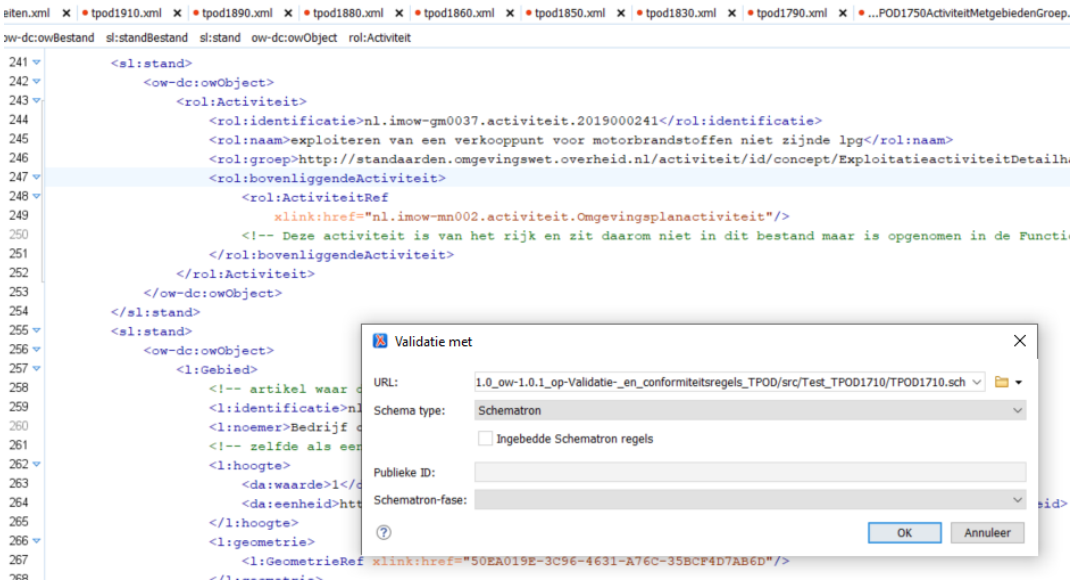
Benodigdheden en handigheden bij de ontwikkelomgeving

Bepaalde software producten zijn beslist nodig en sommigen zijn wel erg handig bij het ontwikkelen van validatie-regels. Ook zijn enkele producten nodigof handig bij het uitvoeren van de validatie-regels. Sommige tools zijn discutabel, en sommige tools hebben alleen mensen die software ontwikkelen op hun PC. Gelukkig zijn er ook altijd omwegen, maar niet altijd even efficient.

Oxygen

De Koningin onder de XML-editors. Oxygen (of <oXygen/> zoals ze het zelf noemen) verwerkt alle soorten XML en aanverwante subtalen en scriptalen. Oxygen kan moeiteloos omgaan met XML, XSLT, Schematron, en vele andere XML-gerelateerde toepassingen, daarnaast doet het ook JSON, Yaml, etc.

Dus in Oxygen kun je goed je validaties schrijven, het biedt vele hulpmiddelen, en je kan binnen deze omgeving een schematron uitvoeren met een xml-bestand als target.



In feite kun je in Oxygen alles doen wat noodzakelijk is, maar dan mis je wel mogelijkheden tot automatisering van de ontwikkelomgeving, en het opslaan in versies.

Dus je kan ermee aan de slag, maar het is niet genoeg om efficient te werken.

Ant

Ant (Engels voor mier) is een veel gebruikte Java-ontwikkeltool. In DEEL III is te lezen hoe een validatie-set eruit ziet. Dat is enigszins complex, maar niet onnodig complex. Lees daar verder over het waarom. Ant helpt om met de validatie-set te kunnen werken. Het kan ingewikkelde opdrachten uitvoeren. Ant is beschikbaar voor Windows en voor alle Unix-smaken.

Om met Ant te kunnen werken is een Java JDK noodzakelijk. Ant is een tool voor ontwikkelaars, en ontwikkelaars hebben altijd een JDK geïnstalleerd. (sommigen zonder het

te weten). JDK is de Java Development Kit. Deze bestaat uit de JRE (Java Runtime Environment) plus ontwikkeltools, compilers, etc. Een van de ontwikkeltoosl die Ant nodig heeft zit in tools.jar, en als er geen JDK is geïnstalleerd, dan gaat Ant daarover mopperen: “tools.jar not found”

De validatie-set kan ook zonder Ant worden gebruikt, maar dan moet je een aantasl dingen zelf doen die nu Ant voor je doet. Dit staat uitgelegd in DEEL V.

Git

Git is noodzakelijk om de gehele ontwikkelset veilig op te slaan en terug te halen. Het onthoudt ook oudere versies en biedt vele features voor versie-beheer.

Git-bash

Git-bash is een MINGW64 implementatie. Het is een Unix/Posix (ISO/IEC 9945)-client voor Windows. Werkt men vanuit Linux dan is dit niet nodig, Linux zelf is Posix compliant.

Men kan echter ook zonder deze. Sommige handige scriptjes die ik heb geschreven moeten dan omgezet worden naar batch bestanden. Ik heb hier nog niet in voorzien. Mogelijk red ik dat nog voordat mijn tijd bij Geonovum voorbij is.

DEEL II: De anatomie van een validatieregel

Dit deel beschrijft hoe een ontwikkel-omgeving van een validatie-regel eruit ziet.

Het begint met een directory. Zoals elders ook beschreven is het nuttig om deze directory met “Test_” te beginnen. Wat er achter “Test_” komt kan worden gebruikt om het op te roepen vanuit het test_nummer.sh in de Tests directory.

De naamgeving

Een validatie-regel bevindt zich in een directory. Deze heet bijvoorbeeld Test_TPOD1860. Dit is de validatie-regel die regel 1860 uitvoert Het schematron bestand heet dan TPOD1860.sch Dit heeft voordeel om dit in stand te houden.

Er bestaat naast de directories met de individuele test-ontwikkelsituaties ook een directory Tests. In DEEL IV is beschreven waarvoor deze nuttig is.

De test-bestanden

Beschrijving test-bestanden

De test-bestanden zijn databestanden zoals die voor kunnen komen in een dataset voor de omgevingswet. Het is geen complete dataset, maar het bevat alleen de delen die nodig zijn om de specifieke validatie-regel van die directory te testen.

Technisch gezien zijn de test-bestanden op orde, ook die delen die niet worden gebruikt zijn zoveel mogelijk technisch op orde en behorend bij de validatie-regel en IMOP/IMOW versies.

Doel test-bestanden

De test-bestanden helpen bij het ontwikkelen en van een validatie-regel. Ze zijn zo ingericht dat een validatie-regel een fout er in vindt, en ook dingen goed keurt, zodat zowel goed als fout wordt getest.

De validatie-regel wordt gedraaid tegen een test-bestand. Soms worden er data getest die verspreid over meerdere bestanden worden opgeroepen. Dan zorgt de test-routine ervoor dat die bestanden worden gevonden. Meestal is dat aan de hand van xml-tags.

Meerdere test-bestanden

Een validatie-regel-directory kan meerdere test-bestanden hebben. Vaak zit er een besluit-bestand bij. Soms heeft dit alleen de functie om het TPOD-type op te zoeken. Dit geldt voor validatie-regels die maar voor enkele TPOD's geldig zijn. Soms dienen ze omdat het verband tussen twee bestanden wordt onderzocht.

Hoe te testen

Tijdens het ontwikkelen is er eigenlijk een manier om te testen die het meest gangbaar is. Men pakt een test xml of gml bestand en opent dit in Oxygen, en via de menu-structuur gaat men naar Document -> Valideer -> Valideer met, en dan kiest men voor het schematron bestand in de directory.

Een andere manier is via het scriptje uit de Tests-directory. Men gaat via een Posix-prompt naar de directory Tests en daar roept men het scripje test_nummer.sh aan. Bijvoorbeeld: ./test_nummer.sh TPOD0230

Het schematron-bestand

Het schematron bestand kan op verschillende wijzen worden opgebouwd, ik heb een stramien opgebouwd die steeds terug komt. Hopelijk wordt in onderstaand verhaal duidelijk waarom dat is, en wat de voordelen zijn.

Hoe werkt schematron?

Richard (Rick) Alan Jelliffe bedacht schematron in 2001 om tests te kunnen uitvoeren op xml die totdan toe onmogelijk waren. Om dit te doen wordt de sch-namespace in een schematron bestand vertaald naar xslt, en die xslt wordt uitgevoerd om een xml bestand te testen. Voor dit doel worden xslt-bestanden gebruikt die de transformatie bewerkstelligen.

Er zijn dus twee zaken nodig om schematron te gebruiken, dat zijn de genoemde xslt's en libraries om xslt uit te voeren. In de validatie-set zijn twee directories. Een directory heeft xsl, hierin zijn de door Jeliffe bedachte xslt bestanden, en de directory lib, hierin zijn de saxon-xml bestanden, in Home-Edition-versie (HE), die is gratis en volstaat voor dit doel.

Belangrijk op dit moment om te onthouden is dat de schematron namespace wordt vertaald naar xslt. Dit is een proces met beperkte mogelijkheden. Vandaar dat er xslt-code wordt toegevoegd aan de schematron validatie.

De onderdelen van het schematron bestand.

Ik heb als voorbeeld Test_TPOD1730 genomen omdat dit alle onderdelen op redelijk eenvoudige wijze gebruikt.

De namespaces XSLT

Bovenin het sch bestand zitten de namespaces. Dit zal iedere XML-expert bekend voorkomen. Eerst worden de namespaces benoemd die door eht xslt-gedeelte van het bestand worden gebruikt. Omdat dit in ieder sch-bestand terugkomt heb ik alle mogelijke namespaces neergezet. In ieder sch-bestand zitten dus namespaces die niet worden gebruikt.

De namespaces SCH

Dan worden deze namespaces herhaald in de sch-namespace en in een andere syntax. Deze worden door de Jeliffe-transformatie gebruikt. Die transformatie laat de reguliere namespaces links liggen. Om ook hierin maar een keer te hoeven na te denken zijn alle mogelijke namespaces opgenomen.

Het xslt-gedeelte gebruikt dus normale namespaces en het sch gedeelte construeert namespaces uit de sch-namespace tags.

De transformatie van namespaces

Tijdens het transformeren vanuit sch naar xslt worden de sch-namespaces gebruikt. Als ze onvolledig zijn komt er een foutmelding. Omdat het sch-transformatie-proces en de xslt-uitvoering nadien beiden foutmeldingen kunnen veroorzaken is het soms moeilijk om erachter te komen waar het fout gaat.

Echter, wanneer men de structuur aanhoudt die ik gebruik, dan gaat het goed.

Het generieke deel

Het generieke deel voorziet in de declaratie van globale variabelen. In het totale validaties.sch uit de test-set wordt dit generieke deel maar een maal opgenomen, dit geldt ook voor de namespace-declaraties (xslt en sch)

De generieke variabelen zijn variabelen die het regelingtype declareren. Dan is er de variabele SOORT_REGELING die het regelingtype ophaalt uit het besluit.xml

Verder is er een variabele xmlDocuments die alle xml-bestanden in de data-set-directory bevat. Er is ook nog een gmlDocuments die de gml-bestanden bevat.

Het schematron gedeelte

Dit gedeelte is in iedere validatie-regel bijna hetzelfde. Meestal wordt er uitgeweken naar een xslt-functie onder het schematron-gedeelte. Heels soms is dat niet nodig. Het probleem met schematron is dat de transformatie naar xslt op niveau van eenvoudige vergelijkingen (bijvoorbeeld regexpr) goed verloopt, maar zo snel er algoritmische structuren (bijvoorbeeld loops) nodig zijn is deze transformatie onbetrouwbaar. Vandaar de functie die er bijna altijd bij staat.

Het schematron gedeelte begint met een pattern-id. Deze moet uniek zijn, daarom krijgt die de naam van de validatie-regel. Dan komt de rule, die geeft de xpath-context aan. Soms kan in de rule ook al een xpath-functie worden gedaan. Dit is bijvoorbeeld het geval in TPOD0460. Die validatieregel is zo eenvoudig dat de xpath-functie in de rule het hele werk doet. De rule, als die wordt ingezet, is altijd fout. Daarom wordt de CONDITION gewoon op false() gezet, zodat de assert altijd wordt getriggerd. Natuurlijk kan dit sneller door de

CONDITION variabele helemaal niet te gebruiken, maar ik heb voor alle regels geprobeerd de structuur zoveel mogelijk het zelfde te laten uitzien. Dit maakt het werken ermee inzichtelijker.

Hierna wordt de variabele APPLICABLE gedeclareerd. Deze bepaalt in welke TPOD deze validatie-regel geldig is. Als hij algemeen geldig is wordt hij gewoon op true() gezet. Dit is het geval in TPOD2150. Ook dit had eenvoudiger gekund door de APPLICABLE variable helemaal niet te gebruiken, maar om redenen die ik al eerder noemde. Ik heb de schematron structuur in alle regels zoveel mogelijk gelijkend opgezet.

Vaak volgt dan nu wat gedoe, er wordt een functie uitgevoerd, een variabele gevuld en uiteindelijk krijgt de variabele CONDITION een waarde.

Uit dit alles volgt een construct dat in iedere validatie-regel hetzelfde is.

```
test="($APPLICABLE and $CONDITION) or not($APPLICABLE)"
```

Als deze constructie false() oplevert dan wordt de assert uitgevoerd. Deze constructie levert false() op als APPLICABLE true() is en CONDITION false(). Dus de assert wordt alleen maar uitgevoerd wanneer de validatie-regel van toepassing is op de TPOD in het besluit.xml en de CONDITION fout gegaan is.

Daarna is de validatie-regel afgelopen. Wil men twee assrts toevoegen, dan kan dat wel, maar de tweede assert wordt alleen uitgevoerd wanneer de eerste assert niet getriggerd is. Dit is dus een rommelig gebeuren, waarbij een foutmelding incompleet kan zijn.

Als je kijkt naar TPOD2100 dan zie je dat wanneer je twee asserts wilt kunnen laten triggeren, je twee rules moet gebruiken

Het schematron gedeelte maakt gebruik van de namespaces in sch-namespace, zoals stukje terug uitgelegd.

Het xslt-gedeelte

Dit is altijd in de vorm van een functie. De functie heeft een naam die uniek is voor deze validatie-regel. Dit komt omdat de functies worden meegenomen naar validaties.sch en daar een unieke naam moeten hebben. De functies bevatten code die bij transformatie van schematron naar xslt riskant is of onmogelijk.

Verder is het erg verschillend hoe deze functies uitzien. Meestal is er een functie, soms zijn er twee, dan roept de ene functie de andere aan. Maar vanuit het schematron gedeelte wordt maar een functie aangeroepen. Deze functie levert altijd een kort eenduidig resultaat op die in de declaratie van CONDITION eenvoudig te evalueren is.

De functie maakt gebruik van de namespaces die als normale xml-namespaces zijn opgenomen.

DEEL III: De anatomie van de test-set

De test-set is dat wat normalerwijze ter download wordt aangeboden op de geonovum website. Het bevat alles wat nodig is om een data-set te valideren. Als men het zip-bestand uitpakt komen er een aantal bestanden en directories uit, die ik hieronder zal verklaren.

Belangrijk is dat er een directory Opdracht (met hoofdletter O) nodig is, in deze directory komen de te valdieren xml en gml bestanden.

De lib-directory

De lib-directory bevat alle libraries die nodig zijn om de validaties te laten draaien. Dit zijn de saxon (home-edition) bestanden. Deze bestanden zijn gratis te downloaden.

Verder zit in de lib-directory ant-contrib library. Dit zijn functionele toevoegingen op ant. Er zit de library in, en een paar directories. Deze directories heft ant nodig om deze ant-contrib te laten draaien. Dus gewoon niets aan doen is het advies. If it ain't broke, don't fix it.

De Opdracht-directory

In deze directory komt de te valideren data-bestanden. Ze moeten in één directory bij elkaar staan omdat het samenspel van ant en schematron ze bij elkaar zoekt, en er soms meerdere xml/gml bestanden bij een validatie-routine zijn betrokken.

De xsl directory

In deze directory staan de xslt-bestanden die saxon gebruikt om van schematron tot xslt te transformeren. Deze set werkt, en is best complex om te doorzien. Dus gewoon niet naar kijken verder.

De overige bestanden

Er zit een setje bestanden in deze directory. Dit zijn:

validaties.sch

Dit bestand bevat de verzameling van alle validatie-regels. Het bevat een aantal dingen eenmalig die de validatie-regels afzonderlijk allemaal hebben. Dit zijn de namespace xslt/sch declaraties en het generieke deel zoals uitgelegd in DEEL II. Daaronder, gescheiden door een lijn, de validatie-regels (schematron gedeelte en bijbehorende functies, ieder in een eigen blokje.) Kijk ernaar en het zal gelijk duidelijk zijn wat ik bedoel.

build.xml

Het build.xml is het ant-configuratie bestandje, erin zitten de opdrachten die ant uitvoert. Ant wordt opgestart vanuit **validate.s**h (posix) of **validate.bat** (windows). Het is belangrijk om dit te gebruiken omdat ant hieruit, samen met het build.xml de juiste dingen doet.

DEEL IV: Tips en instructies om validatie-regels te ontwikkelen.

Ik gebruik altijd een bestaande validatie-regel en kopieer deze in een nieuwe directory. Dan staan de namespaces en globale variabelen goed. De structuur van de onderdelen zit goed. Hierna ga ik bedenken wat de validatie-regel moet doen. Moet dit in een aparte functie, of kan het via xpath of een simpele vergelijking. Goed opletten dat de namen goed staan. Namen komen voor in de assert-melding, en in de pattern-declaratie.

De functie naam moet uniek zijn, ik heb dat altijd bereikt door de naam van de validatie-regel in de functienaam te verwerken. Zelfs als een functie meermalen voorkomt in verschillende validatie-regels, dan maak hem meermalen aan. Want anders wordt het geheel een bundel spaghetti verwijzingen van validatie-regels onder elkaar. En dat is het niet meer onderhoudbaar.

Is alles klaar en getest, dan kopieer ik de validatie-regel in het validaties.sch.

Bij onderhoud of bugs erop letten dat iedere validatie-regel dubbel voorkomt. Eenmaal als zelfstandige validatie regel, en eenmaal in het validaties.sch. Houdt het laatste synchroon met de afzonderlijke validatie-regels. Let hier ook op bij review.

DEEL V: Tips en instructies voor het testen van datasets

Er zijn twee manieren om de validatie-set te gebruiken. Ze hebben allebei voor en nadelen.

Via ant

Pak de validatie-set uit in een directory. Pak de te valideren data-set en zet deze in de directory Opdracht in de validatie-set. Vanuit de DOS/Posix prompt, ga in de directory van de validatie set staan en geef commando ./validate.sh (posix) of validate.bat (windows). De resultaten van de validatie verschijnen in de commmand-window.

Wil je deze naar een tekst bestand uitvoeren, dan kan dat zo in Posix (./validate.sh > log.txt 2>&1)

Het nadeel van deze werkwijze is dat ant en de java-jdk geïnstalleerd moeten zijn. Software-ontwikkelaars hebben dat meestal.

Via Oxygen

Via Oxygen of een andere goede XML-editor kan de validatie set ook gebruikt worden. Het is niet nodig om Ant te installeren. De wijze waarop dit gebeurt is als volgt.

Je zet te valideren data-set in een directory, en in diezelfde directory zet je de validaties.sch.Op deze wijze staan de directories goed en kunnen complexe validaties die meerdere bestanden betreffen vanuit de validatie-routines worden gevonden.

De nadelen van deze werkwijze zijn dat de validatie.sch in de directory van de data-set moet worden geplaatst, wat toch minder elegant is. Een ander nadeel is dat ieder XML-bestand apart moet worden gevalideerd.

Epiloog

Dat was het, hopelijk zijn alle vragen hiermee beantwoord, en hopelijk helpt het om te bedenken dat overal structuur en methodologie achter zit. Alles heeft zijn reden.