**MMBase webservices**
Auteur: Johannes Verelst & Nico Klasens
Date: July 3, 2009

**Aanleiding**
Afgelopen periode zijn er door een aantal partijen webservices gebouwd boven op MMBase: in verschillende technologieën en met verschillende soorten output. Het effect is dat iedereen geneigd is het wiel opnieuw uit te vinden, en dat er geen standaard webservices-laag bovenop MMBase bestaat.
Heel concreet heeft Kennisnet recent een webservice-laag gebouwd en deze aan de community gedoneerd. Op basis hiervan is een gesprek geweest tussen Finalist, Kennisnet, Stichting MMBase en EO over de mogelijkheid om dit generiek op te pakken.

**Document model**
Een terugkerend probleem bij het praten over webservices is dat er binnen MMBase geen definitie van een document bestaat. Er zijn wel definities van objecten en relaties, maar die vormen samen substructuren (documenten) die alleen impliciet bekend zijn: door een bepaalde manier van Editwizards bouwen of JSP Templates te schrijven.

**Architectuur voorstel**
We stellen voor om een oplossing te bouwen waarbij webservices niet specifiek voor een partij geschreven worden, maar waarbij ze kunnen worden gegenereerd op basis van meta-informatie over de structuren binnen MMBase. Dit betekent echter dat deze meta-informatie wel in een bepaald formaat moet worden opgezet die te vertalen is naar webservices.

**Standaarden**
Er zijn ontwikkelingen wat betreft standaardisatie voor webservices op ECM systemen (CMIS door OASIS), maar deze standaard lost een heel ander probleem op dan dat hier beschreven staat.
JSR-170/283 zijn ook standaarden voor Java Content Repositories, maar deze lossen ook niet het probleem van document definities op. De JCR is net zo'n abstracte laag met nodes en properties als MMBase.

**Globale architectuur van webservice gebaseerde applicaties**

In een uitgebreide webservice applicatie zijn er 3 verschillende lagen te onderscheiden.

1. Persistentie laag
2. Domein laag
3. Transport laag

De persistentie laag bevat de logica om data uit de persistente store (database, bestanden, enz) te halen en de applicatie data containers te vullen.
- JPA/Hibernate gebruikt een database-mapping configuratie en POJO's (Plain Old Java Objects) als data containers
- MMbase gebruikt een database-layer om MMObjetNodes te vullen

De domein laag beschrijft het model van de applicatie. In de meeste gevallen is het een uitgebreid model met veel types en relaties.
- JPA/Hibernate heeft veel java classes en collections als members.
- MMBase heeft builders (typedef) en relatie definities (reldef en typerel)

De transport laag transformeert het uitgebreide domein model naar een geoptimaliseerde structuur die efficiënt over de lijn gestuurd kan worden. De representatie van de structuur die over de lijn gaat kan in XML, JSON of een andere formaat zijn.

- Op de JPA/HIbernate, m.a.w. POJO stack, kunnen frameworks gebruikt worden om van de POJO structuur een XML of JSON representatie te genereren. Vaak wordt eerst de data van de domein POJO's gekopieerd naar DTO's (Data Transfer Object) en deze wordt gebruikt binnen de frameworks.
- Geen standaard en daar gaat juist dit document over

**Technische Implementatie**

Zoals hierboven beschreven heeft de transport laag 2 taken.

1. data verzamelen uit het domein model (document model)
2. data serialiseren naar een transport formaat (XML, JSON).

Voor beide taken moet een oplossing uitgewerkt worden voor MMBase. De keuze van een oplossing voor de ene taak heeft effect op de implementatie van de andere.

In het gesprek tussen de partijen zijn 2 oplossingen besproken die afgeleid zijn van ideeën/implementaties die al bestaan binnen de mmbase community.

*Editwizard-style*

De eerste aanpak is vergelijkbaar met de architectuur van de editwizards: in XML beschrijf je daar hoe een editor er uit ziet: welke tabs heb je, welke gerelateerde content toon ik en hoe precies. Maar een editwizard-definitie bevat geen details met betrekking tot het renderen van de wizards. Op een zelfde manier zouden we webservices willen beschrijven: door een high-level definitie van de uit te wisselen data en hoe deze gemapt moet worden naar MMBase objecten en relaties. Hierbij komt het document-model weer langs: door een goede beschrijving te maken van wat een "Artikel" is aan objecten en relaties binnen een bepaalde MMBase installatie kan een webservice die artikelen kan leveren gemakkelijk gegenereerd worden.

Voor de serialisatie kan niet een willekeurig framework worden gebruikt, want het document model levert niet een java type structuur op die geserialiseerd wordt. In het extreemste geval moet zelf de code geschreven worden van document model naar ieder transport formaat.

*annotation-style*

Een tweede aanpak is om een DTO structuur te implementeren met java types en de members te annoteren met mapping informatie. Het DTO model beschrijft het document-model dat samengesteld is uit de mmbase objecten en relaties. Een annotation processor kan op basis van de annotations de objecten vullen met mmbase data.

Zodra de DTO structuur gevuld is door de annotation processor kan elk framework, dat java objecten serialiseert, gebruikt worden voor het transport formaat.

Beide aanpakken hebben voor en nadelen. De editwizard aanpak behoudt de flexibiliteit van mmbase door een mapping in configuratie te definiëren in plaats van in de source code. De annotation aanpak heeft als voordeel dat gebruik kan

worden gemaakt van andere open source frameworks en niet alles zelf geïmplementeerd hoeft te worden.