



HoGent

Academiejaar 2012–2013

Geassocieerde faculteit Toegepaste Ingenieurswetenschappen

Valentin Vaerwyckweg 1 – 9000 Gent

Internet of Things: Integratie in Drupal

**Masterproef voorgedragen tot het behalen van het diploma van
Master in de industriële wetenschappen: informatica**

Kobe WRIGHT

Stef DE WAELE

Promotoren: dr. ir. Pieter SIMOENS

prof. dr. ir. Ingrid MOERMAN (UGent - iMinds)

dr. ir. Jeroen HOEBEKE (UGent - iMinds)

Begeleiders: ing. Jen ROSSEY (UGent - iMinds)

ir. Floris VAN DEN ABEELE (UGent - iMinds)



HoGent

Academiejaar 2012–2013

Geassocieerde faculteit Toegepaste Ingenieurswetenschappen

Valentin Vaerwyckweg 1 – 9000 Gent

Internet of Things: Integratie in Drupal

**Masterproef voorgedragen tot het behalen van het diploma van
Master in de industriële wetenschappen: informatica**

Kobe WRIGHT

Stef DE WAELE

Promotoren: dr. ir. Pieter SIMOENS

prof. dr. ir. Ingrid MOERMAN (UGent - iMinds)

dr. ir. Jeroen HOEBEKE (UGent - iMinds)

Begeleiders: ing. Jen ROSSEY (UGent - iMinds)

ir. Floris VAN DEN ABEELE (UGent - iMinds)

Voorwoord

Graag bedanken we onze begeleiders Jen Rossey en Floris Van Den Abeele. Zij hebben ons doorheen de masterproef meermaals het licht getoond wanneer we in lastige situaties verzeild raakten. Ook willen we onze interne promotor Pieter Simoens bedanken. We waarderen zijn hulp enorm bij het schrijven van deze scriptie en het opstellen van onze poster. Verder bedanken we onze externe promotoren, Jeroen Hoebeke en Ingrid Moerman. Het zijn zij die ons in de eerste plaats deze masterproef hebben aangeboden.

Deze masterproef werd mogelijk gemaakt door de onderzoeksgroep iMinds. De accommodatie en het gebruik van de apparatuur hebben een positieve invloed gehad op het resultaat. We hopen dan ook dat deze scriptie een uitgangspunt mag zijn voor verdere ontwikkelingen.

Tot slot willen we graag alle andere mensen bedanken die ook rechtstreeks of onrechtstreeks een bijdrage geleverd hebben tot het realiseren van deze masterproef.

Kobe Wright & Stef De Waele, mei 2013

Toelating tot bruikleen

“De auteur geeft de toelating deze scriptie voor consultatie beschikbaar te stellen en delen van de scriptie te kopiëren voor persoonlijk gebruik.

Elk ander gebruik valt onder de beperkingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze scriptie.”

Kobe Wright & Stef De Waele, mei 2013

Internet of Things: Integratie in Drupal

door

Kobe WRIGHT

Stef DE WAELE

Masterproef voorgedragen tot het behalen van het diploma van
Master in de industriële wetenschappen: informatica

Academiejaar 2012–2013

Promotoren: dr. ir. J. HOEBEKE, prof. dr. ir. I. MOERMAN, dr. ir. P. SIMOENS

Begeleiders: ing. J. ROSSEY, ir. F. VAN DEN ABEELE

Geassocieerde Faculteit Toegepaste Ingenieurswetenschappen

Hogeschool Gent

Vakgroep Informatica

Samenvatting

Het *Internet of Things* wordt steeds belangrijker. Bij dit concept staat het idee dat verscheidene soorten apparaten rechtstreeks of onrechtstreeks met het internet verbonden worden centraal. Het biedt de mogelijkheid deze apparaten van op afstand te observeren en te besturen op een gemakkelijke en dynamische manier. Deze apparaten zijn in werkelijkheid vaak beperkt in mate van energieverbruik en geheugen. Vaak is het netwerk waarmee ze verbonden zijn eveneens beperkt waardoor beperkte bandbreedte en onbetrouwbare verbindingen niet zeldzaam zijn.

Het doel van deze masterproef bestaat erin een Drupalmodule te ontwikkelen die het mogelijk maakt op een gebruiksvriendelijke en dynamische manier sensordata op te vragen en sensoren te configureren. Hierbij moet rekening gehouden worden met de beperkte mogelijkheden van zowel het netwerk als van de aangesloten apparaten. Ook moet de module ruimte laten voor uitbreidingen en moeten de recente mogelijkheden van Drupal 7 benut worden met een oog op de aangeboden functionaliteit van Drupal 8. Enige commentaar in code mag dan ook niet ontbreken.

Om de communicatie tussen de devices over het netwerk te waarborgen is HTTP te zwaar. Daarom wordt er gebruik gemaakt van het minder zware CoAP. In ruil voor de betrouwbaarheid en robuustheid van HTTP biedt CoAP een minimale overhead, een hoge mate van simpliciteit en features die zich niet beperken tot de observe-functionaliteit en blockwise transfer.

Als resultaat bieden we twee Drupalmodules aan. De CoAP-*library*- en de CoAP-sensormodule. De CoAP-*library* verzorgt alle communicatie tussen clients en servers op een netwerk door middel van native CoAP. Deze module heeft op zich weinig te bieden aan de doorsnee Drupalgebruiker en heeft enkel als doel een CoAP API aan te bieden aan Drupalontwikkelaars. Deze *stand-alone* module kan ook gebruikt worden in andere projecten waar een PHP-implementatie van CoAP voor Drupal vereist is. De CoAP-sensormodule maakt gebruik van de CoAP-*library* en biedt de gebruiksvriendelijke interface aan waarmee Drupalgebruikers sensoren kunnen beheren.

Abstract

The Internet of Things is increasing in importance every day. The Central idea in this concept is that multiple kinds of devices are connected to the internet in a direct or indirect way. It offers the possibility to observe or configure these devices in a user-friendly and dynamic way. These devices are more often than not constrained in both energy supply and memory. Most of the time the network on which they are located is also constrained. Which results in low bandwidth and unreliable connections.

The aim of this thesis is to develop a Drupal module which allows a user-friendly and dynamic way to collect sensor data and to configure sensors. It must be taken into account that the devices and the network which they are connected to, are constrained. The module also needs to leave room for expansions and needs to take advantage from the recent possibilities from Drupal 7 while not neglecting the functionalities of Drupal 8. Therefore comments in the code should not be absent.

HTTP is too heavy to make the communication between the devices possible. Because of this, CoAP is used instead, which is a lightweight protocol. CoAP exchanges the reliability and robustness from HTTP for a minimal overhead, a simplified design and features which are not limited to the observe functionality and blockwise transfer.

As a result we present two Drupal modules. The CoAP library- and the CoAP sensor module. The CoAP library takes care of all the communication between clients and servers on a network, using native CoAP. This module doesn't offer much to the average Drupal user and its only purpose is to offer a CoAP API to Drupal developers. This stand-alone module can also be used in other projects where an PHP implementation of CoAP is required. The CoAP sensor module uses the CoAP library to offer the user-friendly interface which Drupal users can use to maintain sensors.

Trefwoorden

Internet of Things, Drupal, CoAP, sensornetwerk

Inhoudsopgave

Lijst van gebruikte afkortingen	iv
1 Inleiding	1
1.1 Doel	2
1.2 Verloop masterproef	3
1.3 Structuur scriptie	4
2 Drupal	6
2.1 Wat is Drupal?	6
2.1.1 Basiswebsite	8
2.2 Waarom Drupal?	9
2.3 Werking van Drupal	10
2.3.1 Drupal bouwstenen	11
2.3.2 Database Abstraction Layer	15
2.3.3 Laadcyclus van de pagina	17
2.3.4 Fields	24
2.3.5 Modules	25
2.3.6 Aanmaken en verwijderen van contenttypes	26
2.4 JQuery in Drupal	28
2.4.1 Met een HyperText Markup Language (HTML)-formulier	28
2.4.2 Met AJAX in jQuery	29
2.4.3 Problemen	29
3 CoAP: embedded HTTP done right	30
3.1 Wat is CoAP?	30
3.1.1 Berichtformaat	31

3.2	Communicatiemogelijkheden	35
3.2.1	Betrouwbaarheid	35
3.2.2	Request/responsemodel	36
3.2.3	Blokken	37
3.2.4	Observe	39
3.3	Extra Features	39
3.3.1	Resource discovery	39
4	CoAP library module	43
4.1	Doel	43
4.1.1	Essentiële functies	44
4.1.2	Optionele functies	46
4.2	Implementatie	47
4.2.1	Proceduregericht	47
4.2.2	Objectgericht	47
4.2.3	Hooks voor notificaties	49
4.2.4	Opvangen van verloren berichten	50
5	CoAP-sensormodule	52
5.1	Functionaliteit	52
5.2	Architectuur	53
5.2.1	Requestproces	53
5.2.2	Belangrijke aspecten	56
5.3	Evolutie van de module	60
5.3.1	Temperatuurmodule met HTTP/CoAP-proxy	60
5.3.2	Temperatuurmodule met native CoAP	62
5.3.3	CoAP-sensormodule met externe CoAP-library	63
5.3.4	CoAP-sensormodule met contenttype CoAP-resource	64
5.3.5	CoAP-sensormodule met volledige REST-functionaliteit	65
5.3.6	CoAP-sensormodule met resource discovery	67
5.3.7	CoAP-sensormodule met meerdere contenttypes	68
5.4	Contenttype in de databank	69
5.5	Contenttypes in Drupal	73

5.5.1	CoAP-resource	74
5.5.2	CoAP-device	80
5.6	Implementatie van de CoAP-libraryhooks	83
5.6.1	hook_receive_notification()	83
5.6.2	hook_receive_error()	83
5.6.3	hook_stop_observers()	84
6	Uitbreidingen	85
6.1	Keuze van interval bij automatisch ophalen	85
6.2	Opvragen devices in subnetwerk met resource directory	85
6.3	Conditional observe	86
6.3.1	Keep-alive	87
6.4	Custom Entity	88
6.5	Interface Description	89
6.6	Configuratie op maat	89
6.7	View Modes	90
6.8	DNS	90
6.9	Help	90
6.10	Block Options	90
6.11	Anonieme gebruikers	91

Lijst van gebruikte afkortingen

ACK	Acknowledgement CoAP-message
AJAX	Asynchronous JavaScript and XML
ASCII	American Standard Code for Information Interchange
CMS	Content Management System
CON	Confirmable CoAP-message
CR	Carriage Return
HATEOAS	Hypermedia as the Engine of Application State
HTML	HyperText Markup Language
IoT	Internet of Things
IPv4	Internet Protocol versie 4
IPv6	Internet Protocol versie 6
LF	Line Feed
M2M	Machine-to-machine
nid	Node ID
NON	Non-confirmable CoAP-message
OSI	Open Systems Interconnection
PHP	PHP Hypertext Preprocessor
REST	Representational Stateless Transfer

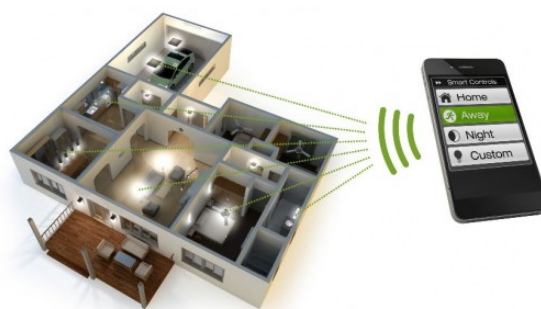
RST	Reset CoAP-message
sid	session-ID
TKL	<i>ToKen Length</i>
TLV	Type Length Value
uid	user-ID
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WOEID	Where On Earth ID

Hoofdstuk 1

Inleiding

Het internet zoals we het vandaag kennen is niet meer weg te denken en het aantal aangesloten machines blijft groeien. Tot voor kort bestond deze verzameling machines uit computers, routers, etc... en bijna alle data die het huidige internet omvat werd ingevoerd door menselijke tussenkomst. Maar daar komt verandering in. De aangesloten machines evolueren naar objecten of 'dingen', waarbij elk object uniek adresseerbaar is en zelf data kan inbrengen zonder menselijke tussenkomst. Deze objecten kunnen allerlei apparaten voorstellen, van een smartphone tot een wasmachine of zelfs een auto.

De komende jaren zal de omvang van het Internet exploderen omdat steeds meer van deze dagdagelijkse objecten ermee verbonden worden. Het Internet zal evolueren naar een *Internet of Things* (IoT) en zal ons toelaten om op een eenvoudige manier en van op afstand informatie te verkrijgen over apparaten en hun omgeving (bv. de temperatuur, een deurslot, de status van de wasmachine) en ermee te interageren (bv. sluiten van de deur, aanschakelen van de verwarming). Tot voor kort was het besturen van apparaten in je huis vanop afstand met een smartphone, computer, etc slechts een theorie, maar de realisatie hiervan is nu dichterbij dan ooit.



Figuur 1.1: *Internet of Things*

Is het internet klaar voor het IoT?

Een grote remming op de ontwikkeling van een IoT is het beperkte adresbereik van Internet Protocol versie 4 (IPv4), maar met de invoering van Internet Protocol versie 6 (IPv6) voor de deur, wordt het aantal mogelijke adressen aanzienlijk groter. Alhoewel IPv6 voldoet aan de eisen voor het IoT zal de invoering van IPv6 op grote schaal nog even op zich laten wachten. Daarom zal het IoT voorlopig beperkt blijven tot testomgevingen en netwerken op kleinere schaal.

Een tweede probleem wordt gevormd door het verkeer dat zo'n sensornetwerk veroorzaakt. Huidig gebruikte protocollen zoals HTTP zijn niet geoptimaliseerd voor verkeer in en uit een sensornetwerk. Er zijn wel nieuwe protocollen ontwikkelt voor deze taak maar die zijn nog niet ingeburgerd.

De conclusie is dat het Internet in het geheel en onder zijn huidige vorm nog niet klaar is voor het IoT, maar dat niets in de weg staat van de ontwikkeling ervan in de zeer nabije toekomst.

1.1 Doel

Omwille van deze evolutie worden er meer en meer embedded devices ontwikkeld voor het IoT. Onder embedded devices verstaan we devices waar resources op aangesloten zijn. Het device biedt deze resources aan, aan de buitenwereld. Deze masterproef behandelt voornamelijk de integratie van die embedded devices in bestaande systemen. De focus ligt dan niet zozeer op de ontwikkeling van de embedded devices zelf, maar op het aanspreken en besturen ervan. Meer specifiek willen we een bijdrage leveren aan de realisatie van webservices voor het IoT. Het vertrekpunt hiervoor zullen de recente CoAP-ontwikkelingen zijn van de iMinds-onderzoeksgroep (C++ CoAP framework, HTTP/CoAP-proxy, CoAP voor sensoren). Als embedded device gebruiken we sensoren aangeboden door iMinds die bereikbaar zijn onder de vorm van een sensornetwerk.

Deze webservice bieden we aan onder de vorm van een Drupalmodule waarbij een gebruiker een device rechtstreeks op locatie kan aanspreken door middel van native CoAP-communicatie zonder dat er een proxy aan te pas komt. Dit is een groot verschil met recente implementaties waarbij er gebruik gemaakt wordt van een HTTP/CoAP-proxy tussen de applicatie en het device. Concreet worden er twee modules ontwikkeld. Een CoAP-*library* die al het CoAP-verkeer stuurt en afhandelt en een CoAP-sensormodule waarin de gegevens afkomstig van de devices, verwerkt

en weergegeven worden. De modules moeten zo ontwikkeld worden dat ze op een eenvoudige manier te gebruiken zijn, zodat geen kennis van de technische aspecten vereist is.

1.2 Verloop masterproef

Na een uitgebreide literatuurstudie van Drupal en PHP Hypertext Preprocessor (PHP, de gebruikte programmeertaal) werd gepoogd een testmodule te maken. Het resultaat van deze eerste module bestond uit een webservice die dynamisch kon worden toegevoegd aan een Drupal website. Men kon deze webservice aanwenden om aan de hand van een Where On Earth ID (WOEID), het weerbericht op te vragen van een plaats naar keuze.

De volgende stap bestond uit de ontwikkeling van een eerste versie van de uiteindelijk te ontwikkelen module. Deze bood de gebruiker de mogelijkheid om de waarde van één resource op te vragen. Bovendien kon de gebruiker met een checkbox aangeven of de waarde periodiek moest worden opgehaald. De opgehaalde waarden werden getoond in een tabel. In deze fase werd er nog gebruik gemaakt van een HTTP/CoAP-proxy. Later wordt de proxy uitgesloten en wordt er enkel nog gebruik gemaakt van CoAP.

Na uitvoerig bestuderen van de CoAP-drafts en het ontmantelen van CoAP-berichten in Wireshark [42], werd geëxperimenteerd met CoAP-berichten (Wireshark is een network sniffer). Al vlug bleek dat de ontwikkeling van een eigen CoAP-*library* geschreven in PHP, de beste oplossing was. Aldus werd een aparte module ontwikkeld, een CoAP-*library* die alle CoAP-communicatie voor zich neemt. Deze module biedt de mogelijkheid tot opstellen, versturen en ontvangen van CoAP-berichten.

Met de CoAP-*library* voorhanden werd het nu mogelijk de CoAP-sensormodule aan te passen zodat die enkel nog gebruik maakt van CoAP-communicatie. Door deze laatste ontwikkeling werd ook de observe-methode van CoAP mogelijk, welke later toegelicht wordt in paragraaf 3.2.4.

Naast deze laatste ontwikkelingen die zich eerder toespitsen op de *backend*, werd ook vooruitgang geboekt op de *frontend*. Er werd ingespeeld op de mogelijkheden van Drupal zodat een gebruiker gemakkelijk een CoAP-resource kan toevoegen aan zijn/haar website.

Omdat een gebruiker soms niet weet welke resources allemaal aangesloten zijn op een embedded device, wordt resource discovery voorzien. Dit houdt in dat de gebruiker enkel het IPv6 adres moet opgeven van een embedded device, waarna een lijst zal worden gegenereerd en getoond die alle resources bevat, aangesloten op dat embedded device.

Dit laatste concept kan nog verder worden doorgedreven tot het concept van service discovery. Hierbij krijgt de gebruiker een overzicht van lijsten van resources van elk embedded device in een bepaald subnetwerk. Als alternatief zien wij een resource directory [11, 38]. Deze wordt geïmplementeerd op een specifieke machine waar alle embedded devices hun core op beschikbaar stellen. Een gebruiker kan deze directory aanroepen om toegang te krijgen tot alle well-known/cores. De integratie met een resource directory werd echter niet geïmplementeerd, maar wordt in deze scriptie wel behandeld als mogelijke uitbreiding (Zie paragraaf 6.2).

Als laatste stap rest ons nog het samenvoegen van de *frontend* en *backend*. Als resultaat bekommt men dan een gebruiksvriendelijke module die de gebruiker in staat stelt sensoren te bekijken en te beheren zonder daarvoor enige kennis van onderliggende technologieën nodig te hebben.

1.3 Structuur scriptie

Hoofdstuk twee handelt over Drupal. We geven de voor- en nadelen van Drupal en gaan dieper in op enkele van de belangrijkste concepten en termen. We bekijken eveneens beknopt de werking van Drupal. Als laatste onderdeel van dit hoofdstuk bespreken we een voorbeeld van een Drupal module waarin jQuery gebruikt wordt.

In hoofdstuk drie gaan we dieper in op de concepten en mogelijkheden van CoAP. We bekijken het berichtformaat van CoAP en geven een vergelijking met HTTP. De voor- en nadelen komen eveneens aan bod. We bespreken de verschillende soorten communicatie en gaan ook dieper in op de concepten die met resource discovery te maken hebben. Deze concepten staan centraal in het automatiseren van ophalen van sensorgegevens.

In het vierde hoofdstuk bespreken we de ontwikkelde CoAP-*library*. Het ontwikkelingsproces wordt hier uitvoerig besproken, waarbij struikelpunten worden aangegeven en de gemaakte keuzes worden verantwoord. We bekijken ook de aangeboden functionaliteiten.

In het vijfde hoofdstuk bespreken we de CoAP-sensormodule. Er wordt grondig ingegaan op de architectuur van de module. Tevens wordt de evolutie van de module behandeld, er zijn namelijk verschillende versies ontwikkeld in de loop van de masterproef. Het eindresultaat wordt uitvoerig besproken.

Het zesde en laatste hoofdstuk handelt over mogelijke uitbreidingen die wij nog graag hadden geïmplementeerd maar niet meer pasten in de tijd die voorzien was. We sommen deze op en lichtten toe welke implementatie wij voor ogen hadden.

Hoofdstuk 2

Drupal

2.1 Wat is Drupal?

We kunnen Drupal het best beschrijven aan de hand van Figuur 2.2. Drupal is een *Content Management System* (CMS), een *framework* voor webapplicaties en een *social publishing platform*. Maar Drupal is meer dan software alleen. Drupal staat voor een gemeenschap van ontwikkelaars en gebruikers met uiteenlopende doeleinden die elk hun eigen visie willen realiseren. [43]



Figuur 2.1: *Drupal logo*

Content Management System

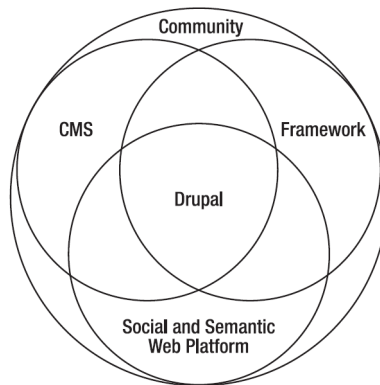
Drupal levert alle functies en mogelijkheden van een krachtig CMS. We denken meteen aan het kunnen inloggen en registreren van gebruikers, verschillende soorten gebruikers kunnen definiëren, verschillende niveaus van permissies,... Ook denken we aan het creëren, aanpassen, beheren, weergeven, categoriseren en aggregeren van content. Drupal biedt bovendien de mogelijkheid om modulair extra functionaliteit toe te voegen naar eigen noden en wensen.

Framework voor webapplicaties

Drupal is zeer flexibel en krachtig waardoor er enorm veel verschillende soorten webapplicaties mee kunnen gebouwd worden. Dit is deels te danken aan de API's die Drupal aanbiedt. Deze worden bij elke versie van Drupal uitgebreid maar ze worden niet complexer om te gebruiken.

Social publishing platform

Dat Drupal een *social publishing platform* is, houdt in dat content gemakkelijk te delen is via Drupal. Drupal biedt de mogelijkheid complexe data in een structuur te gieten die gemakkelijk uit te wisselen is. Op deze manier is het eenvoudig hetzelfde stuk content voor te stellen op verschillende websites.



Figuur 2.2: *Wat is Drupal?*

We kunnen Drupal ook beschrijven als een gratis softwarepakket dat je de mogelijkheid biedt om jouw inhoud gebruiksvriendelijk te beheren en te publiceren. En dit op zo een manier dat je een eindeloze graad van personalisering hebt. Deze inhoud kan bestaan uit allerlei dingen, zoals: een blog, een video, een foto, een artikel, resultaten van een experiment,... Algemeen is dit dus een combinatie van tekst, beelden en audio die bezoekers van je website kunnen zien, lezen en/of horen. Bovendien is Drupal *open source*.

Bij de implementatie van Drupal worden een aantal technologieën gebruikt. De eerste is de *HTML-embedded scripting*taal PHP wat voor PHP: Hypertext Preprocessor staat. Het wordt gebruikt om dynamische webpagina's te creëren. PHP wordt *server-side* uitgevoerd, de code wordt op een webserver uitgevoerd. Dit in tegenstelling tot *client-side* talen waar de code in een browser uitgevoerd wordt aan de clientzijde. De syntax van PHP bevat elementen van C, Java en Perl. Vanaf de huidige versie (PHP5) wordt object-geëoriënteerd programmeren ondersteund. Het is echter nog steeds mogelijk volledig proceduraal te werken.

Voor de *frontend* wordt er een noemenswaardige hoeveelheid JavaScript in de vorm van jQuery gebruikt.

Als laatste wordt voor het opslaan van content en configuratiegegevens van Drupal een relationele databank gebruikt. Welke specifieke technologie als achterliggende databank gebruikt wordt kan zelf gekozen worden. Door in code gebruik te maken van de Database Abstraction Layer heeft deze keuze geen invloed op de Drupal code. Standaard wordt het gebruik van deze laag in combinatie met MySQL, SQLite of PostgreSQL ondersteund. Indien een andere technologie gekozen wordt, is er extra configuratie nodig. De Database Abstraction Layer wordt verder besproken in 2.3.2.

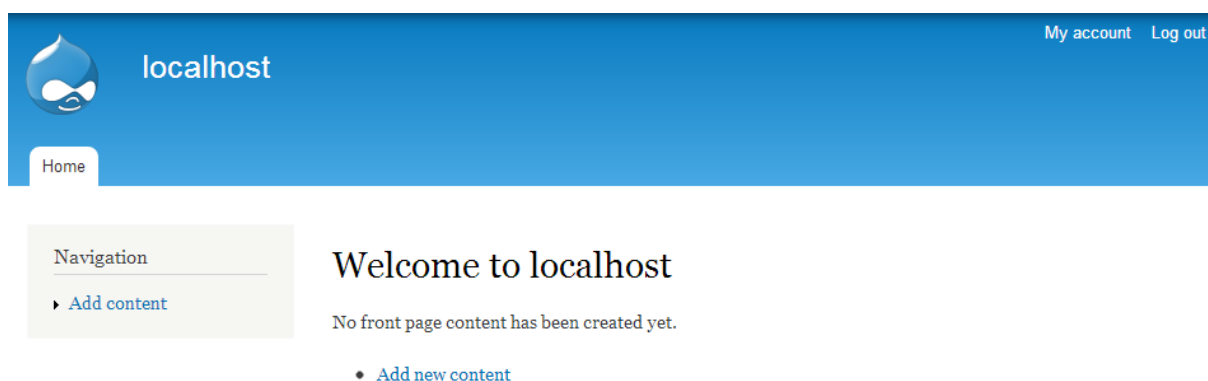
Drupal in zijn huidige versie (7) kan op elk platform draaien onder volgende twee voorwaarden:

- Het platform bevat een webserver die PHP, en dus *server-side scripting* ondersteunt. Voorbeelden van deze webserver zijn: Apache, IIS, Lighttpd en nginx.
- Het platform ondersteunt een van volgende databanktechnologieën: MySQL, SQLite of PostgreSQL.

Bij deze masterproef wordt er gebruik gemaakt van Apache en MySQL.

2.1.1 Basiswebsite

Wanneer je Drupal installeert beschik je meteen over een basiswebsite. Omdat je meteen al een bruikbare website hebt, is de drempel om Drupal te beginnen gebruiken dus laag. Deze website biedt meteen al een heleboel functionaliteit aan die geleverd wordt door de zogenaamde *Drupalcore*.



Figuur 2.3: Basiswebsite van Drupal met Bartik-theme

2.2 Waarom Drupal?

We bekijken de principes waarop Drupal gebouwd is [15]:

- **Modulair en uitbreidbaar:** Drupal kan uitgebreid worden met modules, waarbij je zelf ook modules kan ontwerpen indien er nog geen bestaat die aan jouw noden voldoet.
- **Kwaliteitsvolle codering:** kwaliteitsvolle, elegante en goed gedocumenteerde code is een prioriteit.
- **Standard-based:** Drupal maakt gebruik van ingeburgerde standaarden zoals bijvoorbeeld XHTML en CSS.
- **Low-resource demanding:** om een goede prestatie te garanderen, maakt Drupal gebruik van low-profile codering (bijvoorbeeld minimaliseren van databasequeries).
- **Open source:** Drupal is gebouwd op, en kan gebruikt worden in, andere *open source* projecten.
- **Gebruiksvriendelijk:** Drupal moet gemakkelijk te gebruiken zijn. Zowel voor gebruikers, ontwikkelaars en administrators van een website.
- **Samenwerking:** Drupal voorziet systemen om samenwerking te bevorderen, waaronder het versiebeheersysteem GIT.

Een bijkomend voordeel van Drupal is zijn grote gemeenschap die ondertussen uit al meer dan 630000 actieve gebruikers en ontwerpers bestaat die zich dagelijks inspannen om Drupal steeds beter te maken. Dit aantal neemt elke dag toe. Veel van deze ontwikkelaars werken in hun vrije tijd aan het Drupal concept. Er zijn echter ook een aantal bedrijven die bijdragen leveren. Een van deze bedrijven is Acquia [39], een bedrijf gesticht door Dries Buytaert, de geestelijke vader van Drupal. Acquia biedt een aantal diensten aan voor zowel gebruikers en ontwikkelaars:

- Er werd gepoogd de drempel om Drupal te gebruiken te verlagen door onder andere de installatie te beperken tot het uitvoeren van een file.
- Begeleiding onder de vorm van video's, extra documentatie, allerlei tutorials en een helpdesk die 24/7 bereikbaar is via mail of telefoon.
- Elastische resources om *spikes* in netwerkverkeer op te vangen.
- *Maintenance* van jouw Drupal site.

We bespreken nu enkele nadelen van Drupal:

- Aangezien Drupal gebruik maakt van een databank, heb je een databankserver nodig, al dan niet op dezelfde fysieke server als de webserver.
- Bovendien wordt telkens een pagina wordt opgevraagd, (een stuk van) de *bootstrap*code uitgevoerd, waarbij ook nog eens de databank veelvuldig wordt geraadpleegd. Dit maakt Drupal relatief traag.
- Drupal is zeer gebruiksvriendelijk en ideaal voor de eindgebruikers, die gemakkelijk en interactief inhoud willen toevoegen en beheren. Beginnende Drupal ontwikkelaars zullen evenwel merken dat Drupal een erg steile leercurve heeft.
- Ook ben je afhankelijk van de Drupal gemeenschap en dus de goodwill van de andere leden. Dit is in veel gevallen een voordeel, maar wanneer je een probleem hebt, ben je niet zeker of er wel een oplossing voor bestaat. Indien je gebruik maakt van software verkregen via Acquia heb je dit probleem niet.
- Tenslotte kan iedereen een module maken. Dit heeft natuurlijk zijn voordelen maar wanneer je een module van iemand anders gebruikt, ben je nooit zeker of de module zal onderhouden worden naar de toekomst toe en of er al dan niet bugs in zitten. Nogmaals, bij gebruik van software verkregen via Acquia is dit minder van toepassing.

2.3 Werking van Drupal

Aan de hand van een voorbeeld [43] proberen we de lezer een idee te geven van de werking van Drupal. Bekijk Drupal als een digitale postzegelsorteerder. Nodes in Drupal zijn te vergelijken met de postzegels en het concept van contenttypes in Drupal is vergelijkbaar met de verschillende soorten postzegels (postzegels van € 0,67, € 1,34, ...). Naast contenttypes, kan je *taxonomy* gebruiken om een verdere onderverdeling te maken in de postzegels. De mate waarin en de criteria waarop je je onderverdeling maakt, kan je zelf kiezen (land, kleur, ...). Je kan zelf ook extra criteria opgeven, zoals 'grekegen-van-oma'. Je kan *taxonomy* ook gebruiken om metadata toe te voegen aan content. Views is het mechanisme dat je postzegels sorteert en weergeeft in de vorm van pagina's en *Blocks*. Van deze pagina's en *Blocks* kan je zelf grootte, vorm, kleur en andere criteria opstellen zodat je deze helemaal kan personaliseren.

2.3.1 Drupal bouwstenen

We gaan dieper in op de begrippen die gebruikt werden in het postzegelsorteedervoorbeeld en lichten enkele extra concepten toe die eveneens belangrijk zijn in Drupal.

Contenttype

Een contenttype bepaalt het soort content. Het bundelt soorten gegevens tot een logisch geheel. Wanneer content van een bepaald type wordt aangemaakt, worden de gegevens ingevuld door middel van *Fields*. De toegevoegde content wordt een node genoemd. Ook biedt een contenttype de mogelijkheid om verschillende soorten content te onderscheiden van elkaar op basis van het type. Drupal biedt gebruikers de mogelijkheid aan om hun eigen *custom* contenttypes te maken. Hoe contenttypes aangemaakt en verwijderd worden wordt in paragraaf 2.3.6.

Node

Nodes zijn instanties van een contenttype. Een node kan maar aan één contenttype toebehoren. Alle nodes hebben enkele eigenschappen gemeenschappelijk:

- Node id: zorgt voor een unieke identificatie.
- Menu-instellingen: hier kan je een optionele menulink instellen zodanig dat je het menu op je website volledig kan personaliseren.
- Mogelijkheden in verband met revisie: in de levensloop van de content kan je revisies maken zodat je terug kan keren naar een vorig moment indien er iets fout gaat met de content.
- *Uniform Resource Locator* (URL) pad instellingen: standaard is content beschikbaar via een URL van de vorm `/node/node_id`. Maar via deze instellingen kan je een alias opgeven waarlangs de content ook (en dus gemakkelijker) beschikbaar is, dit principe heet in Drupal "Clean URL's". De URL zal dan van de vorm `/alias` zijn.
- Commentaarinstellingen: je kan zelf instellen of gebruikers commentaar kunnen achterlaten bij deze node.
- Auteurinstellingen: hier stel je in of de creatietijd en auteur getoond moeten worden bij deze node.

- Publicatieinstellingen: soms is het mogelijk dat je deze content nog niet beschikbaar wil stellen op de website, of je wil ze tijdelijk offline halen. Via deze instellingen is dit mogelijk.

Fields

Via *Fields* kan je gegevens van een contenttype invullen. Standaard biedt Drupal een aantal velden aan om gegevens toe te voegen, zoals een tekstveld of een veld om een afbeelding mee te uploaden. Maar soms schieten deze velden tekort, bijvoorbeeld wanneer je een kalender met drie drop-downlists wil combineren om een datum en tijdstip bij elkaar te laten horen in een veld. Daarom heeft een gebruiker de mogelijkheid *custom* velden te maken. Fields worden verder besproken in paragraaf 2.3.4.

View

Views worden gebruikt om content visueel voor te stellen op welke manier dan ook. Een *view* is dan ook niet meer dan een visuele representatie van een verzameling content uit de databank.

Block

Blocks zijn, zoals de naam impliceert, blokken die een verzameling van herbruikbare content bevatten. Ze geven het beeld van die content. *Blocks* kunnen op gemakkelijke wijze toegevoegd worden aan je website waar jij dat wilt en hoe vaak je dat wilt. Het is bijvoorbeeld gemakkelijk om aan te geven dat je een bepaald *Block* enkel op een bepaalde pagina wil laten verschijnen, en bovendien waar op de pagina je dat wilt. Merk echter wel op dat het hier niet echt content betreft, de inhoud van het *Block* wordt aangemaakt bij opvraging en is dus geen blok beheerbare content. We zien een voorbeeld van een *Block* in paragraaf 2.4.

Theme

Themes zijn templates die bepalen hoe jouw website eruit ziet en aanvoelt voor de gebruiker. Net zoals modules zijn *themes* modulair en zijn ze dus gemakkelijk te wisselen, ook *themes* kunnen zelf ontwikkeld worden en zijn ter beschikking op de Drupal gemeenschap.

Taxonomy

Taxonomy geeft je de mogelijkheid om eigenschappen en categorieën toe te voegen aan je content, zodat bijvoorbeeld een gebruiker content kan filteren uit een grote verzameling. Het biedt dus een

manier om je content te organiseren. Een praktijkvoorbeeld is een receptensite, een gebruiker kan dan recepten filteren aan de hand van de eigenschappen van het gerecht (ingrediënten, moeilijkheid, ...).

Gebruikers, rollen en permissies

Alle gebruikerfunctionaliteit zoals registreren, inloggen, enz... zit reeds in de Drupal *core*. Rollen geven weer tot welk type een gebruiker hoort. Meerdere gebruikers kunnen dezelfde rol hebben, en een gebruiker kan meerdere rollen hebben. Met permissies kan je aangeven welke privileges een gebruiker met een bepaalde rol heeft en dus wat die gebruiker wel en niet mag doen. Deze permissies worden gekoppeld aan een rol zodat alle gebruikers die deze rol hebben automatisch de privileges hebben van deze rol. Drupal biedt standaard drie rollen aan: de Administratorrol, die alle privileges bevat; de geauthenticeerde gebruikerrol, die een aantal van de privileges van de Administratorrol bevat en de anonieme gebruikerrol, die een subset van de privileges van de geauthenticeerde gebruikerrol bevat. Anonieme gebruikers zijn gebruikers die zich niet aangemeld hebben op de website.

Module

Modules bieden de mogelijkheid om extra functionaliteit in te pluggen op een website. Modules zijn gratis te downloaden van de Drupal gemeenschap en omdat deze gemeenschap groot is, is de kans zeer groot dat er al een module gemaakt is voor jouw probleem. Indien dit niet het geval is, heb je nog altijd de mogelijkheid om zelf een module te ontwikkelen. Het gebruik van modules voorkomt ook dat functionaliteit die je niet nodig hebt, ook niet op jouw website komt. Je website is dus zeer configureerbaar naar eigen wensen (Zie figuur 2.4).

Drupal core

De Drupal *core* is wat je downloadt van de Drupal website. Het vormt de basis en een uitgebreide *out-of-the-box* functionaliteit, het fungeert eigenlijk als de motor achter een Drupal website. De Drupal *core* is zeer uitgebreid en complex, het vormt op zich al voldoende materiaal voor een scriptie, er verder op ingaan zou ons dus te ver leiden.

[+ Install new module](#)

CORE				
ENABLED	NAME	VERSION	DESCRIPTION	OPERATIONS
<input type="checkbox"/>	Aggregator	7.21	Aggregates syndicated content (RSS, RDF, and Atom feeds).	
<input checked="" type="checkbox"/>	Block	7.21	Controls the visual building blocks a page is constructed with. Blocks are boxes of content rendered into an area, or region, of a web page. <small>Required by: Dashboard (enabled)</small>	Help Permissions Configure
<input type="checkbox"/>	Blog	7.21	Enables multi-user blogs.	
<input type="checkbox"/>	Book	7.21	Allows users to create and organize related content in an outline.	
<input checked="" type="checkbox"/>	Color	7.21	Allows administrators to change the color scheme of compatible themes. <small>Required by: Stylizer (disabled)</small>	Help
<input checked="" type="checkbox"/>	Comment	7.21	Allows users to comment on and discuss published content. <small>Requires: Text (enabled), Field (enabled), Field SQL storage (enabled) Required by: Forum (disabled), Tracker (disabled)</small>	Help Permissions Configure
<input type="checkbox"/>	Contact	7.21	Enables the use of both personal and site-wide contact forms.	
<input type="checkbox"/>	Content translation	7.21	Allows content to be translated into different languages. <small>Requires: Locale (disabled)</small>	
<input checked="" type="checkbox"/>	Contextual links	7.21	Provides contextual links to perform actions related to elements on a page.	Help Permissions

Figuur 2.4: *Organisatie van Drupal modules*

Entities

Een nieuw belangrijk concept in Drupal 7 is *entities* [17]. Dit nieuwe concept in combinatie met de *Entity API* heeft twee positieve effecten. Het eerste effect is dat gebruikers en commentaren dezelfde mogelijkheden krijgen als nodes. In vorige versies van Drupal was het niet mogelijk versies te creëren of velden toe te voegen aan gebruikers of commentaren. Andere voordelen zoals het gebruiken van *views* in combinatie met gebruikers of commentaren was ook niet mogelijk. Het andere effect is het invoeren van object-geïntendeerd programmeren van *entities*. Vroeger was het nodig specifieke functies van de *core* te gebruiken om content te bewerken. Een voorbeeld hiervan is `node_save()` dat nodes opslaat. Deze functie was enkel toepasbaar op een node. Het alternatief bij de *Entity API* is `entity_save()`. Deze functie is niet gelimiteerd tot enkel content of enkel tot gebruikers. Het is ook mogelijk nieuwe *entities* te maken.

Hooks

Hooks zijn functies die gedefinieerd zijn door de Drupal *core*. Ze kunnen worden geïmplementeerd door elke module. In de Drupal *bootstrap* zal de Drupal *core* dan op bepaalde tijdstippen de bijhorende *hooks* oproepen van elke module die de *hook* geïmplementeerd heeft. Hiervoor wordt een zeer eenvoudig mechanisme gehanteerd. Een module kan zo'n *hook* implementeren door de naam van de *hook* te laten voorafgaan door de naam van de module die de *hook* implementeert (zie codevoorbeeld 2.1).

Codevoorbeeld 2.1: Implementatie van `hook_node_view` door de module `coap_sensor`

```
function coap_sensor_node_view($node, $view_mode, $langcode) {
  if($node->type == 'coap_resource'){
    _coap_resource_add_js();
  }
  else if($node->type == 'coap_device'){
    $node->content['coap_device_form'] = drupal_get_form('coap_device_form', $node);
  }
  return $node;
}
```

2.3.2 Database Abstraction Layer

Manipulatie van gegevens in de databank gebeurt via de *Database Abstraction Layer* [3]. Dit zorgt ervoor dat de implementatie van een module onafhankelijk is van de gebruikte soort databank. Concreet biedt deze laag een aantal functies aan voor het manipuleren van de databank. Wanneer een nieuw soort databank in gebruik genomen wordt, worden deze functies geïmplementeerd voor deze nieuwe soort. We geven enkele voorbeelden van de meest gebruikte functies:

Codevoorbeeld 2.2: Voorbeeld gebruik van `db_select`

```
$result = db_select('coap_sensor_interested_user', 'resource')
  ->fields('resource', array('nid'))
  ->condition('uri', $uri, '=')
  ->condition('uid', $user->uid, '=')
  ->execute();
```

In voorbeeld 2.2 wordt opgevraagd wat het `nid` is voor een specifieke `uri` en `uid`. Er wordt gebruik gemaakt van de Drupalvariabele `$user`. Zoals je ziet moet er een naam opgegeven worden na de tabelnaam. Deze naam moet herhaalt worden als eerste element van de fieldstabel. Merk op dat dit alleen nodig is bij het gebruik van `db_select`. Als tweede element van de fieldstabel geef je een tabel op met alle velden die je wil opvragen. Meerdere condities kunnen opgegeven worden, de volgorde is niet belangrijk. De opgevraagde gegevens kunnen gesorteerd worden door gebruik te maken van `->orderBy('nid', 'ASC')`.

De variabele `$result` zal een *SelectQuery*-object bevatten. Er zijn twee manieren om de opgehaalde gegevens uit objecten van deze klasse te halen. Wanneer je weet dat er maar een enkele rij opgehaald wordt, doe je best het volgende

```
$record = $result->fetchAssoc();
$nid = $record['nid'];
```

Wanneer er meerdere rijen teruggegeven kunnen worden gebruik je beter

```
$nids = array();
foreach($result as $record){
    $nids[] = $record->nid;
}
```

om een tabel met al je gewenste resultaten in te bekomen.

Codevoorbeeld 2.3: Voorbeeld gebruik van db_insert

```
$id = db_insert('coap_sensor_interested_user')
    ->fields(array(
        'uid' => $user->uid,
        'uri' => $resource_uri,
        'device' => 0,
        'nid' => $nid,
        'observe' => 0,
    ))
    ->execute();
```

In voorbeeld 2.3 wordt een entry toegevoegd aan de tabel `coap_sensor_interested_user`. Kolommen die niet null mogen zijn en geen *default*waarde hebben moeten opgegeven worden in de fieldstabel. Zoniet wordt er een exceptie opgegooid bij het oproepen van de execute-functie.

Codevoorbeeld 2.4: Voorbeeld gebruik van db_update

```
$num_updated = db_update('coap_sensor_interested_user')
    ->fields(array(
        'new' => 0,
    ))
    ->condition('uid', $user->uid, '=')
```

```
->condition('device',1,'=')
->condition('nid', $nid, '=')
->execute();
```

In voorbeeld 2.4 wordt van alle rijen die overeenstemmen met de opgegeven condities de new-waarde op 0 gezet.

Codevoorbeeld 2.5: Voorbeeld gebruik van db_delete

```
db_delete('coap_sensor_interested_user')
->condition('nid', $nid, '=')
->execute();
```

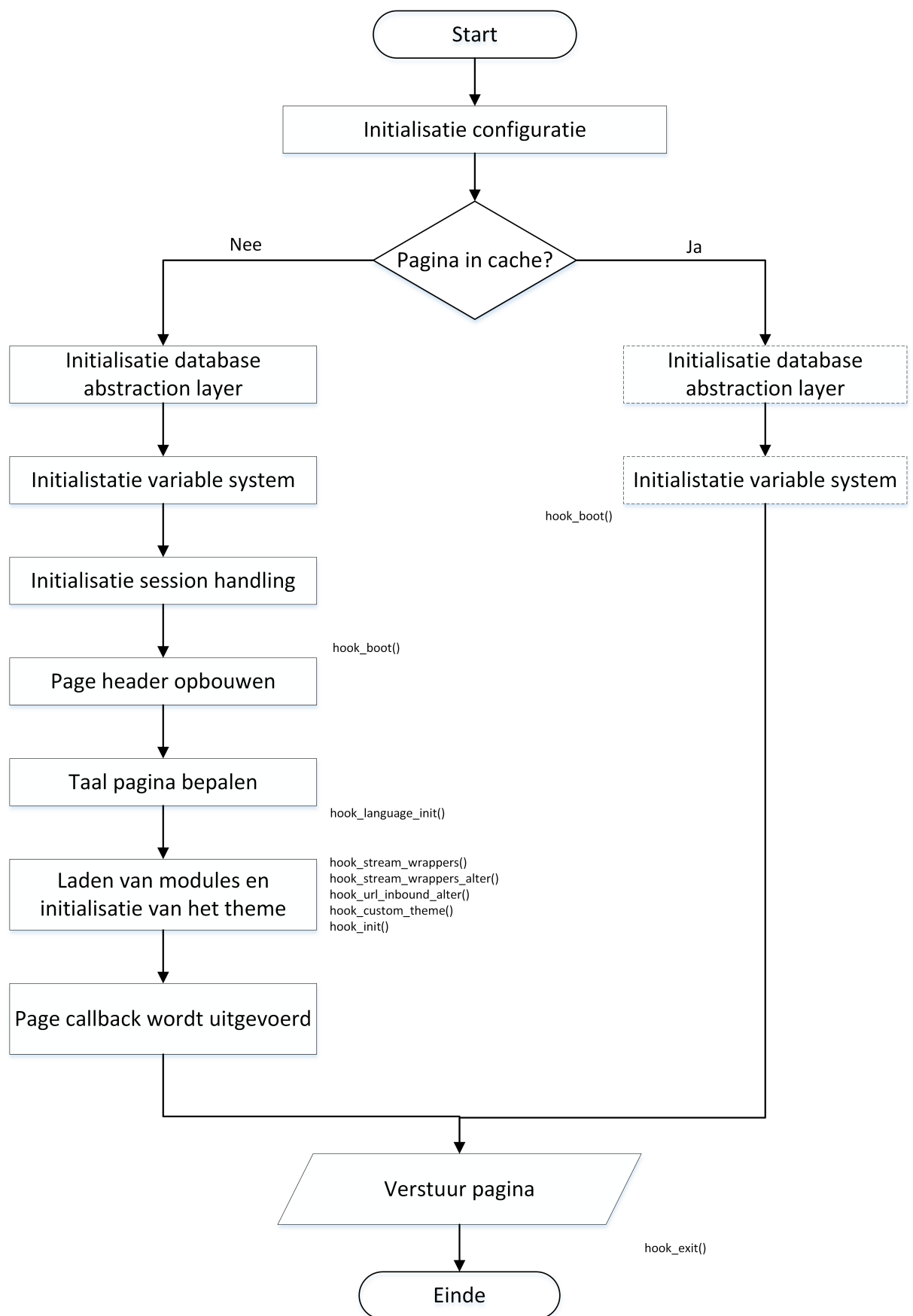
In voorbeeld 2.5 worden alle rijen met als nid de waarde in \$nid verwijderd.

2.3.3 Laadcyclus van de pagina

Wanneer een webbrowser een Drupal pagina opvraagt, wordt de Drupal URL van de pagina gebruikt. Deze heeft vaak volgende vorm: `http://drupalvoorbeeld.org/node/15`. De webserver vormt deze URL om naar de vorm: `index.php?q=node/15` en geeft dan het pad `node/15` door aan Drupal's `index.php` waarvan de code uitgevoerd wordt. Het uitvoeren van deze code resulteert in een reeks complexe stappen die als eindresultaat de gerenderde pagina hebben. Afhankelijk van het feit of de pagina al dan niet in de cache aanwezig is, verlopen deze stappen anders. Indien de pagina niet aanwezig is in de cache, wordt de volledige *bootstrap* en de *page callback* uitgevoerd. De *bootstrap* bestaat steeds uit dezelfde reeks van acht verschillende fasen die we verder nader toelichten. Na de *bootstrap* wordt de *page callback* uitgevoerd die geassocieerd wordt met het pad meegegeven met `index.php`. Indien de pagina wel in de cache aanwezig is, wordt er enkel een aantal van de stappen van de *bootstrap* uitgevoerd. De *page callback* wordt in dit geval niet uitgevoerd. We bespreken elke stap van het proces apart, het hele proces wordt weergegeven in Figuur 2.5.

Initialisatie van de configuratie

Deze fase houdt onder andere in dat globale variabelen worden gezet, zoals de basis-URL van de website. Sommige waarden van deze variabelen worden rechtstreeks uit het configuratiebestand `settings.php` gehaald. Andere worden berekend aan hand van de omgeving waarin de server zich

**Figuur 2.5:** Laadcycclus van de pagina in Drupal

bevindt. In settings.php worden variabelen opgegeven via de variabele `$conf`. Deze variabele is een associatieve array die alle waarden op de variabelenaam mapt.

Poging tot opvragen van een gecachte pagina

Caching van pagina's treedt op indien page caching geëabled is. Bovendien gebeurt dit standaard enkel voor anonieme gebruikers. Er zijn echter modules beschikbaar om dit ook voor geauthenticeerde gebruikers te laten gebeuren. Het doel van caching is vermijden dat de volledige *bootstrap* en *page callback* moeten worden doorlopen om zo tijdswinst te creëren. De pagina wordt enkel uit de cache gehaald als ze eerder is opgevraagd en nog steeds geldig is. De cache backend is pluggable en standaard gebruikt Drupal een databank cache implementatie. Deze cache backend heeft dus een databankconnectie nodig om de pagina op te halen. Daarom worden de volgende twee fasen (initialisatie van de Database Abstraction Layer en initialisatie van het Variable System) uitgevoerd voor de pagina effectief uit de cache kan gehaald worden. Dit wordt aangegeven door de stippellijnkaders in Figuur 2.5. De cache backend en of caching al da niet geëabled is, wordt opgegeven in settings.php.

Initialisatie van de Database Abstraction Layer

Er zijn nog geen verbindingen met de databank nodig. Daarom worden enkel basisklassen en functies geïnitieerd. Gebruik van de Database Abstraction Layer werd beproven in paragraaf 2.3.2.

Initialisatie van het Variable System

In deze fase wordt gebruik gemaakt van de *variable* tabel. Deze tabel heeft maar twee kolommen, *name* en *value* en beschrijft, zoals de naam zegt, variabelen. Alle name-valueparen worden uit de *variable* tabel gehaald en toegevoegd aan de variabelen gedefinieerd in settings.php. Variabelen die al gedefinieerd zijn in settings.php hebben een hogere prioriteit dan degene die uit de *variable* tabel gehaald worden. Het nut van deze variabelen is configuratie toevoegen aan de Drupal website. De reden dat een tabel gebruikt wordt om deze variabelen op te slaan is om de waarden persistent te maken. Wanneer de site offline gaat en terug online komt is de configuratie nog steeds beschikbaar. Drupal biedt ontwikkelaars een aantal functies aan om deze variabelen te manipuleren vanuit code:

- `variable_get($naam, $default = NULL)`: De variabele wordt uit `$conf` gehaald. Indien deze niet gedefinieerd is wordt de waarde in `$default` teruggegeven.
- `variable_set($naam, $value)`: De variabele wordt opgeslaan in *variabletabel* en gezet in `$conf`.
- `variable_del($naam)`: De variabele wordt verwijderd uit de *variabletabel* en uit `$conf`.

Naast alle variabelen worden de modules die nodig zijn in de *bootstrap* geladen. Onder deze modules verstaan we modules die minstens een van volgende *hooks* implementeerd: `hook_boot()`, `hook_exit()`, `hook_language_init()`, of `hook_watchdog()`. Als laatste onderdeel van deze fase wordt het *locking*mechanisme geïnitieerd. Dit is nodig bij langdurige processen die parallel naast elkaar kunnen maar niet mogen uitgevoerd worden, we gaan hier niet verder op in.

Initialisatie Session Handling

In deze fase wordt aan elke geauthenticeerde gebruiker een sessie gekoppeld. Een anonieme gebruiker krijgt geen sessie toegewezen tenzij er iets in de sessie moet worden opgeslagen. In dat geval wordt een nieuw sessie ID gegenereerd en wordt een nieuw *Userobject* gecreëerd met user id 0. Een andere manier dan de standaard databank gebaseerde manier om sessies af te handelen kan opgegeven worden in `settings.php`.

Page Header opbouwen

De eerste HTTP-headers worden opgebouwd. Deze worden echter nog niet verstuurd, dit gebeurt pas op het einde van de cyclus. In deze fase vormt zich de eerste mogelijkheid voor modules om functionaliteit in te pluggen in de cyclus van het laden van de pagina. Dit gebeurt via `hook_boot()`. Deze *hook* wordt uitgevoerd voor het aanmaken van de HTTP-headers.

De taal van de pagina bepalen

Als de website meerdere talen ondersteunt wordt in deze fase de gekozen taal van de gebruiker bepaald. Eens de gekozen taal vastgesteld is kunnen implementaties van `hook_language_init()` gebruikt worden om taalafhankelijke variabelen in te stellen.

Laden van modules en initialisatie van het theme

Alle geënablede modules worden geladen en het *theme* wordt geïnitieerd. Modules hebben nog de kans nieuwe *stream wrappers* toe te voegen en bestaande te wijzigen via de *hooks* `hook_stream_wrappers()` en `hook_stream_wrappers_alter()`. Met de hook `hook_url_inbound_alter()` kunnen er wijzigingen uitgevoerd worden aan de URL die gebruikt werd om deze pagina op te roepen. Bij het instellen van het thema wordt gekeken wat het actieve thema is. Dit kan het default thema zijn, ingesteld met de UI, een thema gekozen door de gebruiker ingesteld met `hook_custom_theme()` of een thema specifiek gezet voor dit pad. Aan het einde van deze fase wordt `hook_init()` opgeroepen. Dit wordt meestal gebruikt om globale parameters in te stellen die later in de request gebruikt worden.

Uitvoeren van page callback

Na het uitvoeren van de *bootstrap* moet de pagina opgebouwd en gerenderd worden. Deze fase leggen we uit aan de hand van een voorbeeld.

Voorbeeld

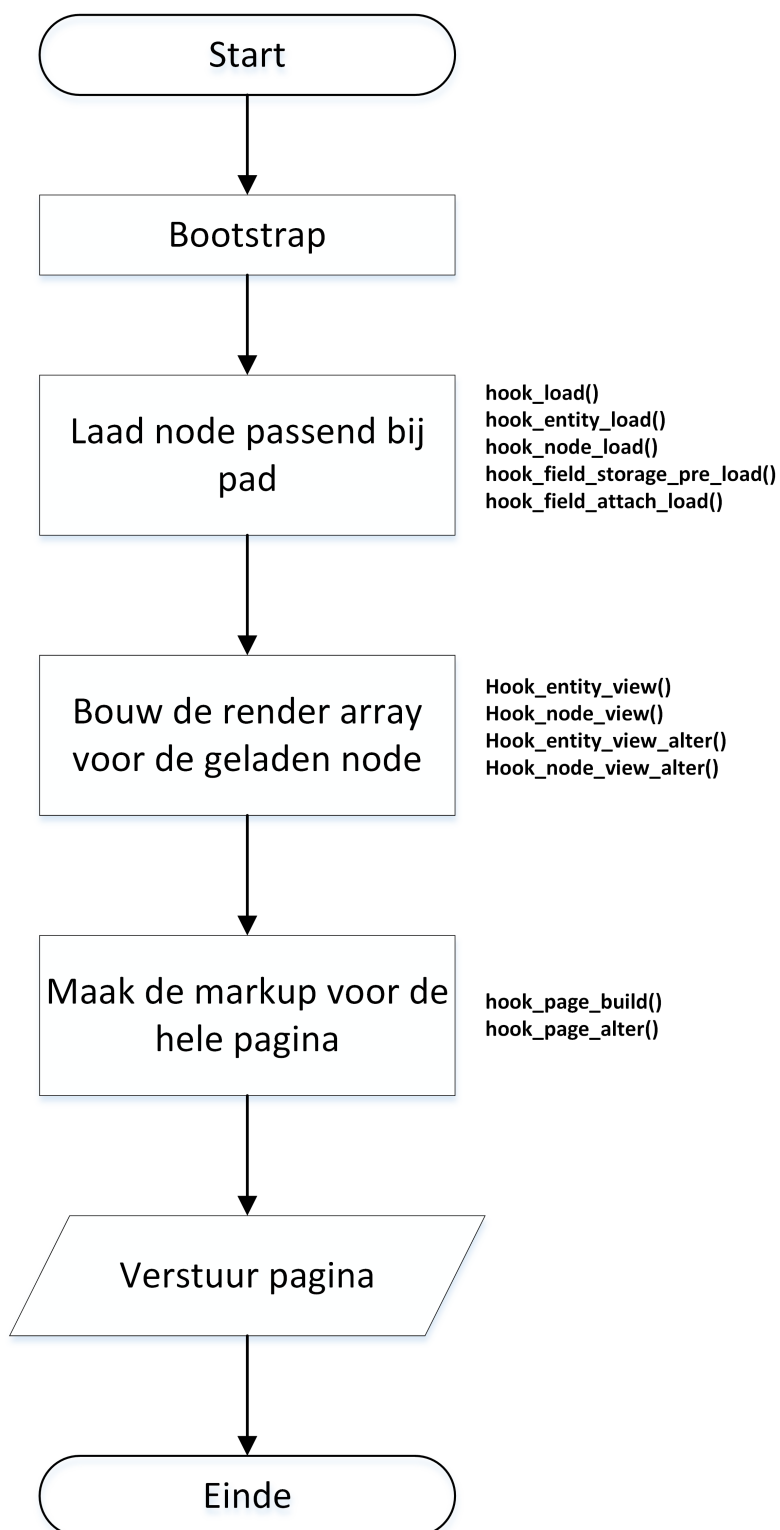
We gaan uit van de specifieke URL <http://drupalvoorbeeld.org/node/15> die we in het begin van deze paragraaf gebruikten. Op Figuur 2.6 zijn de hooks waarmee modules tussen beide kunnen komen worden aangegeven naast de respectievelijke fasen.

Drupal herkent het woord *node* in het pad en laadt de node met Node id = 15 via de functie `node_load(15)`. Gegevens van het node-object worden uit de databank gehaald en de hooks `hook_load()`, `hook_entity_load()` en `hook_node_load()` worden opgeroepen zodat modules de kans krijgen het node-object te veranderen of extra acties te ondernemen. Tijdens het laden van de node worden velden geassocieerd met de node eveneens opgehaald. Dit gebeurt met `field_attach_load()`. Modules krijgen de kans de opgehaalde data van de velden te manipuleren met de hooks `hook_field_storage_pre_load()` en `hook_field_attach_load()`.

Nadat de node geladen is wordt hij doorgegeven aan de *page callback*. Hier wordt hij omgevormd tot een *renderable array*. Dit gebeurt met de functie `node_page_view()`. Het zet als paginatitel de titel van de node. Het voegt ook een canonical en short link toe aan de HTML head elementen en de HTTP headers. In deze functie wordt de functie `node_show()` gebruikt

waarin de functie `node_view_multiple()` opgeroepen wordt die op zijn beurt `node_view()` oproept waarin uiteindelijk de *renderable array* gemaakt wordt. Via de hooks `hook_node_view()` en `hook_entity_view()` kunnen modules de *renderable array* nog manipuleren. Met de hooks `hook_node_view_alter()` en `hook_entity_view_alter()` krijgen modules nog een kans om de array te manipuleren nadat andere modules al veranderingen doorgevoerd hebben.

Wanneer de *renderable array* gemaakt is, en teruggegeven wordt door de *page callback*, wordt hij doorgestuurd naar `drupal_deliver_page()` waarin `drupal_deliver_html_page()` opgeroepen wordt die op zijn beurt de array rendert met `drupal_render()`. Voor `drupal_render()` effectief opgeroepen wordt, kunnen modules nog wijzigingen doorvoeren met de hook `hook_page_build()`. Modules kunnen nog een laatste keer tussenbeide komen met de hook `hook_page_alter()`.



Figuur 2.6: Voorbeeld van de laadcyclus van een pagina waar een node weergegeven wordt

2.3.4 Fields

Velden worden op een bepaalde manier behandeld in Drupal. We kunnen velden vergelijken met klassen. Wanneer een veld gebruikt wordt in een contenttype, wordt een instantie van deze klasse gemaakt. De gegevens van velden en hun instanties worden in verschillende tabellen in de databank opgeslaan.

Fields in de databank

De verschillende tabellen worden weergegeven in Figuur 2.7. We geven een korte beschrijving van deze tabellen:

- `field_config`: Bevat algemene informatie over een veld. Een veld staat hier maar een keer in beschreven omdat deze informatie onafhankelijk is van de module waarin het gebruikt wordt.
- `field_config_instance`: Bevat rijen per instantie van een veld. Een veld komt hier evenveel in voor als het aantal modules waarin het gebruikt wordt, rekening houdend met het aantal keer het voorkomt in een module. De informatie beschrijft dus de relatie tussen het veld en de modules waarin het voorkomt.
- `field_data_naam_field`: Bevat de data bijhorende bij het veld. Data afkomstig van instanties die tot verschillende modules behoren worden in deze tabel opgeslaan. Er worden dus geen aparte datatabellen voorzien per instantie.
- `field_revision_naam_field`: Bevat archiveringswaarden bijhorende bij verschillende *revisions* van een instantie van een veld zodat teruggekeerd kan worden naar een vorige *revision*.

Aanmaken van Fields

Velden kunnen aangemaakt worden met een GUI aangeboden door het Drupal systeem of ze kunnen aangemaakt worden vanuit code. Dit gebeurt dan best op het moment dat een module die ze nodig heeft geïnstalleerd wordt, m.a.w. in de implementatie van `hook_install()`. De velden worden aangemaakt met `field_create_field($field)` en de instanties worden aangemaakt met `field_create_instance($instance)`. Hoe de argumenten voor deze functies opgebouwd zijn is beschikbaar in de Field CRUD API [4].

field_config	field_config_instance	field_data_body	field_revision_body
id INT(11)	id INT(11)	entity_type VARCHAR(128)	entity_type VARCHAR(128)
field_name VARCHAR(32)	field_id INT(11)	bundle VARCHAR(128)	bundle VARCHAR(128)
type VARCHAR(128)	field_name VARCHAR(32)	deleted TINYINT(4)	deleted TINYINT(4)
module VARCHAR(128)	entity_type VARCHAR(32)	entity_id INT(10)	entity_id INT(10)
active TINYINT(4)	bundle VARCHAR(128)	revision_id INT(10)	revision_id INT(10)
storage_type VARCHAR(128)	data LONGBLOB	language VARCHAR(32)	language VARCHAR(32)
storage_module VARCHAR(128)	deleted TINYINT(4)	delta INT(10)	delta INT(10)
storage_active TINYINT(4)		body_value LONGTEXT	body_value LONGTEXT
locked TINYINT(4)		body_summary LONGTEXT	body_summary LONGTEXT
data LONGBLOB		body_format VARCHAR(255)	body_format VARCHAR(255)
cardinality TINYINT(4)			
translatable TINYINT(4)			
deleted TINYINT(4)			
Indexes	Indexes	Indexes	Indexes

Figuur 2.7: Field tabellen

Verwijderen van Fields

Opnieuw kunnen velden verwijderd worden met een GUI aangeboden door het Drupal systeem of het kan vanuit code gebeuren. Via de GUI gebeurt dit impliciet door een contenttype te verwijderen. Achter de schermen wordt de functie `node_type_delete($naam_contenttype)` opgeroepen en in deze functie wordt de functie `field_attach_delete_bundle($entity_type, $bundle)` opgeroepen waar op zijn beurt de functie `field_delete_instance($instance)` oproept. Deze functies die achter de schermen opgeroepen worden wanneer de GUI gebruikt wordt, kunnen eveneens in code gebruikt worden.

2.3.5 Modules

Een extra woordje uitleg hoe een module wordt toegevoegd en verwijderd uit een Drupal systeem kan geen kwaad. We merken meteen op dat er naast deïnstalleren van een module ook *disablen* mogelijk is. Dit kan als gevolg hebben dat een beginnende Drupal ontwikkelaar veronderstellingen maakt die niet volledig kloppen. We maken een opsomming van handige feiten in verband met *enablen*, *disablen* en deïnstalleren voor beginnende Drupal ontwikkelaars:

- Bij *disablen* wordt `hook_disable()` opgeroepen. Dit is analoog voor *enablen*, installeren en deïnstalleren.
- Een gebruiker moet een module altijd eerst *disablen* voor te deïnstalleren.

- Een gebruiker kan een module enkel expliciet *enablen*, *disablen* of deïnstalleren. Een module wordt geïnstalleerd als ze voor de eerste keer enabled wordt of eerst gedeïnstalleerd werd voor te *re-enablen*. Installeren kan dus enkel impliciet.
- Sommige *hooks* worden maar opgeroepen bij installatie van de module, wat betekent dat wanneer je een wijziging van deze *hooks* wil doorvoeren, je de module moet *disablen*, deïnstalleren en *re-enablen*. Deïnstallatie mag niet worden overgeslagen omdat de module dan niet geherinstalleerd wordt. Dit is van toepassing bij volgende *hooks*:

- `hook_node_info()`
- `hook_schema()`

2.3.6 Aanmaken en verwijderen van contenttypes

Aanmaken van contenttypes

Er zijn meerdere manieren om contenttypes aan te maken en te configureren.

- Via een GUI die standaard aangeboden wordt door de Drupal website.
- De functie `node_type_save()` gebruiken om een nieuw type op te slaan. Deze methode kan je overal in code gebruiken.
- `hook_node_info()` implementeren waarin je de nieuwe contenttypes beschrijft. Deze hook wordt enkel opgeroepen bij installatie van een module.

De eerste optie is enkel interessant voor Drupal gebruikers die zelf geen module wensen aan te maken of voor ontwikkelaars die het effect op de databank van het aanmaken van een contenttype willen bekijken. De twee laatste functies kunnen door ontwikkelaars gebruikt worden om contenttypes aan te maken in code. De laatste optie wordt echter algemeen als de betere optie gezien omdat het gebruik van *hooks* de *Drupal way* is. De laatste twee opties zijn echter niet voldoende om een contenttype volledig te configureren. Een aantal specifieke variabelen moeten toegevoegd worden in de *variable* tabel. De variabelen die wij toevoegen, hebben een invloed op de configuratie van de contenttypes die we toevoegen.

De variabelen worden gezet in de implementatie van `hook_enable()` door de module die het contenttype definieert. Deze variabelen moeten voor elk contenttype gezet worden en bevatten de naam van het contenttype waar ze een invloed op hebben. Een volledige lijst van mogelijke

variabelen die een invloed hebben op het contenttype vind je online [16]. We sommen degene op die we gebruiken:

- `comment_naam_contenttype`: Krijgt de waarde 0 om *default* commentaren uit te schakelen.
- `node_options_naam_contenttype`: Krijgt de waarde `array('status')` om *default* 'Promote to Front page' uit te vinken.
- `node_preview_naam_contenttype`: Krijgt de waarde 0 om de mogelijkheid tot een preview te *disablen*.
- `node_submitted_naam_contenttype`: Krijgt de waarde 1 om *default* de auteur en de tijd waarop de content is toegevoegd te tonen bij de content zelf.

Een laatste stap die nog doorgevoerd moet worden is het creëren van de velden die toegevoegd gaan worden aan de contenttypes, vervolgens worden er instanties van de velden aangemaakt om ze te linken aan de contenttypes. Dit gebeurt in `hook_install()`. Er worden velden aangemaakt voor: de URI van devices, de URI van resource en een veld voor meerdere referenties naar content van het type `coap_resource`. Velden werden besproken in paragraaf 2.3.4.

Verwijderen van contenttypes

De contenttypes worden verwijderd in `hook_uninstall()`. Voor een contenttype kan verwijderd worden moeten nog een aantal andere zaken verwijderd worden. We overlopen de verschillende stappen die voor elk te verwijderen contenttype moeten overlopen worden:

- Alle content van het contenttype wordt verwijderd door middel van de functie `node_delete_multiple($nids)`. De variabele `$nid` bevat een array met alle nids van de content die verwijderd wordt.
- Alle variabelen die toegevoegd werden in de *variable*tabel worden verwijderd door gebruik te maken van `variable_del($naam)`;
- De comment-entiteit die geassocieerd is met dit contenttype wordt gemarkeerd om te verwijderen. Hiervoor wordt de functie `field_delete_instance()` gebruikt.
- Het contenttype wordt verwijderd met `node_type_delete($naam_contenttype)`. In deze functie worden de instanties van de velden geassocieerd met dit contenttype ook verwijderd door gebruik te maken van de functie `field_attach_delete_bundle($entity_type, $bundle)`. We merken op dat de commentaren apart verwijderd werden.

- De databankcache van node types wordt geupdated met `node_types_rebuild()`. Dit is nodig omdat Drupal vaak in caches kijkt en verouderde informatie kan gebruiken.
- Drupal zal bij het verwijderen van velden de kolomwaarde `deleted` op 1 zetten. Om de velden effectief uit de databank te verwijderen wordt de functie `field_purge_batch()` gebruikt.

2.4 JQuery in Drupal

In deze paragraaf bekijken we hoe een module (`weather_info`) gemaakt werd die het weerbericht ophaalt voor een regio naar keuze, aangegeven door een WOEID. In eerste instantie werd gewerkt met een formulier, maar in een latere fase werd overgestapt op jQuery, wat de gebruikerservaring bevordert.

2.4.1 Met een HyperText Markup Language (HTML)-formulier

In eerste instantie bevatte de module een formulier bestaande uit een tekstveld als invoer voor de WOEID en een knop om het formulier in te dienen. Wanneer de gebruiker op de knop klikt, gebeuren volgende stappen:

- het formulier wordt ingediend
- de pagina wordt opnieuw geladen
- de *bootstrapcode* roept `hook_form_submit()` op door `weather_location_form_submit()` op te roepen (`weather_location` is de naam van het formulier):
 - het ingegeven WOEID wordt opgeslagen op serverniveau met de Drupal functie `variable_set()`
- de *bootstrapcode* roept `hook_block_view` op van de weermodule door `weather_info_block_view()` op te roepen:
 - het ingegeven WOEID wordt opgehaald met behulp van de Drupal functie `variable_get()`



Figuur 2.8: Weermodule

- er wordt een *HTTP-request* uitgevoerd naar de *Yahoo Weather API* met de PHP-functie `file_get_contents()`, dat een URL als parameter verwacht
 - het ontvangen xml-bestand wordt in een object gestopt met de PHP-functie `SimpleXMLElement()`, waarna het weerbericht gemakkelijk uit het XML-bestand kan gehaald worden
 - Het weerbericht wordt toegevoegd aan de inhoud van het *Block*
- De pagina wordt in de browser weergegeven met het weerbericht in het *Block*

2.4.2 Met AJAX in jQuery

Een pagina zal pas getoond worden wanneer de *bootstrap* afgelopen is en aangezien de code in de *hooks* die worden uitgevoerd onderdeel is van de *bootstrap*, zal de pagina pas getoond worden wanneer de *hooks* afgelopen zijn. Dit heeft als gevolg dat de gebruiker van de website langer moet wachten op de pagina omdat eerst nog een *HTTP-request* moet gebeuren om het weerbericht op te halen. Het spreekt voor zich dat dit een zeer nadelig effect is dat moet vermeden worden.

Als alternatief hebben we gekozen om een jQuery-event te koppelen aan de submit-knop die het formulier indient. jQuery is namelijk geschreven in JavaScript en JavaScript is een *client-side scripting language*, wat inhoudt dat deze code wordt uitgevoerd op de machine van de gebruiker en dit nadat de pagina geladen is.

Wanneer de gebruiker op de knop klikt, wordt een *Asynchronous JavaScript and XML(AJAX)-call* uitgevoerd. Zoals de naam suggereert, is dit een asynchrone aanroep, wanneer het antwoord aankomt, wordt automatisch een opgegeven functie opgeroepen waarin de data kan verwerkt worden. Als gevolg heeft de gebruiker dus geen enkele hinder van het internetverkeer dat noodzakelijk is om het weerbericht op te halen.

2.4.3 Problemen

JQuery laat geen *cross-domain AJAX calls* toe wegens veiligheidsoverwegingen. De *Weather Service API* bevindt zich namelijk op een ander domein. Een oplossing hiervoor is een *proxyscript* in PHP gebruiken. De *AJAX-call* gebeurt dan naar het *proxyscript* dat zich op de server en dus hetzelfde domein bevindt. Het *proxyscript* vraagt daar effectief de data op en geeft de uitvoer terug. De browser wordt dus eigenlijk om de tuin geleid. [33]

Hoofdstuk 3

CoAP: embedded HTTP done right

3.1 Wat is CoAP?

Het Constrained Application Protocol is een *web transfer protocol* speciaal ontwikkeld voor netwerkcomponenten die beperkt zijn in zowel geheugen als energieverbruik, en voor *machine-to-machine* (M2M)-communicatie. Met M2M-communicatie bedoelen we communicatie tussen machines waar geen menselijke tussenkomst nodig is. Zoals de berichten die routers uitwisselen om hun routingtabel te synchroniseren. Naast het minimaliseren van *overhead* concentreert CoAP zich ook op het automatiseren van taken. Het mechanisme van resource *discovery* (Zie paragraaf 3.3.1) is hier een voorbeeld van. Met het verminderen van energieverbruik in het achterhoofd biedt CoAP naast synchrone ook asynchrone communicatie aan. Het biedt ook nieuwe soorten berichten aan zoals *non-confirmable*, *piggy-backed*, etc. Deze worden toegelicht in paragraaf 3.2.1

Een CoAP-programmeur is verplicht applicaties volgens een *Representational Stateless Transfer* (REST)-architectuur op te bouwen. Hierbij wordt een *device* opgesplitst in *resources*, elk geïdentificeerd door een *Uniform Resource Identifier* (URI). Bij CoAP specifiek wordt *Hypermedia as the Engine of Application State* (HATEOAS) gebruikt, dit is een beperkte versie van REST waarbij een client geen voorkennis nodig heeft om een server te bevragen. De informatie hiervoor wordt dynamisch geleverd door applicatieservers.

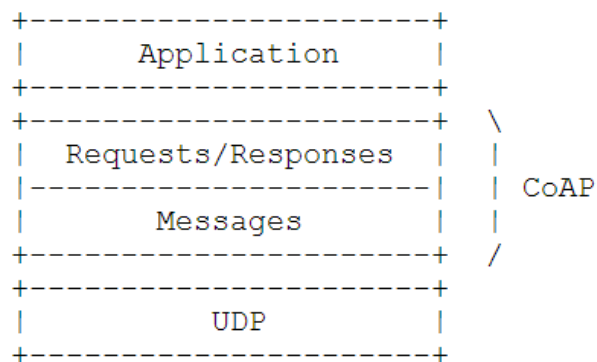
Het interactiemodel van CoAP is vergelijkbaar met het *client/server*model van HTTP. Hoewel, bij een CoAP-implementatie voor M2M interacties kan een *device* bij de ene berichtuitwis-

seling *client* zijn, en bij de andere server. Een CoAP-*request* is equivalent aan een HTTP-*request* en wordt ook gestuurd van de *client* naar de server om een actie aan te vragen op een resource die zich op die server bevindt. De actie wordt bepaald door een method code (GET, PUT, POST of DELETE) en de resource wordt aangeduid met een URI. De server antwoordt met een response die onder andere een response code bevat.

Verschillend met HTTP, gebeurt de berichtenuitwisseling over een UDP. Dit houdt in dat berichten mogelijks verloren gaan of in een andere volgorde kunnen aankomen dan dat ze verzonden zijn. Toch voorziet CoAP enige vorm van betrouwbaarheid met een soort bericht dat kan worden bevestigd door een *Acknowledgement* (zie paragraaf 3.2.1). Wanneer zo'n bericht niet wordt bevestigd, wordt het bericht meermaals opnieuw gestuurd volgens het *exponential back-off* mechanisme (Zie paragraaf 4.2.4).

CoAP definieert vier soorten berichten: *Confirmable*, *Non-confirmable*, *Acknowledgement* en *Reset*. Door gebruik te maken van method codes en response codes transporteren sommige van deze berichten *requests* of *responses*. In paragraaf 3.2 gaan we hier dieper op in.

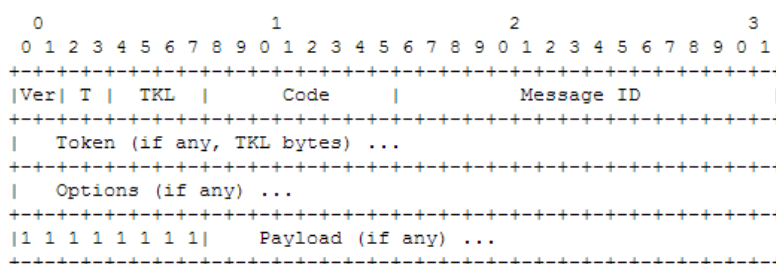
We kunnen CoAP ook in het *Open Systems Interconnection* (OSI)-model met 7 lagen plaatsen. Conceptueel gezien hebben we twee deellagen, CoAP-berichtenlaag die het UDP gedeelte van de berichten afhandelt en een *request*/response laag die gebruik maakt van method en response codes (zie Figuur 3.1). Nogtans bestaat CoAP in werkelijkheid slechts uit één laag waarbij berichtenuitwisseling en het *request*/response mechanisme enkel en alleen door manipulatie van de *header* verwezenlijkt wordt.



Figuur 3.1: CoAP-lagen (CoAP 17 draft)

3.1.1 Berichtformaat

Om een minimale *overhead* te realiseren worden de berichten zeer compact ge-



Figuur 3.2: Berichtformaat (CoAP 17 draft)

houden. We geven een kort overzicht van de onderdelen van het bericht-formaat (zie Figuur 3.2) en bespreken dan de belangrijke delen apart in subparagrafen. De eerste vier bytes stellen de *header* voor. Bemerkt dus dat de *header* gerealiseerd wordt met slechts 4 bytes. Het veld na de *header* is optioneel en bevat een *token* waarvan de lengte aangegeven is in de *header*. Vervolgens zitten er nul of meer opties in het bericht. Het laatste onderdeel van een bericht is de *payload*. Indien er een *payload* aanwezig is in het bericht, wordt die altijd voorafgegaan door een vaste byte, de *payloadmarker* (0xFF). Deze geeft het einde van de opties en het begin van de *payload* aan. Indien er geen *payload* is mag deze marker niet aanwezig zijn.

We merken hier op dat een *token*, opties en een *payload* optioneel zijn. Dit zorgt ervoor dat sommige berichten beperkt blijven tot de *header* van 4 bytes, wat relatief weinig is.

Header

Deze vier bytes worden opgedeeld in drie delen:

- Versiegetal (Ver): een 2-bit *unsigned integer* die de CoAP-versie aangeeft.
- Type-aanduiding (T): een 2-bit *unsigned integer* die het berichttype aangeeft. De mogelijkheden zijn: *confirmable* (CON) (0), *non-confirmable* (NON) (1), *Acknowledgement* (ACK) (2) of *Reset* (RST) (3).
- *tokenlengte* (TKL): een 4-bit *unsigned integer* die de variabele *tokenlengte* aangeeft.
- Code: 8-bit *unsigned integer* die aangeeft of het bericht een *request* of een response overbrengt, of leeg is.
- *message-ID*: een 16-bit *unsigned integer* die gebruikt wordt om duplicatie van berichten op te merken. Het wordt ook gebruikt om berichten van het type ACK/RST te linken aan berichten van het type CON/NON.

Token

Een *token* wordt gebruikt om een response te linken aan een *request*. De lengte wordt bepaald door de TKL en is nul tot acht bytes lang. Elk bericht heeft een *token*, dit kan lengte nul

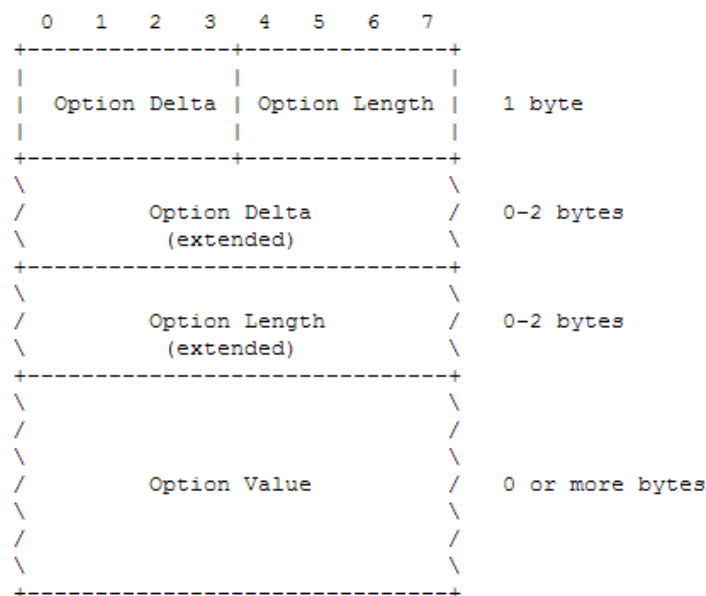
hebben. Wanneer een *token* nodig is, moet dat worden gegenereerd door de *client*. Indien de server wil dat zijn response geaccepteerd wordt, moet hij dit *token* zonder meer overnemen in zijn response.

Opties

Opties worden opgesteld door middel van de Type-Length-Value (TLV) notatie (zie Figuur 3.3). Er wordt een mechanisme toegepast om opties in een pakket te stoppen, dat ervoor zorgt dat het pakket compact blijft en dus bijdraagt tot een minimale *overhead*.

Elke optie heeft een uniek nummer, maar wanneer meerdere opties in één pakket worden gestopt, worden de opties niet door dat nummer aangeduid, maar door de delta-encoding in het *Option Delta*-veld. De *Option Delta* is het verschil tussen het nummer van een optie en dat van de vorige optie. Concreet, stel dat men na een optie met nummer 6 een optie met nummer 11 wil plaatsen, dan wordt deze laatste aangeduid met *Option Delta* gelijk aan 5 (11 - 6). Dit alles samen maakt dat een minimale (lege, maar daarom niet nutteloze) optie slechts 1 byte in beslag neemt. Dit heeft als gevolg dat opties na elkaar moeten worden geplaatst met oplopende optienummers.

Er zijn gevallen dat dit systeem tekort schiet. Daarom zijn er speciale waarden ingevoerd die het gebruik van een *extended Option Delta* aangeven. Indien de *Option Delta* groter is dan 12, maar kleiner is dan 269, wordt als *Option Delta* waarde 13 gebruikt. Er wordt dan een extra byte voor de Option Value gezet dat opgevuld wordt met de echte *Option Delta* - 13. Indien de Option delta meer is dan 268, wordt als *Option Delta* de waarde 14 gebruikt. Deze waarde geeft aan dat er twee extra bytes gebruikt worden. Deze bytes worden opgevuld met de echte *Option Delta* - 269. De waarde 15 mag niet als *Option Delta* gebruikt worden want deze geeft het begin van de *payload* aan. Voor Option Length zijn de regels analoog.



Figuur 3.3: CoAP-optie (CoAP draft 16)

Verskil met HTTP

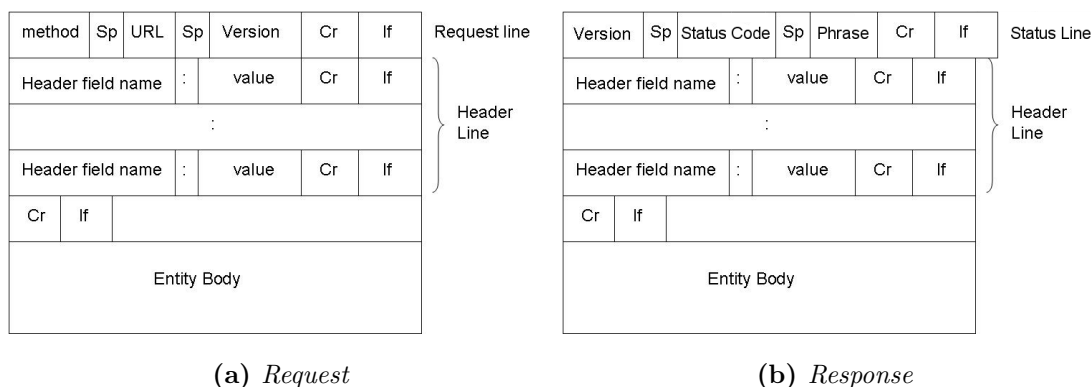
Als we dit kort vergelijken met het berichtformaat van HTTP (zie Figuur 3.4) zien we dat er aanzienlijke verschillen zijn. Bij HTTP zijn *request* en *response* niet helemaal gelijk. We kijken eerst naar de HTTP-*request* die opgedeeld is in drie delen:

- Een *request*lijn die op zijn beurt opgedeeld is in drie delen gescheiden door een spatie:
 - de methodenaam (GET, POST, HEAD, PUT of DELETE voor HTTP 1.1)
 - de URL van de gevraagde resource
 - het versienummer
- Een aantal *header*lijnen die bijkomende opties voorstellen.
- De entity body wordt gebruikt door de POST-methode om gegevens door te sturen en wordt gescheiden van de *header*lijnen door een lege lijn.

Bijkomend wordt elke lijn (ook de lege) afgesloten met een *carriage return* (CR) en een *line feed* (LF).

De HTTP-response is analoog aan de *request* met als verschillen dat de eerste lijn opgebouwd is uit het versienummer, de statuscode die aangeeft wat het resultaat van de *request* inhoudt en een korte beschrijving van de statuscode.

Het is dus duidelijk dat een HTTP-bericht aanzienlijk groter zal zijn dan een CoAP-bericht [29].



Figuur 3.4: HTTP-berichtformaat

3.2 Communicatiemogelijkheden

In deze paragraaf bespreken we de communicatiemogelijkheden van CoAP. We bekijken de variabele betrouwbaarheid van CoAP-berichten en gaan na hoe het *request/response*model bij CoAP werkt aan de hand van voorbeelden. Als laatste lichten we het principe van *blockwise transfer* toe.

3.2.1 Betrouwbaarheid

HTTP realiseert een betrouwbare en robuuste vorm van communicatie. Het is gebaseerd op het *Transmission Control Protocol* (TCP). Dit protocol zet een verbinding op aan de hand van *stream sockets*. Het zorgt ervoor dat pakketten gegarandeerd aankomen bij de bestemming en dit in volgorde van verzending. Maar deze betrouwbaarheid komt met een prijs, namelijk extra netwerkbelasting voor het opzetten en beheren van die verbinding. In tegenstelling tot HTTP dat gebouwd is op TCP, is CoAP gebaseerd op berichtenuitwisseling over UDP. Wanneer men met dit protocol werkt, is er echter geen garantie dat pakketten aankomen en wanneer dat wel het geval is, kan de volgorde van aankomst gewijzigd zijn ten opzichte van verzending. Daarom moet men bij CoAP zelf de betrouwbaarheid verzorgen indien nodig.

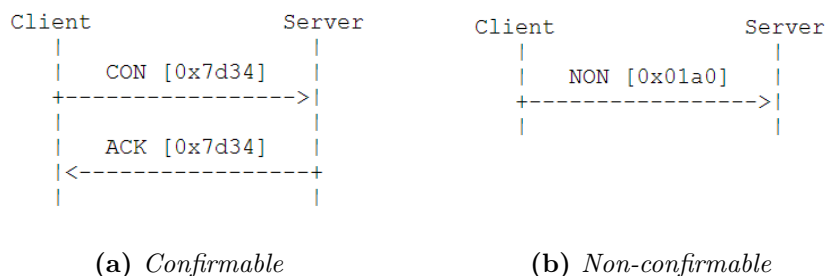
Confirmable berichten

Wanneer we de betrouwbaarheid van de berichtenuitwisseling willen opdrijven, kiezen we voor *confirmable* berichten. Een *CON-request* moet door de server worden beantwoord met een ACK- of RST-bericht (zie Figuur 3.5), dit ACK-bericht moet hetzelfde *message-ID* bevatten als het CON-bericht waarop geantwoord wordt. Wanneer CON-berichten niet worden beantwoord met

een ACK-bericht vóór een bepaalde time-out, wordt het bericht opnieuw verzonden. Bij het opnieuw verzenden wordt een *exponential back-off* mechanisme toegepast. Eerst wordt een time-out bepaald tussen een `ACK_TIMEOUT` en `ACK_TIMEOUT x ACK_RANDOM_FACTOR`, wanneer die time-out verstrijkt wordt het CON-bericht opnieuw verzonden en de time-out verdubbeld. Wanneer de server niet in staat is het CON-bericht te verwerken, wat betekent dat die zelfs geen geldige *errorresponse* kan geven, antwoordt die met een RST-bericht in plaats van met een ACK-bericht.

Non-confirmable berichten

Soms heeft een bericht geen betrouwbaar transport nodig. Een voorbeeld hiervan is een stroom van sensordata waarbij elke meting verstuurd wordt met een NON-bericht. Dit soort berichten wordt niet bevestigd met een ACK bericht, maar de berichten hebben wel nog steeds een *message-ID* om duplicatie van berichten te detecteren (zie Figuur 3.5). Wanneer een ontvanger niet in staat is het bericht te verwerken, opnieuw bedoelen we daarmee dat het geen geldige *errorresponse* kan geven, zendt die een RST-bericht naar de zender.

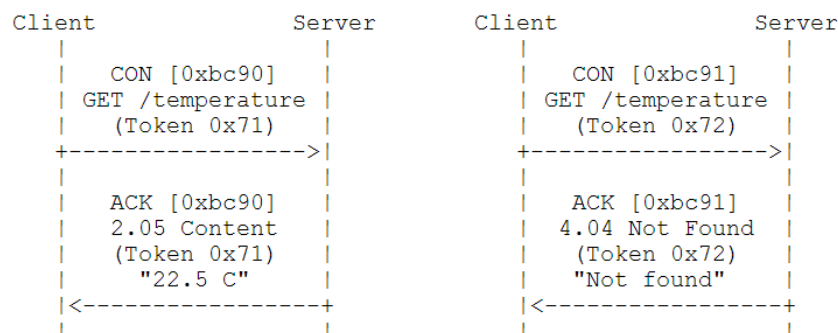


Figuur 3.5: Berichtuitwisseling (CoAP 17 draft)

3.2.2 Request/responsemodel

Piggy-backed response

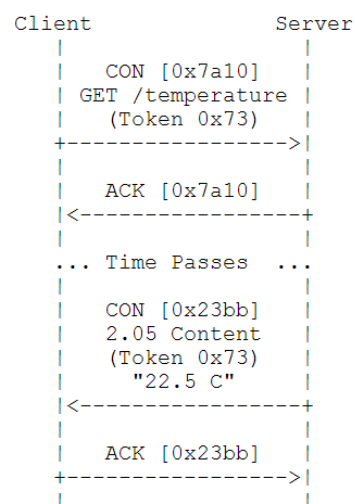
Wanneer een *request* met een CON-bericht verstuurd wordt, is het mogelijk dat het antwoord meteen beschikbaar is bij de server. Indien dit het geval is, wordt het antwoord op de *request* meteen meegestuurd met het ACK-bericht. Dit wordt een *piggy-backed* response genoemd. In Figuur 3.6 worden twee voorbeelden van GET-requests met *piggy-backed* responses getoond. De ene is succesvol, de andere geeft een *errorresponse* terug.



Figuur 3.6: Twee GET-requests met piggy-backed responses (CoAP 17 draft)

Separate response

Wanneer de server niet onmiddellijk kan antwoorden op de *request* van de *client*, antwoordt die met een leeg ACK-bericht zodat de *client* niet zou beginnen heruitzenden als gevolg van het *exponential back-off* mechanisme. Wanneer de response klaar is, stuurt de server dit antwoord in een nieuw CON-bericht dat op zijn beurt beantwoord moet worden door de *client*. Dit soort van berichtenuitwisseling heet *separate response* (zie Figuur 3.7).



Figuur 3.7: GET-request met separate response (CoAP 17 draft)

3.2.3 Blokken

CoAP werkt goed als de berichten geen of kleine *payloads* bezitten. Dit mag je veronderstellen als je werkt met temperatuursensors of lichtschakelaars. Maar soms is het nodig dat applicaties berichten sturen met grotere *payloads*, zoals bij resource *discovery* dat we in 3.3.1 bespreken of bij resources die nu eenmaal veel gegevens als antwoord op een *request* hebben.

Bij HTTP doet TCP (op de transportlaag) al het werk om de pakketten te fragmenteren en ervoor te zorgen dat de deelpakketten in de juiste volgorde verwerkt worden. Dit is niet mogelijk bij CoAP aangezien UDP als transportprotocol gebruikt wordt. CoAP voorziet zelf een vorm van fragmentatie. Meerdere blokken informatie van een enkele resource worden uitgewisseld via meerdere *request*/responseparen. Belangrijk op te merken hierbij is dat elk *request*/responsepaar afzonderlijk af te handelen is door *client* en server. Dit heeft als gevolg dat er geen connectie

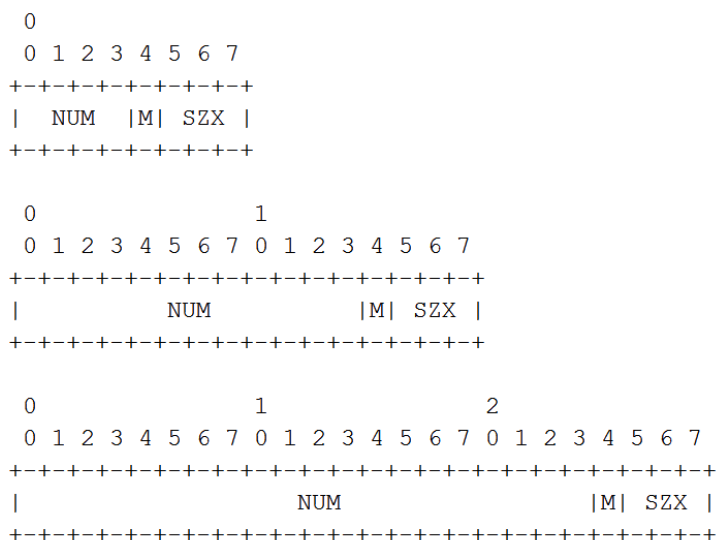
opgezet moet worden en dat de server geen geheugen moet vrijmaken om bij te houden welke blokken hij al verstuurd heeft. Waarom er voor de Block Options gekozen is wordt uitvoerig besproken in de draft '*Blockwise transfers in CoAP*' [34].

Block Options

Om grote *payloads* op een goede manier te ondersteunen worden twee Block Options ingevoerd. Optie 23 heeft als naam Block2 en geeft informatie over de *payload* van de response. Optie 27 heeft als naam Block1 en geeft informatie over de *payload* van de *request*. Beide opties geven een *blockwise transfer* aan en kunnen zowel in een *request* als in een response voorkomen. Ze ondersteunen blokgroottes die een macht van twee zijn, beginnend bij 16 t.e.m. 1024 bytes.

Structuur

Bij beide opties hebben de optiewaarden dezelfde structuur. De lengte van de waarde is variabel en kan 1, 2 of 3 bytes lang zijn zoals te zien is in figuur 3.8.



Figuur 3.8: Blokoptiewaarde - bytes en bits worden aangegeven bovenaan de figuur

Wat we nog uit figuur 3.8 halen is dat een *Block*-optie telkens drie elementen bevat:

1. NUM (*Block Number*): Het relatief nummer van het blok binnen een sequentie van blokken.
2. M (*More Flag*): Is dit het laatste blok in een reeks?
3. SZX (*Size Exponent*): Geeft de grootte van het blok.

De blokgrootte is geëncodeerd gebruik makend van een drie-bit *unsigned integer*. 0 komt overeen met 2^4 en 6 komt overeen met 2^{10} . Concreet wordt de grootte van het blok gegeven door 2^{SZX+4} . 7 mag niet gebruikt worden als waarde omdat deze gereserveerd is.

3.2.4 Observe

Wanneer een resource *observable* is of anders gezegd, de *observe*functionaliteit ondersteunt, kan die resource op eigen initiatief data sturen naar eventueel geïnteresseerde *clients*. Een *client* kan zijn interesse uiten door een CON-bericht met een lege *observe*-optie (optie 6) naar de server te sturen. Wanneer de *client* dan een ACK-bericht terug krijgt met een *observe*-optie, weet die dat de server de *client* heeft toegevoegd aan de lijst van *observers*. Een *client* kan aangeven aan de server dat die niet meer geïnteresseerd is door een RST-bericht te sturen naar de server, als alternatief kan de client een *request* sturen zonder *observe*-optie. De server verwijdert de *client* dan uit de lijst van geïnteresseerden.

3.3 Extra Features

3.3.1 Resource discovery

Resource *discovery* zorgt ervoor dat lokale *client* en servers elkaar kunnen vinden en met elkaar kunnen interageren zonder menselijk tussenkomst.

Principe

Resources kunnen maar aangesloten zijn op één *device*. Dit *device* maakt een extra resource aan met als naam '*well-known/core*'. Deze extra resource bevat een opsomming van alle resources verbonden met dit *device* en een aantal gegevens over deze resources onder de vorm van attributen. Sommige *well-known/core*'s bevatten ook verwijzingen naar resources op andere *devices*. Gebruikers kunnen een GET-*request* sturen om de *well-known/core* op te halen en hebben dan deze informatie tot hun beschikking. De *well-known/core* bevat echter veel informatie die niet in een enkel *request/response*paar uitgewisseld kan worden. Daarom wordt er gebruik gemaakt van blokken, welke uitgelegd worden in 3.2.3. De manier waarop deze informatie doorgegeven wordt, is door gebruik te maken van het *CoRE Link Format* [38].

CoRE Link Format

Er worden geen resources opgeslaan in de *well-known/core* maar links naar resources. Als formaat van de links moet het *CoRE Link Format* gebruikt worden. Een resource wordt binnen het device uniek geïdentificeerd door een *target-URI*. De *target-URI* in combinatie met de URI van het device vormt de URI van de resource waarmee hij globaal uniek geïdentificeerd wordt. Deze *target-URI* staat tussen een kleiner-dan (<) en een groter-dan (>) teken. Na de URI worden alle attributen opgesomd gescheiden door een puntkomma (;). De attributen zelf hebben de vorm van een naam-waardepaar. Sommige attributen kunnen meerdere waarden hebben, deze waarden worden dan gescheiden door een spatie. Tot slot worden verschillende links gescheiden met een komma (,).

Als voorbeeld geven we het antwoord op een GET-request van een *well-known/core*:

```
</temp>;rt="temp-c";if="sensor";ct="0 40",</light>;rt="light-lux";if="sensor"
```

We zien dat het *device* met deze *core* twee resources aanbiedt. De eerste resource wordt geïdentificeerd met de *target-URI* temp, de andere met light. De eerste resource heeft drie attributen en ondersteund twee *content formats*. De tweede resource heeft twee attributen. Er zijn echter meer attributen mogelijk. We sommen enkele algemene attributen op:

- obs (*Observable*): Geef aan of de resource *observable* is. Indien de optie ontbreekt wordt aangenomen dat de resource niet *observable* is.
- rt (*Resource Type*): Geeft het type van de resource. De waarden zijn afhankelijk van de gebruikte applicatie. Ze hebben als nut dat binnen een applicatie, resources van hetzelfde type herkent kunnen worden. Dan is het mogelijk via query filtering, alle resources van een type opgehaald worden. Meerdere waarden zijn mogelijk.
- ct (*Content Type*): Bevat een opsomming van de *content formats* die ondersteund worden door deze resource. Meerdere waarden zijn mogelijk.
- if (*Interface Description*): Bevat een beschrijving van welke methoden deze resource ondersteunt. Meerdere waarden zijn mogelijk.
- sz (*Maximum Size Estimate*): Bevat een schatting van de maximale grootte van een response. Voor grote resources zou dit geïmplementeerd moeten worden maar het is geen verplichting.

- title: Bevat een omschrijving van de resource.
- anchor: Wordt gebruikt om de link te laten verwijzen naar een willekeurige resource. Dit kan een resource binnen de eigen *core* zijn of erbuiten. Het rel-attribuut wordt vaak samen met deze optie gebruikt om de relatie aan te duiden tussen de link en de resource. Implementaties die het vermogen niet hebben deze optie te verwerken moeten de hele link te negeren.
- rel: Geeft de relatie tussen de link en de resource.

Al deze attributen mogen per resource maar een keer voorkomen. De lijst met alle attributen vind je in de *CoRE Link Format draft* [38]. Maar niet alle beschrijvingen van deze attributen staan in de *CoRE Link Format RFC*. De ontbrekende beschrijvingen vind je in de *Web Linking RFC* [37].

Core ophalen

Om een *well-known/core* op te halen, moet je twee maal de Uri-Path optie (optie 11) toevoegen aan een GET-request die gericht is naar het IP-adres van het *device* waar de *core* zich op bevindt. De eerste keer met als waarde 'well-known' en de tweede keer gebruik je 'core' als waarde.

We bespreken de eerste berichtenparen bij een resource *discovery* door de hexadecimale waarde te bekijken. Verschillende onderdelen binnen het bericht scheiden we met een streep.

De *client* stuurt een eerste *discovery request*:

```
40 01 78 a5 | bb 2e 77 65 6c 6c 2d 6b 6e 6f 77 6e | 04 63 6f 72 65
```

Het eerste deel bevat dezelfde elementen als elke GET-request. Het tweede deel bevat de eerste optie met als eerste byte 'bb'. Dit komt overeen met optie 11 met lengte 11. De 11 volgende byte zijn de hexadecimale waarden van de karakters 'well-known'. Het laatste deel bevat ook een optie. Deze keer zijn de twee eerste bytes '04'. De *Option Delta* is 0, dus hebben we opnieuw optie 11. De lengte is deze keer 4. De volgende byteparen komen overeen met 'core'.

Het antwoord van de server:

```
60 45 78 a5 | c1 28 | b1 0c | 52 06 92 | ff ...
```

Het eerste deel is analoog aan andere responseberichten. We merken wel op dat de *message-ID*

(0x78a5 in het antwoord) dezelfde is als de *message-ID* in de *request* van de *client*. De eerste twee bytes van het tweede deel zijn 'c1'. Wat optie 12 (*Content-Format*) met lengte 1 impliceert. De waarde van deze optie is 28. Omgezet naar decimaal is dit 40, wat application/link-format aangeeft. Het volgende deel begint met 'b1'. Dit komt overeen met optie 23 (Block2) met lengte 1. De hexadecimale waarde is '0c' wat overeenkomt met 12 of binair 00001100. Als we dit herschikken zien we de elementen die in figuur 3.8 te zien zijn: 0000 1 110. We zien dat dit het eerste blok is van een reeks van blokken met grootte 1024 (2¹⁰) bytes. We zien ook dat er nog blokken aangevraagd moeten worden. Het voorlaatste deel begint met '52' wat optie 28 aanduidt met lengte 2. Optie 28 (*Block Size*) mag genegeerd worden en wordt besproken in 6.10. Het laatste deel begint met de *payloadmarker* (0xff) waarop de *payload* volgt.

De *client* stuurt een nieuwe *request* om het volgende pakket op te vragen:

40 01 78 a6 | bb 2e 77 65 6c 6c 2d 6b 6e 6f 77 6e | 04 63 6f 72 65 | c1 16

De eerste drie delen zijn analoog aan de eerste *request*. Het laatste deel dat erbij is gekomen geeft het volgende blok aan dat verstuurd moet worden door de server. De eerste twee bytes zijn 'c1'. *Option Delta* is 12 dus de optie is $11 + 12 = 23$ (Block2). De lengte is 1 bytepaar. 16 binair is 0001 0 110. Het tweede blok in de reeks van blokken met lengte 1024 bytes wordt opgevraagd.

De server zal met het aangevraagde blok antwoorden. Dit proces blijft zich herhalen tot de server een response stuurt waar de M-bit op 0 staat.

Query Filtering

Het is mogelijk een specifieke resource *discovery* uit te voeren indien er enkel informatie over een resource moet gekend zijn of enkel de resources waarbij een bepaald attribuut een specifieke waarde heeft. Dit heeft als voordeel dat het netwerk niet overbodig belast moet worden als dat niet nodig is. De *queryfilter* wordt voorgesteld door een naam-waardepaar. Deze wordt toegevoegd door de *Uri-Query*optie (optie 15) te gebruiken.

Hoofdstuk 4

CoAP library module

In hoofdstuk 3 bekeken we de definitie, aspecten en functionaliteit van CoAP. In dit hoofdstuk bekijken we hoe een eigen CoAP-*library* voor CoAP versie 13 in PHP werd ontwikkeld als Drupalmodule. De module stelt een Drupalgebruiker in staat CoAP-berichten op te stellen, te versturen en te ontvangen. Let wel, deze module heeft niet als doel om visueel iets voor te stellen op een website. Ze biedt enkel functionaliteit en is bijvoorbeeld te gebruiken in een andere module die dan uitgewisselde gegevens kan omvormen tot een visuele representatie. Dit laatste is wat wij doen in de uiteindelijk te ontwerpen module welke wordt toegelicht in hoofdstuk 5.

4.1 Doel

Wanneer we trachtten CoAP-berichten op te stellen, te versturen en ontvangen in onze module, werd al snel het nut duidelijk van een aparte module die zou fungeren als CoAP-*library*. Er ontstaat namelijk erg veel duplicatie van code, bovendien wordt de code onoverzichtelijk en slordig. Nog een zeer nadelig gevolg is dat de code niet hergebruikt kan worden in andere modules. Men zou al de code moeten doorspitten om net dat stuk te vinden waar CoAP gesproken wordt. Bovendien is het niet zeker dat die code zal voldoen aan de waarschijnlijk andere eisen van de nieuwe gebruiker. Wanneer de code niet fungeert in de nieuwe situatie, zal er veel tijd gespendeerd moeten worden aan debuggen en zal de frustratie oplopen bij de gebruiker. Kortom, het is duidelijk dat dit een zeer slechte praktijk is die ten alle kosten moet vermeden worden, het druist in tegen enkele van de belangrijkste programmeerprincipes.

Met het nut bewezen bekijken we even wat een gebruiker van deze *library* kan verwachten. Ze moet de gebruiker in staat stellen dynamisch berichten op te stellen, volledig naar eigen wens. Dit wil zeggen dat de verzameling operaties om een bericht op te stellen, alle mogelijke berichten moet kunnen genereren. Dit heeft als gevolg dat elke elementaire operatie moet kunnen worden uitgevoerd op een bericht. Opstellen van bijvoorbeeld een bepaald bericht aan de hand van slechts één functie is niet prioritair en dus bijkomstig. Het is uiterst belangrijk bij het ontwerp van deze *library* dat de gebruiker zich niet meer hoeft te bekommeren om technische details zoals hexadecimale code die nodig is in een bericht. Alle technische details zitten verborgen in de *CoAP-library* module. We besluiten dat de *library* een gebruiker in staat moet stellen dynamisch en gebruiksvriendelijk berichten op te stellen, en dit met een minimum aan code.

4.1.1 Essentiële functies

We sommen enkele van de eerder vermelde elementaire functies op, deze opsomming is allesbehalve limitatief:

- Opstellen van de *CoAP-header* en *CoAP-token* waarbij de gebruiker het type, de methode en het *token* zelf opgeeft. De *token* lengte wordt afgeleid uit het *token*. Wanneer geen *token* gewenst is, kan de gebruiker een lege string meegeven.
- Een *CoAP-optie* toevoegen aan het bericht waarbij de gebruiker slechts het optienummer en de waarde van de optie moet meegeven. De gebruiker hoeft zich hierbij niet te bekommeren over de encoding van opties in het resulterende *CoAP-bericht*.
- De *payload* toevoegen aan het bericht waarbij de gebruiker slechts de *payload* zelf hoeft op te geven. Dit houdt in dit geval in dat de *payload marker* wordt toegevoegd en dat de lengte van de *payload* wordt afgeleid uit de *payload* zelf.
- Een bericht versturen, hierbij onderscheiden we 2 mogelijkheden:
 - Het sturen van een bericht zonder *observe*-optie: Hierbij hoeft men enkel rekening te houden met één antwoord. Men kan dus het antwoord teruggeven als teruggeefwaarde van de functie die het bericht stuurt. Men hoeft zich niet te bekommeren om meerdere antwoorden die op willekeurige tijdstippen kunnen aankomen. Wel is het nodig rekening te houden met resources die grote antwoorden eventueel bloksgewijs opsturen. Hierbij moet de client zelf initiatief nemen om de volgende blokken iteratief

op te halen. De client geeft hierbij bij elk blok aan welk blok hij wil ophalen (Zie paragraaf 3.2.3). Ook kan het gebeuren dat een server niet onmiddellijk kan antwoorden, de server kan dan kiezen om gewoon het antwoord laat te sturen of om gebruik te maken van *separate response*. De server stuurt dan een lege ACK terug om de client ervan te vergewissen dat de aanvraag ontvangen is en zal dan later, wanneer de waarde beschikbaar is, antwoorden met een afzonderlijk antwoord (Zie paragraaf 3.2.2).

- Het sturen van een bericht met *observe*-optie: Het antwoord dat hierop terug gestuurd wordt, bevat slechts één resourcewaarde als *payload* en is slechts een bevestiging dat de server de client heeft toegevoegd aan de lijst van geïnteresseerden. De server zal nu op willekeurige tijdstippen een antwoord sturen op eigen initiatief. Er moet dus een mechanisme voorzien worden dat het aankomen van die berichten opvangt. We zullen later in dit hoofdstuk zien (Zie paragraaf 4.2.3) hoe we deze probleemstelling in de *library* hebben opgelost.

Hierbij wordt de gebruiksvriendelijkheid doorgetrokken, de gebruiker staat namelijk niet in voor de controle van de aanwezigheid van een *observe*-optie. Dit wordt automatisch gedetecteerd door de CoAP-*library*.

- Operaties om een bericht te ontmantelen, hieronder verstaan we het parsen van onder andere de *payload*, een optie op basis van optienummer, het *message-ID*, het *token*, ...

Merk op dat bij het genereren van CoAP-berichten ook een *message-ID* nodig is. Ook hierop is de CoAP-*library* voorzien, er wordt bijgehouden welke *message-ID*'s al gebruikt zijn door telkens het laatst gebruikte *message-ID* te incrementeren. Dit is een mechanisme dat aangeraden wordt in de CoAP-*draft* [11].

Een soortgelijk principe wordt toegepast wanneer een *token* moet worden gegenereerd. De gebruiker moet dan wel een lengte aangeven.

Ook wanneer een ACK of RST moet worden gegenereerd, wordt door aan te geven op welk bericht het antwoord moet worden gestuurd, automatisch het *message-ID* uit het oorspronkelijk bericht geparsed en in het antwoord geplaatst. Hetzelfde principe geldt wanneer een *token* moet worden overgenomen.

4.1.2 Optionele functies

Sommige berichten zijn vaker nodig dan andere. Om de gebruiksvriendelijkheid van de *library* te verhogen en de drempel om ze te gebruiken te verlagen, is het een goed idee enkele functies te voorzien om bepaalde volledige berichten op te stellen. Belangrijk hierbij is dat deze opgestelde berichten nog steeds aanpasbaar moeten zijn. Dit zorgt ervoor een gebruiker bijvoorbeeld al een basisbericht kan opstellen en het daarna nog naar eigen wens kan aanpassen, wat de gebruiker veel tijd en code bespaart. We geven enkele voorbeelden die ook geïmplementeerd zijn in de *library*, deze opsomming is alweer niet limitatief:

- Opstellen van een basis GET-request: hierbij krijgt de gebruiker meteen de kans om een *Uri-Path* mee te geven aan het bericht. De benodigde *Uri-Path*-optie(s) worden dan automatisch toegevoegd door de *library* op basis van de opgegeven Uri-Path. Wanneer de gebruiker geen *URI-path* meegeeft, wordt er ook geen optie toegevoegd aan het bericht.
- Opstellen van een basis GET-request met *observe*-optie: deze operatie is zeer vergelijkbaar met de vorige, het enige verschil is dat er een lege *observe*-optie wordt toegevoegd. Het is hierbij verplicht een *token* op te nemen in het bericht, hiervoor moet de gebruiker enkel de lengte van het token opgeven. Bij ontvangst van deze *request* zal de server, waarnaar het bericht wordt verstuurd, de *client* toevoegen aan de lijst met geïnteresseerden indien de server *observable* is. Om de goede programmeerprincipes te volgen wordt in deze operatie gebruik gemaakt van de vorige operatie. Dit om duplicatie van code te vermijden.
- Automatisch genereren van een ACK op basis van een opgegeven bericht: de gebruiker hoeft enkel het bericht mee te geven waarop de ACK moet worden opgesteld. Het *message-ID* wordt automatisch overgenomen uit het oorspronkelijk bericht. Naast het feit dat de gebruiksvriendelijkheid verhoogd wordt, heeft deze operatie nog een niet te verwaarlozen nut. De kans op fouten wordt namelijk veel kleiner omdat de ACK automatisch gegenereerd wordt, menselijke fouten zijn dus zo goed als uitgesloten als deze operatie op punt staat.
- Automatisch genereren van een RST op basis van een opgegeven bericht: deze operatie is volledig analoog aan de vorige met dat verschil dat het type van dit bericht RST is in plaats van ACK.

4.2 Implementatie

In deze paragraaf wordt besproken hoe de *library* effectief geïmplementeerd werd. De volgorde van de subparagrafen is tevens chronologisch.

4.2.1 Proceduregericht

De eerste versie van de CoAP-*library* werd proceduregericht geprogrammeerd. Dit houdt in dat geen staten of objecten bijgehouden worden. Bijgevolg moet de gebruiker zelf het bericht bijhouden onder een vorm die hij/zij zelf kiest. Deze versie van de CoAP-*library* verwachtte echter het bericht onder de vorm van een hexadecimale string. Met een hexadecimale string bedoelen we een string die zich beperkt tot 16 hexadecimale karakters. Alle benodigde operaties die eerder vermeld werden, werden geïmplementeerd. Alhoewel deze eerste versie al zeker een stap in de goede richting was, zijn er toch wel enkele nadelen die de kop opsteken. Zoals eerder al vermeld, is de gebruiker verplicht zelf de hexadecimale string bij te houden die het bericht voorstelt. Niet elke gebruiker is bekend met het gebruik van hexadecimale strings. Een tweede nadeel heeft te maken met bescherming van het bericht. Wanneer de gebruiker de hexadecimale string bijhoudt, kan die worden aangepast naar believen. Het bericht kan zo vervormd en mogelijks nutteloos worden. Bovendien kan de gebruiker zonder het te beseffen het bericht door één of andere operatie omvormen tot een normale string met *American Standard Code for Information Interchange* (ASCII)-karakters. Dit heeft mogelijk als gevolg dat de *library* het bericht zal omvormen tot een verkeerde hexadecimale string. Kortom, er is nog ruimte voor verbetering en een objectgerichte implementatie lijkt voor de hand liggend.

4.2.2 Objectgericht

In de vorige subparagraaf zagen we de nadelen van een proceduregerichte oplossing. Het zijn net deze punten waar een objectgerichte oplossing goed op scoort. De tweede versie van de CoAP-*library* maakt gebruik van twee klassen welke besproken worden in de volgende subparagrafen.

CoAPFactory

Zoals de naam al suggereert, wordt deze klasse gebruikt voor generatie van CoAP-berichten. De gebruiker maakt eerst een instantie aan van de CoAPFactory-klasse. Dit object kan dan gebruikt worden om CoAP-berichten aan te maken. Het is deze klasse die de eerder besproken handige functies (zie 4.1.2) implementeert om bepaalde berichten aan te maken, samen met de

operatie die de *header* en *token* opstelt. Alle operaties die een berichtobject aanmaken geven een object terug van de klasse `CoAPMessage`, deze klasse wordt hierna besproken. De constructor van deze *factory*-klasse ontvangt volgende parameters:

- **Modulenaam:** deze parameter bevat de naam van de module die gebruik maakt van de *CoAP-library*, de reden voor deze parameter wordt hierna uitgelegd in paragraaf 4.2.3.
- **IPv6-adres:** deze parameter bevat een geldig IPv6-adres, dit is het IPv6-adres van het *embedded device* waarop de beoogde resource is aangesloten.
- **Uri-path:** deze parameter bevat een *Uri-path* van een resource op het *embedded device* dat aangegeven wordt door het IPv6-adres. Deze parameter mag achterwege gelaten worden, het *Uri-path* is dan leeg en duidt het *embedded device* zelf aan.

Zoals blijkt uit de parameters wordt een *factory* per gewenste resource aangemaakt.

CoAPMessage

Deze klasse representeert het bericht zelf. Het bevat alle onderdelen van het betreffende bericht in een gemakkelijk te gebruiken vorm. Dit heeft als voordeel dat de gebruiker van de *library* zich bijvoorbeeld niet hoeft te bekommeren om volgorde van toevoegen bij opties. Wanneer het bericht verzonden moet worden, wordt de hexadecimale string opgesteld om daarna verzonden te worden. De *library* verzamelt dus eerst alle gegevens en zorgt er zelf voor dat het bericht correct wordt opgesteld. Het is deze klasse die de essentiële operaties (Zie paragraaf 4.1.1) implementeert op het opstellen van de *header* en *token* na, zodanig dat het bericht kan worden aangepast naar eigen wensen en noden. Het bericht kan dan gemakkelijk worden verstuurd door een *send* operatie op het object uit te voeren. Een IPv6-adres opgeven hoeft niet meer, dit is reeds gebeurd bij constructie van de *factory* die dit bericht heeft aangemaakt. Aangezien CoAP werkt met UDP, moet er een UDP-socket geopend worden. Bovendien moet in het geval de *observe* functionaliteit gebruikt wordt, de *socket* onderhouden worden. Daar berichten later kunnen toekomen op initiatief van de server. Dit is ook het geval bij *separate response*.

Deze UDP-socket wordt geopend in een achtergrondproces, zodat de *socket* kan gebruikt worden zonder andere processen te blokkeren. Het betreft hier een achtergrondproces dat gemaakt wordt in Drupal aan de hand van de *Background Process*-module [20] (Zie paragraaf 5.2.2). Bijgevolg is deze implementatie niet gebonden aan een bepaalde webserver (Apache, nginx, ...).

4.2.3 Hooks voor notificaties

Eerder werd al aangehaald dat er een speciale voorziening nodig is voor de *observe*-functionaliteit. Hierbij kan de server op eigen initiatief notificaties sturen naar geïnteresseerde *clients*. Enkele alternatieven werden bedacht en overwogen, het een al beter dan het ander:

- *Socket* teruggeven: Dit houdt in dat de *send*-functie die opgeroepen wordt in de CoAP-*library* als teruggeefwaarde de *socket* zou teruggeven. Dit heeft echter als gevolg dat een gebruiker verder zelf de *socket* moet onderhouden en telkens de nodige berichten moet opbouwen, versturen en ontvangen. Het spreekt voor zich dat dit geen goede oplossing is. De CoAP-*library* wordt zo allesbehalve modulair en een eventuele gebruiker ervan moet al een redelijke hoeveelheid technische kennis hebben om ze te gebruiken.
- Databank: bij dit alternatief zou men de *socket* toch beheren in de CoAP-*library* zelf en de notificaties opslaan in de databank. Men kan dan bijvoorbeeld een gebruiker van de module een naam van een databanktabel laten opgeven en de notificaties daarin opslaan. Dit is al een betere oplossing, maar deze heeft ook wel wat nadelen. Er is namelijk geen enkele controle mogelijk of de kolommen van de databank wel kloppen en nog belangrijker, of de tabel wel bestaat. Ook is er nu niet echt sprake van abstractie, de gebruiker moet zich aanpassen aan de *library*. Bovendien moet de gebruiker kennis hebben over het gebruik van een databank, wil hij de *library* gebruiken.
- *Hook*mechanisme: Dit is het laatste en door ons gekozen alternatief. De CoAP-*library* definieert hierbij 3 hooks die een gebruiker moet implementeren. Één van de voordelen hiervan is dat de Drupalontwikkelaar normaal gezien al bekend is met het *hook*mechanisme van Drupal zelf. Het gebruikte mechanisme is namelijk analoog, de *hook* wordt geïmplementeerd door het woord *hook* te vervangen door de naam van de module. Deze moeten geïmplementeerd worden maar de functies zelf mogen leeg zijn. We sommen even de te implementeren *hooks* op:
 - *hook_receive_notification*: Deze *hook* wordt opgeroepen telkens wanneer een notificatie binnen komt. De CoAP-*library* bouwt een response-object op van de klasse CoAPMessage en geeft deze mee als argument met de *hook*. Zo kan de gebruiker zelf beslissen wat er met het antwoord moet gebeuren. Men kan zo spreken van een hoge mate van abstractie.

- *hook_receive_error*: Wanneer een fout gebeurt, bijvoorbeeld een sensor die laat of zelfs niet meer reageert, wordt deze *hook* opgeroepen. Als argumenten worden de foutboodschap, het IPv6-adres en *Uri-path* van de resource meegegeven. Alweer kan de gebruiker van de *library* zelf beslissen wat er met deze foutboodschap moet gebeuren.
- *hook_stop_observers*: Wanneer een *observe* op een resource stopt, om welke reden dan ook, wordt deze *hook* opgeroepen. Het kan hier gaan om het uitblijven van antwoorden, een *socket* die sluit, enzoverder. Dit biedt een gebruiker van de *library* de kans om de consistentie van de eigen module te behouden en fouten te vermijden. Men kan bijvoorbeeld melden op een eventueel gemaakte website dat de *observe* gestopt is. Als parameters worden het IPv6-adres en het *Uri-path* van de resource meegegeven zodat de gebruiker weet over welke resource het gaat.

We merken verder nog op dat de CoAP-*library* zelf instaat voor het bijhouden van de te observeren resources. Bovendien wordt om de resource zo weinig mogelijk te belasten, slechts één *observe* uitgevoerd op de resource, ongeacht hoeveel gebruikers erin geïnteresseerd zijn. Een *observe* wordt pas gestopt wanneer geen enkele gebruiker meer geïnteresseerd is in de resource.

4.2.4 Opvangen van verloren berichten

Omdat CoAP gebruik maakt van UDP kunnen pakketten verloren gaan. Dit moet dus opgevangen worden indien CON-berichten verstuurd worden. In de CoAP-*draft* [11] wordt *exponential back-off* voorgeschreven. In de CoAP-*library* werd die geïmplementeerd met de standaardwaarden die eveneens in de CoAP-*draft* [11] staan. Het principe is dat wanneer geen ACK ontvangen wordt als antwoord op het CON-bericht, het CON-bericht opnieuw verstuurd wordt met daartussen exponentieel stijgende perioden tot een ACK ontvangen wordt of een maximum aantal verzendingen overschreden wordt. Hiervoor moeten 2 dingen worden bijgehouden: een time-out waarde en een *retransmission counter* die bijhoudt hoeveel keer het CON-bericht al opnieuw werd verstuurd. De time-outwaarde wordt geïnitieerd op een random waarde tussen ACK_TIMEOUT en ACK_TIMEOUT*ACK_RANDOM_FACTOR, ACK_TIMEOUT en ACK_RANDOM_FACTOR zijn constanten. De *retransmission counter* wordt initieel ingesteld op 0. Wanneer de time-out overschreden wordt en de *retransmission counter* nog steeds een waarde bevat die kleiner is dan MAX_RETRANSMIT (nog een constante), wordt het CON-bericht opnieuw verzonden en de time-outwaarde verdubbeld. De constanten werden in de

CoAP-*library* ingevuld met volgende waarden die voorgeschreven worden in de CoAP-*draft* [11]:

- *ACK_TIMEOUT*: 2 seconden
- *ACK_RANDOM_FACTOR*: 1,5
- *MAX_RETRANSMIT*: 4

Hoofdstuk 5

CoAP-sensormodule

Nu we genoeg kennis hebben over Drupal en CoAP en een *CoAP-library* ter beschikking hebben, kunnen we overgaan tot de uiteindelijk te ontwikkelen module. Het is deze module die een eindgebruiker zal installeren. Ze biedt dan de mogelijkheid om op een gebruiksvriendelijke en dynamische manier sensoren te bekijken en beheren van op de Drupal-website, en dit zonder technische kennis nodig te hebben van CoAP of programmatie in PHP voor Drupal. We bekijken eerst globaal wat de module concreet moet aanbieden. Daarna bekijken we hoe de module zal opgebouwd worden en welke de principes zijn waar we veel belang aan moeten hechten bij de ontwikkeling. Eens we dit weten, kunnen we overgaan tot de verschillende versies van de ontwikkelde module en de implementatie ervan. Verder bespreken we de ingevoerde databank-tabellen en de nieuwe contenttypes. Als laatste onderdeel bespreken we de implementatie van de *CoAP-libraryhooks*.

5.1 Functionaliteit

Het doel van deze module is de eindgebruiker in staat te stellen om gemakkelijk sensoren te beheren op zijn/haar website. Belangrijk hierbij is dat elke Drupal-gebruiker die een website maakt, deze kan gebruiken zonder extra kennis nodig te hebben over CoAP. Ook is het niet de bedoeling dat de eindgebruiker nog moet programmeren om de module te kunnen gebruiken. Kortom, de module moet out-of-the-box werken en zoveel mogelijk omvatten wat mogelijk is met CoAP-sensoren.

Wanneer de gebruiker een resource of een *device* wil toevoegen, hoeft die enkel de URI van

de resource of het IP-adres van het *device* op te geven. Bij het toevoegen van een resource worden de gegevens van die resource automatisch opgehaald. En bij het toevoegen van een *device*, worden de resources die zich in de well-known/core van het *device* bevinden automatisch opgehaald. De gebruiker hoeft geen idee te hebben over hoe dit gebeurt.

Eens de gebruiker de gewenste resources heeft toegevoegd aan zijn/haar website, is het de bedoeling dat de 4 REST-methodes GET, PUT, POST, DELETE kunnen uitgevoerd worden. De gebruiker zal ervan op de hoogte gebracht worden wanneer een methode niet ondersteund wordt. Naast deze methodes kan een resource ook nog observable zijn (zie hoofdstuk 3), de gebruiker moet dus ook in staat gesteld worden om notificaties te ontvangen van een resource. Ook wanneer een gebruiker de website verlaat moet het mogelijk zijn de *observe* te laten doorlopen.

Een gebruiker moet ook gepersonaliseerde lijsten kunnen samenstellen van verschillende resources en/of *devices*.

5.2 Architectuur

In deze paragraaf bespreken we de architectuur van de module en hoe die inpast in het Drupal-systeem. We behandelen tevens ook enkele belangrijke aspecten die in rekening werden gebracht.

5.2.1 Requestproces

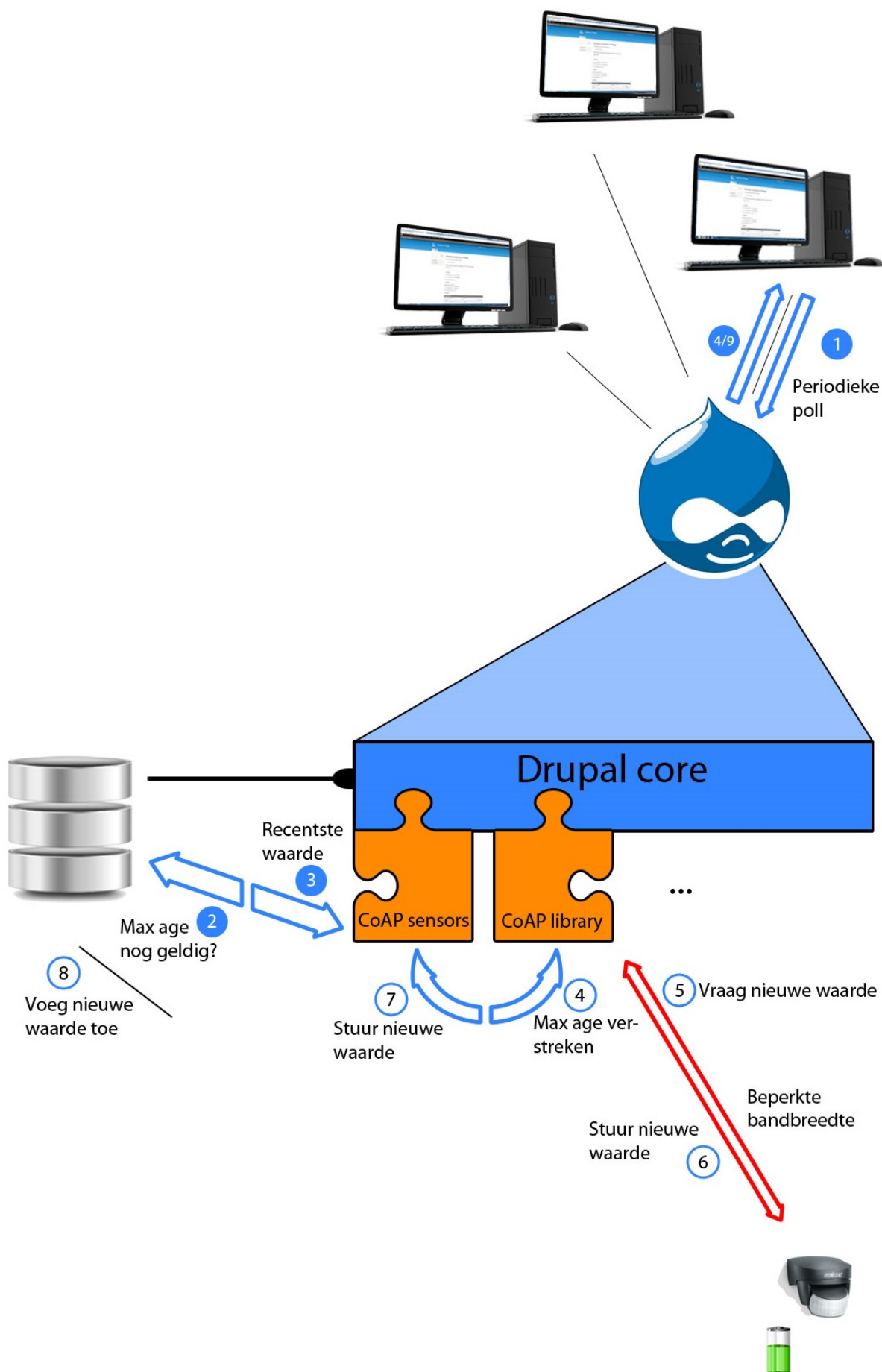
Het proces voor het opvragen van een resourcewaarde wordt weergegeven in figuur 5.1. De acties aangeduid met een gekleurd bolletje gebeuren altijd bij een *request*, de acties aangeduid met een leeg bolletje zullen pas gebeuren wanneer geen geldige waarde uit de databank kan worden gehaald (Zie paragraaf 5.2.2). We overlopen de opeenvolgende stappen:

1. De browser van de *client* start een *request* naar de webserver waarop Drupal draait. Deze poll gebeurt onder vorm van een AJAX *call*.
2. De CoAP-sensormodule voert een query uit op de databank om een geldige waarde op te vragen indien die aanwezig is.
3. De databank stuurt een resultaten set terug die een geldige waarde bevat, of leeg is indien er geen geldige waarde meer is.
4. In deze stap zijn er twee mogelijkheden:

- Indien er een geldige waarde aanwezig is in de databank, wordt deze teruggestuurd naar de client als antwoord op de AJAX call. De *request* eindigt dan hier.
- Indien er geen geldige waarde aanwezig is in de databank, wordt een beroep gedaan op de *CoAP-library* module om een nieuwe waarde op te vragen aan de resource.

Dit mechanisme zorgt ervoor dat de sensor niet onnodig belast wordt (Zie paragraaf 5.2.2).

5. De *CoAP-library* module stuurt een *CoAP-GET-request* naar de resource.
6. De resource stuurt zijn waarde terug in een responsebericht als antwoord op de *GET-request*.
7. De *CoAP-library* geeft een response-object terug en roept bovendien wordt de *CoAP-libraryhook* `hook_recieve_response` opgeroepen.
8. De *hook* zal ervoor zorgen dat de nieuwe, ontvangen waarde opgeslagen wordt in de databank.
9. Uiteindelijk wordt de waarde ook teruggestuurd naar de client die de *request* heeft gestart.



Figuur 5.1: Resourcewaarde opvragen met de CoAP-sensormodule in Drupal

5.2.2 Belangrijke aspecten

Deze module werd ontworpen met het oog op bepaalde aspecten waarmee rekening moet worden gehouden. We sommen enkele van de belangrijkste op.

Modulariteit

Een van de grote troeven van Drupal is de modulariteit die aangeboden wordt. Het is dan ook van uiterst belang dat onze module dit principe niet verbreekt, maar juist doorzet. In hoofdstuk 4 zagen we al hoe een aparte CoAP-*library* ontwikkeld werd. Dit heeft als gevolg dat elke Drupal-gebruiker de kans heeft gebruik te maken van deze *library* en dus zelf een module zou kunnen bouwen gelijkaardig aan de onze.

De CoAP-sensormodule maakt nog gebruik van enkele andere modules welke eerst moeten geïnstalleerd worden door de gebruiker vooraleer hij gebruik kan maken van de CoAP-sensormodule. Drupal zal bij installatie van de CoAP-sensormodule automatisch aangeven welke modules eerst moeten worden geïnstalleerd. Het betreft de volgende modules:

- *Background Process* [20]: Deze module maakt het gebruik van achtergrondprocessen mogelijk. (bijvoorbeeld voor *observe*)
- *Progress* [25]: Deze module moet worden toegevoegd omdat de *Background Process* module hiervan afhankelijk is.
- *CoAP-library* (Zie hoofdstuk 4): Onze eigen geschreven module die instaat voor alle CoAP-communicatie.
- *Entity API* [22]: Een veelgebruikte module (die opgenomen wordt in de core van Drupal 8). Breidt de functionaliteit uit die met entities kan verwezenlijkt worden.
- *Node reference* [26]: Een module die gebruikt wordt om in een bepaalde node door middel van een veld te verwijzen naar andere nodes.

Er zijn ook modules die niet strikt noodzakelijk zijn maar wel extra functionaliteit bieden:

- *Views*[27]: Maakt het gebruik van views mogelijk.
- *Views UI*: Stelt een *user interface* ter beschikking om eigen views te maken en te beheren. Deze module is onderdeel van de viewsmodule.

Minimalisatie van netwerkbelasting

Mogelijks zijn meerdere gebruikers van een Drupal-website geïnteresseerd in een observable resource. Het is niet aanvaardbaar dat de netwerkbelasting zou stijgen met het aantal geïnteresseerden voor die resource. Daarom gedraagt Drupal zich eigenlijk als intermediaire server voor de eindgebruikers. Er wordt namelijk maar één *observe* op die resource uitgevoerd door Drupal, ongeacht het aantal gebruikers dat geïnteresseerd is in die resource. De notificaties worden op de server in de databank gestopt en het zijn deze waarden waarop de gebruikers uiteindelijk pollen.

Asynchroniteit

In een eerste fase van deze module werd er gebruik gemaakt van een formulier. Bij het indienen van dit formulier, werd eerst de *request* om de waarde op te halen uitgevoerd en dan gewacht op de response om de nieuwe pagina op te bouwen. Het spreekt voor zich dat dit geen goede oplossing is, omdat het renderen van de pagina maar kan gebeuren nadat de response aangekomen en verwerkt is. Zo kan de gebruiker de indruk krijgen dat de verbinding met de website verbroken is wanneer de communicatie traag is of misloopt.

In hoofdstuk 2 zagen we als mogelijke oplossing het gebruik van jQuery. In deze situatie is jQuery echter geen goede optie omdat jQuery de databank van Drupal niet zonder bijkomende informatie kan manipuleren. Hiervoor zouden een connectiestring, wachtwoord en dergelijke gegevens nodig zijn. JQuery van die informatie voorzien is een grote bedreiging voor de veiligheid. Bovendien zou dit een oplossing zijn met een zeer sterke koppeling, wat nooit een goed idee is. De gebruikte oplossing illustreert alweer de voordelen van een open source platform met een uitgebreide community. Er is namelijk al een *Background Process* module gemaakt, het zou dus zonde zijn om hier niet dankbaar gebruik van te maken. Zoals de naam suggereert, biedt de *Background Process* module de mogelijkheid om achtergrondprocessen op te starten. Deze draaien op de achtergrond op de server en storen dus geen andere processen zoals het opbouwen van een pagina voor de gebruiker. De gebruiker wordt dus niet meer geconfronteerd met lange wachttijden.

Sterker nog, er is ook een functie voorzien om een HTTP-GET-*request* uit te voeren, waarbij je een *callback*-functie opgeeft. Wanneer er een response is, zal dus automatisch de opgegeven functie als achtergrondproces opgeroepen worden met de response als argument. Deze functie zal dus goed van pas komen in de eerste versie van de module, die gebruik maakt van een

HTTP/CoAP-proxy. Deze wordt beschreven later in dit hoofdstuk (Zie paragraaf 5.3.1).

Strategie bij ontvangst van waarden

Push-strategie: De mooiste oplossing en tevens degene met het minste aandeel aan overhead, is een oplossing waarbij de server zelf op eigen initiatief data kan sturen naar de client. Merk op dat we hier als server de Drupal-server bedoelen en als client de browser. Hierbij is dan geen *polling*mechanisme nodig door de client, wat de netwerkbelasting drastisch verlaagt en de verantwoordelijkheid verschuift naar de server.

Om een *push*-strategie te verwezenlijken werd een uitgebreide literatuurstudie van node.js uitgevoerd. Node.js is een JavaScript *library* die je in staat stelt bi-directioneel verkeer te verwezenlijken. Hierbij wordt gebruik gemaakt van kanalen die worden opgezet, zo'n kanaal kan dan door beide partijen gebruikt worden.

Concreet zou het in de context van deze masterproef mogelijk zijn om met node.js een JavaScript-functie op te roepen bij de client op initiatief van de server.

Er is meermaals gepoogd dit te realiseren, maar de summiere documentatie van node.js laat op sommige vlakken te wensen over. Zo zijn we er niet in geslaagd documentatie te vinden over het opzetten van een eigen kanaal of een bestaand kanaal te gebruiken.

Bovendien is het nodig voor *websockets*, de onderliggende technologie, om een extra server te draaien waarlangs het verkeer moet passeren. Aangezien het vaak niet mogelijk is om de shell van de webserver te gebruiken in een shared-hosting omgeving, is dit een erg groot nadeel. Er bestaan wel servers die je kan gebruiken, maar dit tegen betaling. Wij, als ontwikkelaar van de module, kunnen niet verwachten dat een eindgebruiker een extra server ter beschikking heeft of dat zelfs wil. Het is de bedoeling dat onze module zo veel mogelijk out-of-the-box bruikbaar is.

De conclusie is dat wij geopteerd hebben geen gebruik te maken van node.js en dus ook niet van een *push*-strategie. Er zijn echter wel nog pull-mechanismen die het push-mechanisme simuleren die het vernoemen waard zijn, zoals long polling [41], SSE [28], ...

Pull-strategie: Aangezien een *push*-strategie niet of moeilijk kan gebruikt worden, hebben wij gekozen voor een *pull*-oplossing.

Uiteraard was de eerste reactie gebruik te maken van de Drupal-community en dus te zoeken in de vele modules die beschikbaar zijn. Er is tot op heden geen module geschreven door iemand anders in de Drupal-community dat ons probleem behandelt. De inspiratie voor de uiteindelijke oplossing werd wel gehaald uit een bestaande module, namelijk de *Block Refresh*-module [21]. Deze laatste maakt gebruik van jQuery en *AJAX calls* om periodiek de inhoud van een block te refreshen, waarbij je zelf de lengte van de periode kan bepalen. In jQuery loopt een timer die periodiek een JavaScript-functie oproept. In deze functie wordt dan een *AJAX call* uitgevoerd naar de Drupal-server, die op zijn beurt een antwoord terugstuurt.

Wij hebben geopteerd het mechanisme licht te wijzigen in plaats van de module zelf te gebruiken. Aan de basis van deze beslissing liggen drie redenen:

- Wij wensen de content slechts aan te passen wanneer nodig. Dit wil zeggen, wanneer een nieuwe waarde is binnengekomen. Bovendien hoeft niet alle content vernieuwd te worden, dit doet de *Block Refresh* module wel.
- De configuratie van de periode voor het vernieuwingsinterval is in onze ogen relatief omslachtig. De gebruiker moet al een configuratievenster openen om het interval te wijzigen, wat wachttijden met zich meebrengt. Het zal blijken dat in onze uiteindelijke module het interval snel en gemakkelijk te wijzigen is.
- De uiteindelijke module zal geen blocks meer gebruiken, maar contenttypes. Dit maakt het gebruik van de *Block Refresh* module zelfs onmogelijk.

De aangepaste methode werkt als volgt: Wanneer de pagina geladen wordt bij de client, start een timer die periodiek een *AJAX call* uitvoert naar de Drupal-server. De gebruiker kan hierbij bepalen hoe lang die periode moet zijn. Op de Drupal-server is dan een *AJAX callback* functie gedefinieerd aan de hand van de *hook* `hook_menu()`. Deze *hook* wordt gebruikt om menu items toe te voegen aan de site en om *AJAX callbacks* te definiëren. Alle output die gegenereerd wordt in deze *callback* wordt als antwoord teruggestuurd op de *AJAX callback*.

Wat de *callback* terugstuurt en hoe die inhoud wordt verwerkt, verschilt per versie van onze module. Dit zal dan ook besproken worden in de gepaste paragrafen (Zie paragraaf 5.3).

Caching

Een bericht kan een maximum levensduur hebben, dit wordt aangegeven met een *max-age* optie (optie 14) in het bericht. Deze bevat dan een numerieke waarde die aangeeft in seconden hoelang de waarde als geldig mag worden beschouwd. Wanneer deze maximum levensduur overschreden wordt, mag deze waarde niet meer worden gebruikt. Dan moet bij een volgende aanvraag door een gebruiker een nieuwe waarde worden opgehaald van de resource. Bijgevolg wordt een opgehaalde waarde in de databank gestopt. Wanneer de maximum levensduur nog niet overschreden is bij een aanvraag, wordt de waarde uit de databank gehaald. De resource en het netwerk worden dan dus niet onnodig belast. Dit zorgt er ook voor dat de belasting op een resource niet evenredig stijgt met het aantal gebruikers die er een waarde van willen opvragen. Dit aspect is zeker een van de meer belangrijke omdat verbindingen naar een resource vaak een beperkte bandbreedte kennen. Bovendien bestaat de energievoorziening van sensoren vaak uit een batterij. Het werk dat zo'n sensor moet verrichten, moet dus worden geminimaliseerd.

5.3 Evolutie van de module

5.3.1 Temperatuurmodule met HTTP/CoAP-proxy

In deze paragraaf wordt besproken hoe de eerste versie van de module werd gemaakt die gebruik maakt van een HTTP/CoAP-proxy. Bovendien wordt ook aangetoond hoe waarden opgeslagen worden in de databank om de geschiedenis van opvragingen bij te houden. We spreken hier (en in enkele volgende paragrafen) beter van een temperatuurmodule, aangezien deze module slechts één resource bevraagt. Het betreft een resource die de waarde van een temperatuursensor terugstuurt. Deze resource werd gekozen vanwege de numerieke waarde die teruggestuurd wordt en omwille van het feit dat de resource observable is.

De module wordt aan de gebruiker gepresenteerd onder de vorm van een Drupal-block. Dit block bevat een HTML-formulier met twee componenten die als invoer dienen. Een *checkbox* die aanduidt of de gebruiker waarden automatisch wil laten ophalen, en een knop die het formulier indient wanneer de gebruiker erop klikt.

Wanneer de gebruiker op de knop klikt en de *checkbox* heeft aangevinkt, worden waarden automatisch opgehaald met een interval gelijk aan de waarde van de *max-age*. Merk op dat het hier nog niet om een CoAP-*observe* gaat. In deze module werken we met een HTTP/CoAP-proxy. Periodiek wordt een HTTP-GET-*request* naar een HTML-pagina uitgevoerd. Wanneer de response ontvangen wordt, wordt de HTML-pagina geparsed, waarna de nuttige informatie in de databank wordt geplaatst. Met jQuery worden dan nieuwe waarden periodiek opgehaald en getoond aan de gebruiker.

Er is bij deze versie nog geen sprake van achtergrondprocessen waardoor het lang kan duren eer een pagina geladen wordt. Bovendien krijgt de gebruiker enkel een foutmelding (van de gebruikte webserver) te zien wanneer er geen communicatie mogelijk is.

Demo sensor

Temperatuur

Automatisch ophalen is momenteel niet actief

Temperatuur:
20.2

Max age:
30

Hid	Temperatuur	Max age	Timestamp
77	20.2	30	22/03/2013 11:36:02
76	20.2	30	22/03/2013 11:35:32
75	22.9	30	13/03/2013 13:40:55
74	24	30	13/03/2013 13:40:24
73	24.2	30	13/03/2013 13:39:53

☐ Automatisch ophalen

Figuur 5.2: *Temperatuurmodule met HTTP/COAP-proxy*

Proxy

Als proxy werd de coap.me-website [10] gebruikt. Deze biedt de mogelijkheid om notificaties van een *embedded device* te bekijken in een browser. De website biedt ook de mogelijkheid om door te klikken naar de notificatie om die volledig te bekijken op een pagina.

Het is deze laatste pagina die periodiek wordt opgehaald en geparsed. De communicatie

vanwege de Drupal-module bestaat dus enkel uit HTTP-communicatie, de proxy verzorgt de nodige CoAP-berichten tussen zichzelf en het *embedded device*.

Polling

In deze eerste versie van de module bestond het antwoord bij de *polling* uit de volgende velden:

- *Hid*: De History ID dat een numeriek ID is voor de opgevraagde waarde. Het is tevens de index in de databanktabel met waarden en is dus uniek.
- *Temperatuur*: Dit is de effectieve waarde in graden Celsius. Deze werd geparsed uit de responsestring afkomstig van de resource.
- *Max-age*: De antwoorden van de betreffende resource bevatten een max-ageoptie. Dit is het aantal seconden dat deze waarde als geldig mag beschouwd worden.
- *Timestamp*: Het tijdstip waarop de waarde ontvangen werd.

5.3.2 Temperatuurmodule met native CoAP

Nu de omliggende structuur opgezet en uitgetest is, kan de communicatie veranderen van HTTP-berichten naar native CoAP. Er is in deze versie echter nog geen sprake van de CoAP-*library*. Deze versie beperkt zich tot een eerste test met CoAP-communicatie.

Voor CoAP-verkeer zal UDP gebruikt worden zoals voorgeschreven in de CoAP-draft [11]. Eerst wordt een UDP-socket geopend naar het betreffende IPv6-adres van het *embedded device* waarop de resource is aangesloten. Dit gebeurt met de functie `psockopen()` van PHP, deze opent een persistente socket. Vervolgens wordt een hexadecimale string opgesteld die het bericht voorstelt. Ook deze opstelling gebeurt aan de hand van de CoAP-draft [11]. Het bericht wordt verstuurd met de `fwrite()` functie. Hierna kan het antwoord opgehaald worden van de socket met de `fread()` functie die als enige parameter een grootte in bytes verwacht. Belangrijk hierbij is dat de opgegeven grootte minstens even groot is als het aantal bytes van het volledige antwoord.

Observe

Nu er gebruik gemaakt wordt van native CoAP, is er ook een echte CoAP-*observe* mogelijk. Hierbij is het noodzakelijk dat de socket opengehouden wordt. Dit omdat het *embedded device* nu op eigen initiatief berichten kan sturen. Aangezien de socket moet worden opengehouden, kan de code niet meer op de voorgrond draaien. Indien dit wel zo zou zijn, zou de pagina nooit gerenderd worden. De gebruiker zou dan geconfronteerd worden met een timeout van de webserver waarop Drupal draait. Achtergrondprocessen zijn nu dus geen optie meer, maar een noodzaak. De code om een socket open te houden en te beheren zal dus opgestart worden in een achtergrondproces aan de hand van de *Background Process* module die eerder al vermeld werd [20].

Bij ontvangst van een notificatie van de CoAP-resource zal de waarde in de databank worden gestopt met bijbehorende velden. Deze velden zijn dezelfde als in de vorige versie van de module (Zie paragraaf 5.3.1). Het zijn deze waarden waarop de client met jQuery zal pollen. Ook deze *polling* gebeurt op dezelfde manier als in de vorige versie van de module.

5.3.3 CoAP-sensormodule met externe CoAP-library

Alle code die te maken had met CoAP-communicatie was hardgecodeerde code wat onvermijdelijk leidde tot duplicatie van code. Bovendien bevond deze code zich in dezelfde module als diegene die nu wordt besproken. Dit heeft als gevolg dat andere gebruikers van Drupal geen gebruik kunnen maken van onze code om CoAP-berichten te sturen en te ontvangen. Dit alles heeft geleid tot de ontwikkeling van onze eigen CoAP-*library* in PHP onder de vorm van een externe module die apart kan worden gebruikt. Deze *library* werd eerder al besproken in hoofdstuk 4.

Nieuwe block

De module onderging in deze stap naast een visuele verbetering, ook een verbetering van functionaliteit (Zie figuur 5.3). Zo kan de gebruiker nu met één block meerdere resources bevragen.

De gebruiker kan één resource selecteren met *radiobuttons*, wanneer de gebruiker dan op de knop 'Bekijken' klikt, wordt de pagina herladen en is de geselecteerde resource te bevragen. Door op de knop 'GET' te klikken, wordt een GET-*request* uitgevoerd op de geselecteerde resource.

Er wordt ook een lijst met *checkboxes* ter beschikking gesteld. De gebruiker kan hierbij de resources aanvinken waarop een *observe* moet worden uitgevoerd. Wanneer de gebruiker een aangevinkte *checkbox* uitvinkt, zal de *observe* stoppen. De instellingen worden opgeslagen en de benodigde acties worden uitgevoerd wanneer de gebruiker op de knop 'Observe' klikt. Belangrijk hierbij is dat wanneer de pagina herladen wordt, de juiste *checkboxes* al worden aangevinkt zodat de gebruiker ten allen tijde weet welke resources al geobserveerd worden.

Een laatste verbetering bestaat uit een grafiek die automatisch wordt gegenereerd op basis van de geschiedenis van opvragingen. De grafiek wordt aan de hand van een *AJAX call* en de *Google Charts API* [13] gerealiseerd. Het betreft hier de waarden van de resource die momenteel bekeken wordt en dus aangeduid is met een *radiobutton*.

5.3.4 CoAP-sensormodule met contenttype CoAP-resource

In de vorige versies werd de module weergegeven aan de hand van een HTML-formulier in een Drupal-block. Het nadeel hiervan is dat men standaard slechts één instantie van het block gebruikt. Men kan echter wel het een en het ander verbeteren met de *Views*module, maar het is niet de bedoeling dat wij dit verlangen van de gebruiker. Bovendien is een block slechts een visuele blok op de website, het is niet echt een node of een stuk content.

Een beter en vaak gebruikt alternatief is het maken van een eigen contenttype. Wanneer men nu deze module installeert worden de contenttypes gedefinieerd in de module, mee geïnstalleerd. Dit heeft als gevolg dat de gebruiker slechts op 'Add content' hoeft te klikken (Zie figuur 5.7), enkele benodigde velden in moet vullen en de content wordt automatisch gegenereerd en opgeslagen. Bij deze methode wordt al deze configuratie afgehandeld door Drupal-mechanismen, zo worden bijvoorbeeld de velden opgeslagen in tabellen die behoren tot de Drupalcore. Dit betekent dat onze content effectief wordt ingepast en geen losstaand geheel is. Het beheer van de content, manipulatie en dergelijke zal dus gebeuren met Drupal zoals dat gebeurt voor andere contenttypes. We merken ook op dat wanneer een resource verwijderd wordt, dit niet het geval is voor de waarden die ervan opgehaald zijn. Deze blijven aanwezig in de databank en zullen ook weer te zien zijn wanneer de resource later opnieuw wordt toegevoegd.

Sensoren bevragen met het CoAP protocol



Message: B

Geef een URI

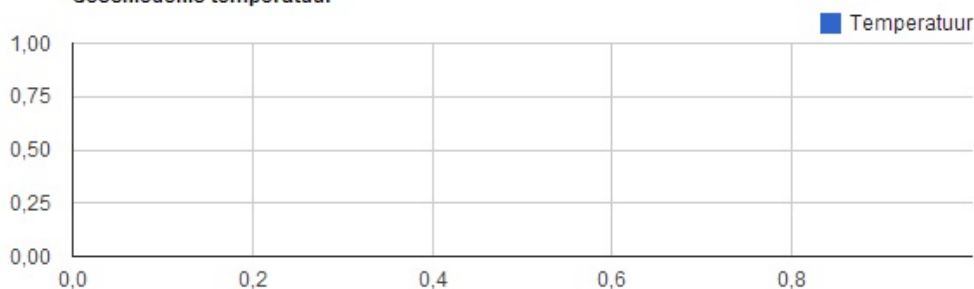
Invoeren

☐ 2001:6a8:1d80:200::2/obs☐ 2001:6a8:1d80:200::2/test

Observe

Hid	Value	Max age	Timestamp
-----	-------	---------	-----------

Geschiedenis temperatuur



Selecteer een resource

☒ 2001:6a8:1d80:200::2/obs☐ 2001:6a8:1d80:200::2/test

Bekijken

Response:

GET

Figuur 5.3: Block met HTML-formulieren die de gebruiker toelaat meerdere resources met één block te bevragen

5.3.5 CoAP-sensormodule met volledige REST-functionaliteit

Tot nog toe was enkel een *GET-request* mogelijk naar een resource, maar vaak ondersteunt een resource ook nog andere REST methodes (GET, PUT, POST, DELETE). Het is dus van groot belang dat onze module ook de mogelijkheid biedt om alle REST methodes uit te voeren.

Dit brengt ook mee dat de visuele representatie zal wijzigen (Zie figuur 5.4). De vier methodes worden met knoppen weergegeven in een tabel, eventueel vergezeld van een tekstveld indien invoer vereist is (PUT en POST). Onderaan de tabel wordt de response en response method (205 Content, 405 Bad method,...) getoond.

Wij hebben ervoor gekozen dat de gebruiker voor elke resource, op elk moment elke REST methode kan uitvoeren. Er wordt namelijk al een gepaste response getoond wanneer een methode niet ondersteund is. Bovendien is een server niet verplicht in een resource *discovery* (Zie paragraaf 3.3.1) aan te geven welke methodes ondersteund zijn. Wil men voor elke resource toch methodes uitschakelen die niet ondersteund zijn, dan zou men elke methode eens moeten uitproberen en verifiëren of een 405 Bad Request teruggestuurd wordt.

Resource URI:

2001:6a8:1d80:200::2/validate

Polling gebeurt om de seconden. [Wijzigen](#)

[Start observing](#)

Request methods			
GET	<input type="text" value="input"/> <input type="button" value="PUT"/>	<input type="text" value="input"/> <input type="button" value="POST"/>	<input type="button" value="DELETE"/>
Response:			
Response type:			

Hid	Value	Max age	Timestamp
/			
/			
/			
/			
/			

Figuur 5.4: CoAP-resource met volledige REST functionaliteit

5.3.6 CoAP-sensormodule met resource discovery

De gebruiker is nu in staat meerdere CoAP-resources toe te voegen aan zijn/haar website. Het kan echter moeilijk zijn een overzicht te behouden over al deze resources en bovendien moet men van elke resource de specifieke URI kennen, wat zeker niet vanzelfsprekend is.

Deze nadelen kunnen worden weggewerkt door gebruik te maken van het concept resource *discovery*. De technische uitleg en implementatiedetails werden al besproken in paragraaf 3.3.1. In deze module krijgt de gebruiker na het uitvoeren van een resource *discovery*, een lijst van CoAP-resources te zien die aangesloten zijn op het *embedded device*. Om performantieredenen en omwille van het feit dat een well-known/core weinig of nooit verandert, wordt het resultaat van de resource *discovery* (Zie paragraaf 3.3.1) opgeslagen in de databank. De manier waarop de gegevens opgeslagen zijn is dezelfde als in het eindresultaat en wordt besproken in 5.4. Er wordt echter op de website wel nog een Refresh-knop voorzien mocht de gebruiker de gegevens in de databank toch willen vernieuwen.

Op de website wordt een resource weergegeven door de URI en door de *human readable name*. Bovendien is voor elke resource ook een *checkbox* voorzien om aan te duiden dat de gebruiker geïnteresseerd is in de resource. De gekozen resources worden dan opgesomd en ter beschikking gesteld op dezelfde manier als in de vorige versie van de module (Zie figuur 5.3). In de volgende twee subparagrafen bespreken we twee mogelijke vormen die wij gemaakt hebben om de resource *discovery* en opsomming van de resources te combineren.

Multi-form

Een eerste mogelijkheid bestaat erin eerst de lijst te presenteren aan de gebruiker. De gebruiker kiest dan een aantal resources die hij/zij wil toevoegen aan de website, waarna hij/zij op de knop 'Save' klikt. Hierna wordt een nieuwe pagina getoond die de gebruiker toelaat de resources te beheren en te bevragen (Zie figuur 5.3). De gebruiker kan altijd terugkeren naar het *discovery*-formulier om de resource *discovery* opnieuw uit te voeren of resources te (de)selecteren. Dit gebeurt wanneer hij op een knop met opschrift 'Back' onderaan het formulier klikt. Wanneer op Save of Back geklikt wordt, wordt het formulier ingediend. In de submit handlers van de knoppen worden gegevens verwerkt en wordt het juiste formulier getoond.

Resources | **Discovery** | Configuration

Device IPv6 address *

Load device | Refresh device | Clear

	uri	human-readable name
<input type="checkbox"/>	2001:6a8:1d80:200::2/create2	Create2 resource
<input type="checkbox"/>	2001:6a8:1d80:200::2/large-update	Large resource that can be updated using PUT
<input type="checkbox"/>	2001:6a8:1d80:200::2/obs	Observable resource that changes every 5 seconds.
<input type="checkbox"/>	2001:6a8:1d80:200::2/t	
<input type="checkbox"/>	2001:6a8:1d80:200::2/test	Default test resource
<input type="checkbox"/>	2001:6a8:1d80:200::2/validate	ETag test resource

Save

Figuur 5.5: Opsomming van resources aan de hand van resource discovery

Tabbladen

De tweede mogelijkheid bestaat uit een pagina met tabbladen. Op deze manier kan de gebruiker gemakkelijker navigeren tussen de verschillende formulieren en hoeven deze niet steeds ingediend te worden. Er zijn 3 tabbladen voorzien (Zie figuur 5.5):

- **Resources:** Dit tabblad bestaat uit het formulier dat de gebruiker in staat stelt de resources te beheren en te bevragen zoals in de vorige versie van de module (Zie figuur 5.3).
- **Discovery:** Dit tabblad toont de lijst van resources met bijbehorende *checkbox* (Zie figuur 5.5).
- **Configuration:** Dit tabblad is voorzien voor een configuratie die door de gebruiker kan worden ingesteld. Deze is echter niet geïmplementeerd in deze module, maar kan wel toegevoegd worden in verder ontwerp (Zie paragraaf 6.6).

5.3.7 CoAP-sensormodule met meerdere contenttypes

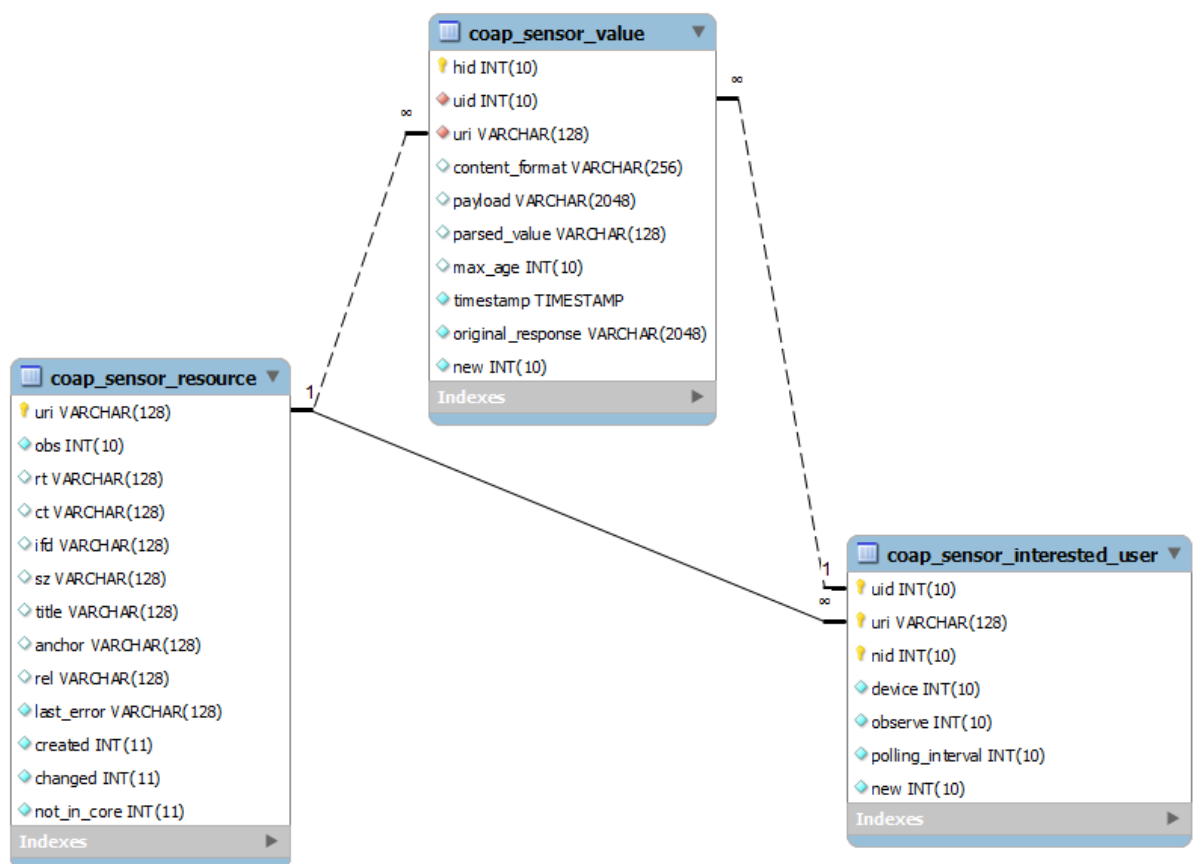
Dit is de, voor ons laatste, versie van de module die uitgebreid in het restant van dit hoofdstuk besproken wordt. Hier wordt nog een groot nadeel dat de kop opsteekt weggewerkt. De gebruiker was niet in staat een aparte resource toe te voegen zonder dat de functionaliteit van resource *discovery* erbij kwam. We lossen dit op door meerdere contenttypes in te voeren.

5.4 Contenttype in de databank

Onder contenttype van de databank verstaan we de definitie van nodige databanktabellen en -kolommen. Er moet namelijk heel wat data opgeslagen worden waaronder de verschillende resources, de relatie tussen resources/*devices* en de opgehaalde waarden per gebruiker.

Zoals eerder vermeld worden de tabellen aangemaakt bij installatie van de module. Dit wordt mogelijk gemaakt door gebruik te maken van `hook_schema()`. In deze *hook* wordt het databankschema opgesteld en teruggegeven als array door de functie. Bij deïnstallatie worden de tabellen verwijderd uit de databank. Dit gebeurt door gebruik te maken van `hook_uninstall()` die de functie `db_drop_table()` gebruikt.

Er worden drie tabellen gedefinieerd (Zie figuur 5.6), welke beschreven worden in de komende paragrafen.



Figuur 5.6: EER-diagram van de door ons toegevoegde tabellen

coap_sensor_resource

De tabel bevat informatie over CoAP-resources die toegevoegd werden aan de website. Hij bevat volgende kolommen (Zie figuur 5.6):

- uri: Dit is de volledige URI van de resource, dus IPv6-adres en URI-path. Dit is tevens de primaire sleutel van deze kolom, elke waarde moet dus uniek zijn.
- obs, rt, ct, ifd, sz, title, anchor en rel: Deze kolommen bevatten waarden die bekomen worden bij resource discovery. Deze werden eerder uitgelegd in paragraaf 3.3.1.
- last_error: Deze kolom bevat voor elke resource de laatst opgetreden fout. Zo kan een browser te weten komen bij *polling* dat er iets foutliep bij communicatie met de betreffende resource.
- created: De UNIX timestamp van de tijd waarop deze resource aangemaakt werd.
- changed: De UNIX timestamp van de tijd waarop deze resource het laatst gewijzigd werd.
- not_in_core: Een waarde om aan te geven of deze resource nog in de well-known/core zit van een *device*.

coap_sensor_interested_user

In deze tabel wordt bijgehouden welke gebruikers geïnteresseerd zijn in specifieke resources en *devices*. Hij bevat een entry voor elk koppel gebruiker/resource en gebruiker/*device* en wordt impliciet geassocieerd met een instantie van respectievelijk coap_resource of coap_device. Op die manier kan per user worden bijgehouden wat hij/zij met een bepaalde resource wil doen. Ze bestaat uit de volgende kolommen:

- uid: Een unieke identificatie voor een gebruiker. Dit ID is overgenomen uit Drupal, daar hebben alle gebruikers ook een uniek user ID. Deze kolom is tevens deel van de primaire sleutel.
- uri: De URI van de resource waarin de gebruiker geïnteresseerd is. Dit is dezelfde URI als in de vorige tabel (coap_resource_resources). Deze kolom vormt dus een foreign key en maakt tevens deel uit van de primaire sleutel.
- nid: Een unieke identificatie van de node (node ID) waaraan de resource gekoppeld is in Drupal. Deze kolom is ook deel van de primaire sleutel.

- *device*: Deze waarde geeft aan of het een resource of een volledig *embedded device* betreft (0 = resource, 1 = *device*).
- *observe*: Deze waarde geeft aan of de gebruiker momenteel de resource wil observeren of niet (0 = niet observeren, 1 = wel observeren).
- *polling-interval*: Dit is de periode in seconden tussen twee polls van de browser naar de Drupal-server, deze is default drie seconden.
- *new*: Deze waarde duidt aan of er inmiddels een nieuwe *discovery* is uitgevoerd of bezig is voor een *device*. Deze waarde heeft momenteel enkel nut voor *devices*. (0 = up-to-date, 1 = *discovery* bezig, 2 = *discovery* klaar/referentiegegevens moeten geupdated worden)

De primaire sleutel bestaat uit de combinatie van de kolommen uid, uri en nid. Elke combinatie van waarden uit deze drie kolommen moet per entry uniek zijn.

coap_sensor_value

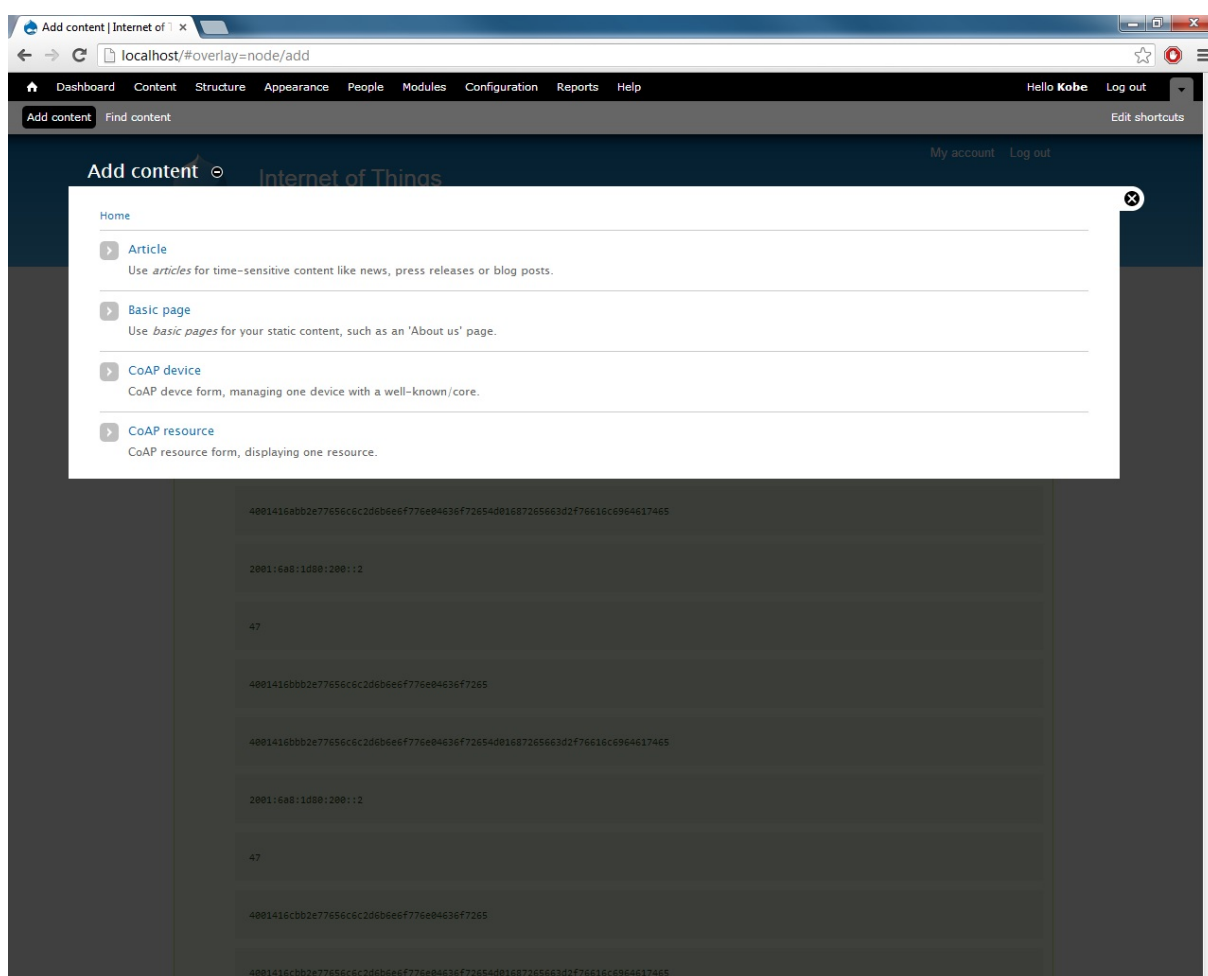
Deze tabel bevat een entry voor elke waarde die is binnengekomen. Bovendien wordt een nieuwe waarde even veel keren toegevoegd als er geïnteresseerden zijn voor die waarde op dat moment. Dit lijkt op het eerste zicht redundante informatie, maar het heeft wel degelijk nut. Zo kan men bijvoorbeeld later te weten komen voor een bepaalde user, welke waarden hij/zij in welke periode heeft opgevraagd. Men zou als alternatief ook relaties kunnen leggen tussen waarden en users, zo hoeven er geen duplicaten van waarden opgeslagen te worden. De tabel bevat de volgende kolommen:

- *hid*: Een uniek ID voor de entry automatisch gegenereerd bij toevoegen aan de databank. Dit is de primaire sleutel, dus deze waarden moeten uniek zijn.
- *uid*: Het ID van de user waarvoor deze waarde opgeslagen is. Dit is hetzelfde ID als in de *coap_resource_users*-tabel en vormt dus een foreign key.
- *uri*: De URI van de resource waarvan deze waarde komt. Dit is dezelfde URI als in de *coap_resource_resources*-tabel en vormt dus een foreign key.
- *content_format*: Bevat voor elke entry het formaat van het bericht, aangeduid met een leesbare naam (bijvoorbeeld plain text, JSON, ...).
- *payload*: De payload van het bericht in leesbare vorm.

- `parsed_value`: Indien mogelijk bevat dit een waarde die geparset is uit de payload, bijvoorbeeld een numerieke waarde. Indien er geen waarde kon geparset worden bevat deze kolom gewoon de payload.
- `max_age`: Geeft aan wat de geldigheidsduur is voor deze waarde, deze is default nul.
- `timestamp`: Het tijdstip waarop de waarde in de databank werd opgeslagen in mysql timestamp-formaat. Deze wordt automatisch gegenereerd. Samen met de *max-age* kan men dus bepalen of de waarde nog geldig is.
- `original_response`: De originele response op het CoAP-*request*bericht in hexadecimale vorm.
- `new`: Geeft aan of de waarde reeds opgehaald werd om te tonen aan de gebruiker (0 = reeds opgehaald, 1 = nog niet opgehaald, default waarde = 0).

5.5 Contenttypes in Drupal

Bij installatie van de module worden twee contenttypes mee geïnstalleerd. We bespreken wat de mogelijkheden ervan zijn en hoe ze eruitzien. Het laten installeren van contenttypes zorgt ervoor dat een Drupal-gebruiker op een eenvoudige manier content kan toevoegen, in dit geval onderdelen van een CoAP-sensornetwerk. Concreet moet de gebruiker enkel op 'Add content' klikken op zijn/haar Drupal-website en het gewenste contenttype aanklikken. Na invullen van enkele vereiste velden wordt een node gecreëerd die de content voorstelt.



Figuur 5.7: Content toevoegen door op 'Add content' te klikken

5.5.1 CoAP-resource

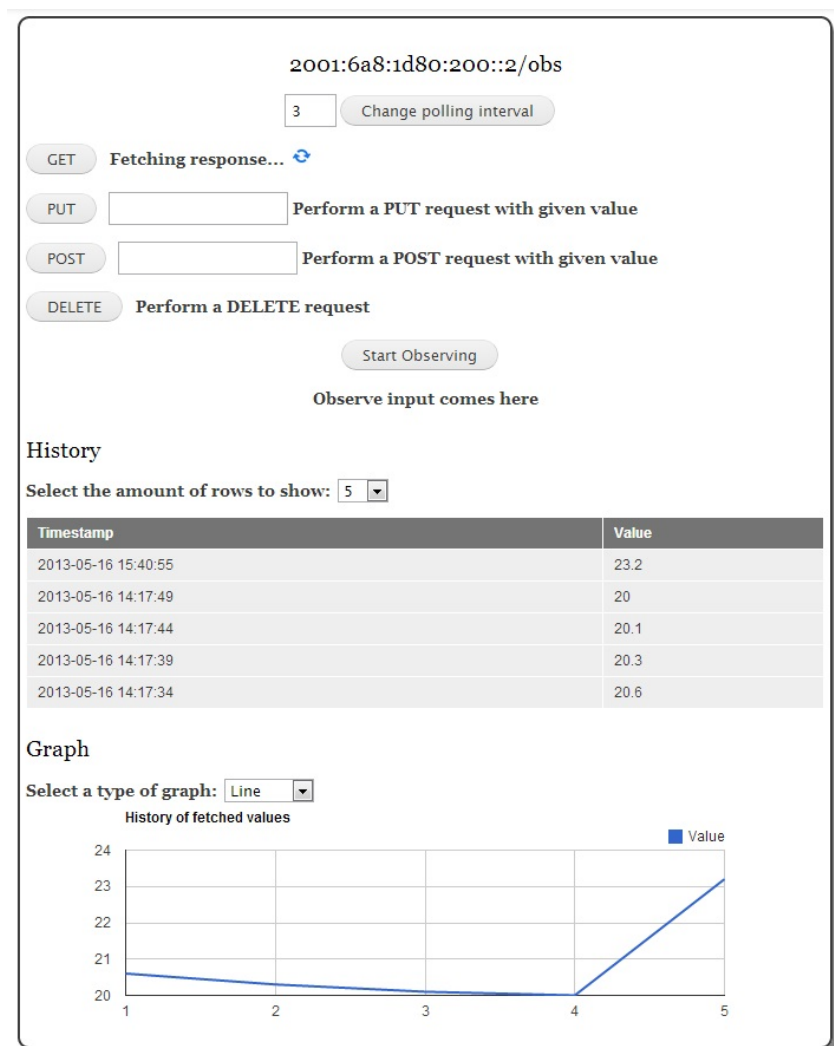
Dit contenttype stelt één resource voor. Bij het toevoegen wordt de URI opgegeven, deze bestaat uit het IPv6-adres van het *embedded device* waarop de resource is aangesloten en een URI-path. IPv6-adres en URI-path worden gescheiden door een slash ('/'). Een voorbeeld van een door ons vaak gebruikte URI is:

coap://2001:6a8:1d80:200::2/test. Vooraleer een resource effectief toegevoegd wordt, wordt hij gevalideerd. We voegen custom validatie toe via `hook_form_alter`. Indien de structuur van de URI niet geldig is of de resource al eerder door de gebruiker is toegevoegd, wordt er een foutmelding getoond en krijgt de gebruiker opnieuw de kans om een geldige/andere URI op te geven. Een andere manier om validatie van een veld te verwezenlijken is het maken van een custom veld met de Field API [5]. Op deze manier is de validatie onderdeel van het veld en hoeft ze niet achteraf toegevoegd te worden, wat gebeurt in `hook_form_alter()`. Deze methode gaf echter problemen indien content van dit type gewijzigd werd door middel van de *Entity API*. Na een succesvolle validatie, wordt er content van het type `coap_resource` toegevoegd. Door middel van `hook_node_insert` worden er enkele dingen in de databank toegevoegd. Er wordt een entry in de tabel `coap_sensor_interested_user` toegevoegd waarmee wordt aangegeven dat de gebruiker die de resource toevoegde geïnteresseerd is in de resource. Er wordt ook gekeken of er voor deze resource informatie beschikbaar is in de tabel `coap_sensor_resource`. Indien dit niet het geval is wordt een partiële resource *discovery* uitgevoerd voor die resource. Door middel van query filtering worden de opgehaalde gegevens toegevoegd aan de databank. Indien er wel informatie beschikbaar is moet er niets meer gebeuren en krijgt de gebruiker de visuele representatie van de content te zien.

De gebruiker krijgt nu de kans om de REST-methodes GET, PUT, POST en DELETE uit te voeren en een *observe*. Bovendien krijgt de gebruiker een geschiedenis van opvragingen voor deze resource te zien en krijgt hij/zij ook de kans een grafiek te laten genereren. Voor het type grafiek heeft de gebruiker de keuze uit een lijn-, staaf-, of taartgrafiek. De gebruiker moet zelf weten welke soort grafiek nuttige informatie kan bevatten. In tegenstelling tot vorige versies

Figuur 5.8: Adding content of contenttype CoAP-resource

wordt de visualisatie van content van het type CoAP-resource verzorgd door een template.



Figuur 5.9: Visuele representatie van het contenttype CoAP-resource

Templating

Drupal voorziet een handig mechanisme wanneer je een template voor een specifiek contenttype wil maken. Het enige wat men moet doen om ervoor te zorgen dat die template automatisch wordt opgeroepen voor het contenttype, is het template-bestand de juiste naam geven. Drupal voorziet namelijk in alle themes een standaard template voor een node, namelijk `node.tpl.php`. Om nu onze template te definiëren, moet deze de naam `node-coap_resource.tpl.php` dragen, daar de naam van ons contenttype `coap_resource` is.

Het voordeel van deze benadering is dat de view gescheiden wordt van de rest van de code.

Wanneer men bepaalde attributen of content nodig heeft voor de visuele representatie, kan men die variabelen voorzien door gebruik te maken van `hook_preprocess_node()`. Deze *hook* wordt op voorhand opgeroepen en variabelen worden klaargezet. In het template-bestand kan men deze variabelen bereiken aan de hand van PHP-scriptlets (Zie listing 5.1). Men kan in deze templates dus ook PHP-code laten uitvoeren, al beperkt men dit best tot een minimum om de view gescheiden te houden. Zo wordt bij ons de URI van de resource en enkele waarden gebruikt in het template-bestand. Voor de rest bevat het template-bestand vooral HTML om de gegevens te visualiseren.

Er stelt zich nu wel nog een probleem, het template-bestand moet namelijk op de juiste plaats staan, en dit in de directory van het gebruikte theme. Men voegt echter beter geen bestanden toe aan de Drupalcore en bovendien kunnen wij dat ook niet verwachten van de eindgebruiker. Het template-bestand zou automatisch moeten worden opgenomen in het gebruikte theme, waarbij het template-bestand gewoon in onze module kan blijven staan. Na grondig onderzoek bleek dit mogelijk aan de hand van `hook_theme_registry_alter()`. Deze *hook* geeft ons de kans een pad toe te voegen aan het theme-registry, waardoor het template zal gevonden worden bij het bouwen van de view [40].

Codevoorbeeld 5.1: Voorbeeld van een template met PHP-scriptlets (node.tpl.php)

```
<div id='node-<?php print $node->nid; ?>' >
  <?php print render($title_prefix); ?>
  <?php if (!$page): ?>
    <h2<?php print $title_attributes; ?>>
      <a href='<?php print $node_url; ?>'>
        <?php print $title; ?>
      </a>
    </h2>
  <?php endif; ?>
  <?php print render($title_suffix); ?>
</div>
```


Polling

In vorige versies van de module had de gebruiker niet de kans om meerdere resources op één pagina te zetten met de *Views*module. Dit was in principe wel mogelijk, maar er zouden fouten optreden daar de *polling* gebeurde op basis van welke resource geselecteerd was. Dit selecteren van resources is in deze versie weggewerkt, elke resource is een op zichzelf staand geheel (Zie figuur 5.9). In deze laatste versie is dit wel mogelijk omdat het *polling* mechanisme grondig werd aangepast zoals nu zal blijken.

Een poll gebeurt nu specifiek voor een URI die opgegeven wordt in de URL waarnaar een *AJAX call* wordt uitgevoerd. Bovendien werd in de databank een extra kolom 'New' toegevoegd aan de waardentabel. Deze duidt aan of een waarde reeds is opgehaald of nog niet (0 = reeds opgehaald, 1 = nog niet opgehaald). De kolom heeft als default waarde 1, dus nieuwe waarden worden automatisch als nieuw aangeduid. Wanneer een waarde opgehaald wordt uit de databank wordt de waarde in de kolom 'New' op 0 gezet.

Het pollen naar waarden voor een specifieke URI gebeurt nu als volgt:

- Er wordt in jQuery een *AJAX call* uitgevoerd naar een bepaalde URL, bijvoorbeeld `http://localhost/coap_resource/poll/2001:6a8:1d80:200::2|test`. In de URI wordt een slash vervangen door een rechte streep ('|'), anders wordt de slash geïnterpreteerd.
- Op de webserver wordt in Drupal de URI uit de aanvraag-URL gehaald en worden de slashes terug geplaatst.
- De URI wordt gebruikt in een select-query om de nog niet opgehaalde waarden op te halen, dit wil zeggen de rijen waarvan de waarde in de kolom 'New' op 1 staat.
- Er wordt een update-query uitgevoerd om van de opgehaalde rijen de waarde in de kolom 'New' te wijzigen naar 0. Dit om aan te geven dat de waarden reeds opgehaald werden.
- De opgehaalde rijen worden in een XML-structuur gegoten samen met identificatie van de resource (de URI) en een eventuele errorstring. Deze XML-structuur wordt na deze stappen toegelicht.
- De XML-structuur wordt uitgeprint en dus teruggestuurd als antwoord op de *AJAX call*.
- in jQuery wordt dan het antwoord geparsed. De errorstring wordt eruit gehaald en eventuele fouten worden getoond aan de gebruiker (Zie paragraaf 5.5.1). Daarna wordt elke

opgehaalde rij één voor één overlopen en toegevoegd aan de visuele content die de gebruiker ziet.

De XML-structuur die eerder werd aangehaald bestaat uit een allesomvattend hoofdelement `<poll>`. Dit element bevat 3 andere soorten elementen:

- `<uri>`: Dit element komt één keer voor, het bevat de URI en dus de identificatie van de resource.
- `<error>`: Net als het vorige element komt dit element ook één keer voor, het bevat de eventuele errorstring. Deze kan één van de volgende zijn: none, delay, broken of unreachable.
- `<entry>`: Dit element stelt een rij voor met een waarde. Dit element kan dus nul of meer keren voorkomen. Het bestaat zelf uit de volgende elementen:
 - `<Hid>`: de history ID van de rij.
 - `<Value>`: de effectieve (eventueel geparse) waarde van de rij.
 - `<Max age>`: Het aantal seconden dat deze waarde als geldig mag worden beschouwd.
 - `<Timestamp>`: Het tijdstip waarop de waarde ontvangen werd op de Drupal-server.

Een voorbeeld van een antwoord op een *AJAX call* bij *polling*:

Codevoorbeeld 5.2: Voorbeeld antwoord op *AJAX call* bij polling

```
<poll>
  <uri>2001:6a8:1d80:200::2</uri>
  <error>none</error>
  <entrys>
    <entry>
      <hid>76</hid>
      <value>20.2</value>
      <max_age>30</max_age>
      <timestamp>2013-05-20 20:36:32</timestamp>
    </entry>
    <entry>
      <hid>77</hid>
```

```

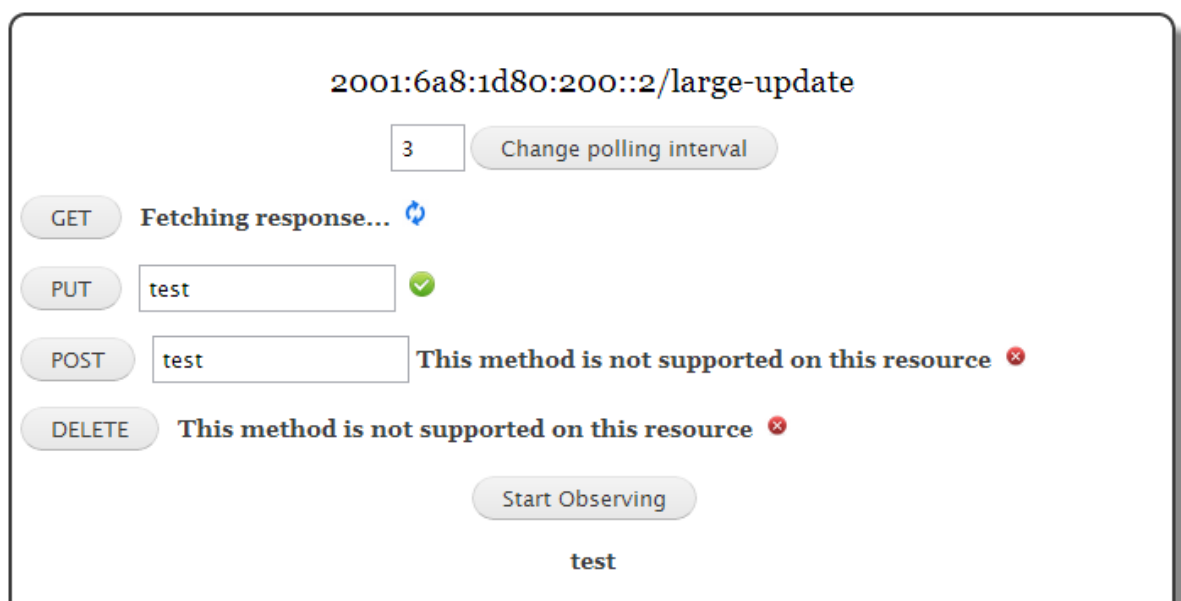
        <value>20.4</value>
        <max_age>30</max_age>
        <timestamp>2013-05-20 20:36:37</timestamp>
    </entry>
</entrys>
</poll>

```

Foutmechanisme

Tot nog toe bleef de gebruiker in het ongewisse wanneer er iets foutliep in de communicatie met een CoAP-resource. Dit is niet gebruiksvriendelijk en kan voor ergernissen zorgen.

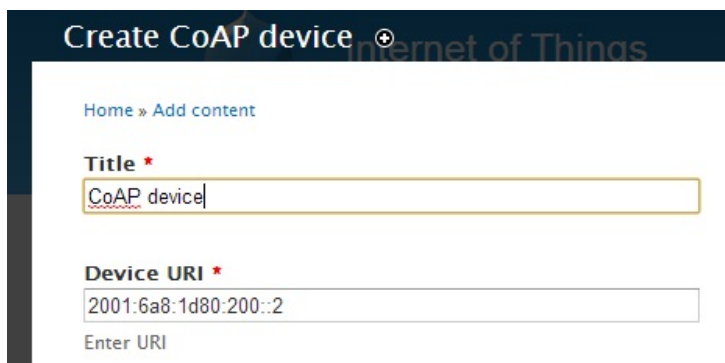
Nu hebben we in de vorige hoofdstukken al gezien dat fouten al geregistreerd worden aan de hand van enkele errorstrings. Deze zijn zelfs al ter beschikking in jQuery bij elke poll (Zie *polling* hierboven). Het enige wat nu nog moet gebeuren is een gepaste boodschap tonen in de visuele representatie. Zoals te zien in figuur 5.10 wordt in het groen een vinkje getoond wanneer een operatie geslaagd is. Wanneer de communicatie niet gelukt of foutgelopen is wordt de methode geannuleerd en wordt een kruisje in het rood getoond. Ook wanneer een methode niet ondersteund is, wordt dit kruisje getoond. Wanneer het antwoord op zich laat wachten worden roterende blauwe pijltjes getoond.



Figuur 5.10: Mechanisme om fouten te tonen aan de gebruiker

5.5.2 CoAP-device

Dit contenttype stelt een volledig *embedded device* voor waarop zich één of meerdere resources bevinden. Bij toevoegen van dit contenttype hoeft de gebruiker enkel het IPv6-adres op te geven in het veld `field_device_uri`. Er is nog een veld aanwezig in dit contenttype, namelijk het veld



Figuur 5.11: Adding content of contenttype CoAP-device

`field_resource_references`. Dit veld is onzichtbaar bij het aanmaken of wijzigen van content zodat een gebruiker dit veld niet kan aanpassen. Het mag enkel vanuit code aangepast worden en bevat referenties naar content van het type CoAP-resource die overeen komt met de resources uit de well-known/core van het *device*.

Net zoals bij CoAP-resource voegen we custom validatie voor het URI-veld toe via `hook_form_alter`. De structuur van de URI moet bestaan uit een IPv6-adres en de gebruiker mag het *device* niet eerder toegevoegd hebben. Ook hier stoppen we extra gegevens in de databank via `hook_node_insert`. Er wordt een entry in de tabel `coap_sensor_interested_user` toegevoegd waarmee wordt aangegeven dat de gebruiker geïnteresseerd is in het *device*. Deze tabel heeft een kolom om aan te geven dat de entry verwerkt moet worden. Deze kolomwaarde is afhankelijk van het feit of er andere gebruikers geïnteresseerd zijn in het *device* op het moment dat dit *device* toegevoegd wordt. Als dit het geval is hoeven we geen resource *discovery* voor dit *device* uit te voeren en wordt als kolomwaarde het getal 2 gebruikt. We zijn immers zeker dat alle resourcegegevens al in de databank zitten. Indien niemand geïnteresseerd is in het *device* wordt als kolomwaarde het getal 1 gebruikt en wordt een resource *discovery* uitgevoerd. Na het toevoegen van het *device* krijgt de gebruiker de visuele representatie van de content te zien.

De visuele representatie gebeurt hier niet met een template maar door gebruik te maken van een form [2]. Wanneer de pagina geladen is, start een periodieke poll vanuit jQuery naar de *callback*functie `coap_device_page_callback()`. In deze functie wordt de rij die overeenkomt met het toegevoegde *device* uit de tabel `coap_sensor_interested_user` gehaald. Afhankelijk van de

kolomwaarde 'new' gebeuren er enkele zaken. Indien 'new' de waarde 0 heeft, is de output van deze *callback*functie een lege string en gebeurt er verder niks. Bij de waarde 1 heeft de functie als output 'show_counter'. Bij elke poll vanuit jQuery waar de functie 'show_counter' als output heeft, wordt een teller verhoogd. Afhankelijk van deze teller wordt er een boodschap getoond op de pagina. Op deze manier heeft de gebruiker feedback indien de resource *discovery* langer duurt dan normaal. Een voorbeeld zie je op figuur 5.12.

coap device

[View](#)[Edit](#)[Devel](#)

Submitted by [Stef2](#) on Thu, 05/30/2013 - 13:14

Device URI:

2001:6a8:1d80:200::2

[Refresh device](#)

It's been 16s since the discovery has started. This is taking longer than usual. Please refresh the device.

Figuur 5.12: Variabele boodschap bij content van type CoAP-device

In het geval 'new' de waarde 2 bevat, moet `field_resource_references` geupdated worden. Alle referenties in het veld worden verwijderd en de nieuwe lijst van referenties wordt op volgende manier opgebouwd. De gegevens van de resources in de *well-known/core* worden opgehaald uit de `coap_sensor_resourcetabel` en voor elke resource wordt gekeken of de gebruiker al geïnteresseerd is in deze resource. Als dit zo is hoeft er enkel een referentie aangemaakt te worden naar deze resource want een gebruiker mag een resource maar één keer toevoegen. Als dit niet het geval is moet er content van het type CoAP-resource aangemaakt worden. Zoniet kan er geen referentie naar deze content toegevoegd worden. Er zijn twee aanvaardbare manieren om content toe te voegen, namelijk door gebruik te maken van `node_save()` en dergelijke specifieke functies voor nodes of door gebruik te maken van de *Entity API*. Deze laatste manier heeft de voorkeur omdat de aangeboden functies van de *Entity API* generiek zijn en op alle soorten entities werken. De node-specifieke functies werken enkel op nodes.

Als het veld geupdated is heeft de *callback*functie 'reload' als output. Wanneer deze string herkend wordt in jQuery wordt de pagina herladen. Na het herladen zie je dat het veld

`field_resource_reference` een lijst van referenties bevat. Nu kan de gebruiker doorklikken naar een resource naar keuze. Wanneer de gebruiker die het *device* toevoegde deze content bekijkt, heeft hij de mogelijkheid de *discovery data* te vernieuwen door een nieuwe resource *discovery* uit te voeren. Alle content die overeenstemt met dit *device* wordt geupdated bij de volgende jQuery-poll wanneer ze weergegeven worden. De mogelijkheid om een nieuwe resource *discovery* uit te voeren voor een *device* is enkel beschikbaar voor de gebruiker die het *device* toevoegde. Dit om anonieme gebruikers geen kans te geven het *device* te overspoelen met resource discoveries.



Figuur 5.13: Visuele representatie van het contenttype CoAP-device

5.6 Implementatie van de CoAP-libraryhooks

In deze paragraaf bekijken we hoe deze module de *hooks* implementeert die de CoAP-library voorziet (Zie paragraaf 4.2.3).

5.6.1 `hook_receive_notification()`

Deze *hook* wordt opgeroepen door de CoAP-library wanneer een bericht binnenkomt op de socket. Er wordt een response-object gemaakt van de klass CoAPMessage dat tevens meegegeven wordt als enige attribuut van de *hook*.

In deze module worden in de *hook* volgende stappen ondernomen:

- De benodigde waarden worden uit het response-object gehaald en in de databank gestopt voor de huidige gebruiker.
- De user ID's van alle andere geïnteresseerden (dus niet de huidige gebruiker) voor de betreffende resource worden opgehaald uit de databank.
- Voor elk van deze gebruikers worden de waarden uit het response-object ook toegevoegd aan de databank.

5.6.2 `hook_receive_error()`

Deze *hook* wordt voorzien van de volgende parameters: een errorstring, het IPv6-adres van het *embedded device* en de naam van de resource. Ze wordt opgeroepen wanneer één van de volgende gebeurtenissen zich voordoet:

- De socket kon niet worden geopend naar het *embedded device*. Deze gebeurtenis resulteert in een errorstring gelijk aan 'unreachable'.
- De tijdsspanne verstrijkt waarin een antwoord zou moeten ontvangen zijn, dit wordt bepaald met het exponential backoff-mechanisme (Zie paragraaf 4.2.4). Dit levert een errorstring gelijk aan 'delay'.
- Het maximum aantal pogingen om het bericht opnieuw te versturen is verstreken (Zie paragraaf 4.2.4). Nu zal de errorstring gelijk zijn aan 'broken'.
- Ook wanneer het zeker is dat er geen fout is opgetreden, wordt dit gemeld met deze *hook*. De errorstring wordt dan gelijk aan 'none'.

Wanneer nu de *hook* wordt opgeroepen voor de module die in dit hoofdstuk besproken wordt, zal de errorstring opgeslagen worden bij de betreffende resource in de databank.

5.6.3 `hook_stop_observers()`

Wanneer een *observe* moet beëindigd worden, om welke reden dan ook (bijvoorbeeld wanneer de resource niet meer te bereiken is), wordt deze *hook* opgeroepen. Men moet hier echter wel opletten, want de resource kan zich in een soort slaaptoestand bevinden. De resource lijkt dan onbereikbaar, maar deze zal nog steeds notificaties sturen.

De hook krijgt als parameters het IPv6-adres van het *embedded device* en de naam van de resource mee. Zo kan de module die deze *hook* implementeert ervoor zorgen dat de toestand consistent blijft voor de andere gebruikers.

Concreet zal de module die in dit hoofdstuk wordt besproken, in de databank aanduiden dat gebruikers de betreffende resource niet meer aan het observeren zijn. Dit wordt voor elke user gedaan die geïnteresseerd was in deze resource.

Hoofdstuk 6

Uitbreidingen

Het eindresultaat van onze module is een afgerond geheel en is dus ook bruikbaar. Er zijn echter nog enkele mogelijke uitbreidingen die wij voor ogen zagen. Deze hebben wij niet geïmplementeerd wegens tijdsgebrek. We sommen in dit hoofdstuk enkele van deze uitbreidingen op en bespreken hoe wij ze eventueel zouden implementeren.

6.1 Keuze van interval bij automatisch ophalen

Onze huidige module voorziet een alternatieve vorm van de CoAP-*observe* voor CoAP-resources die niet *observable* zijn. Een gebruiker kan namelijk waarden periodiek laten ophalen, dit gebeurt aan de hand van opeenvolgende *GET requests* waartussen een bepaald interval ligt. In onze module bedraagt die een hardgecodeerd aantal seconden (namelijk 5). Men zou echter de module gebruiksvriendelijker en meer configureerbaar kunnen maken door de gebruiker het interval te laten kiezen. Dit zou gelijkaardig kunnen gebeuren aan de keuze van het polling interval, eerder besproken in paragraaf 5.3.5.

6.2 Opvragen devices in subnetwerk met resource directory

Eerder werd al uitgelegd hoe *resource discovery* in zijn werk gaat (Zie paragraaf 3.3.1). Men kan dit principe doortrekken op subnetwerkniveau. Wanneer op een subnetwerk een *resource directory* voorzien is kunnen *embedded devices* er hun *well-known/core*'s in plaatsen. De module zou dan de gebruiker een lijst kunnen presenteren die alle aanwezige *devices* in het subnetwerk opsomt, samen met de aangesloten resources. Men zou ook hier weer modulair kunnen werken en de *devices* voorstellen als instanties van het *contenttype* CoAP *device*.

Nog een ander mogelijk alternatief dat overwogen werd is het sturen van een *broadcast* op het subnetwerk. Deze *broadcast* bevat dan een *GET request* die de *well-known/core* opvraagt van elk *embedded device* in het subnetwerk. Het nadeel bij deze benadering is dat een *broadcast* zeer belastend is voor het subnetwerk, een netwerkbeheerder kan er dan ook voor kiezen broadcast-verkeer niet toe te laten. Bovendien is het niet de bedoeling dat als een gebruiker geïnteresseerd is in een ander subnetwerk, dat die belasting zomaar kan worden uitgevoerd. Dit zal vaak ook zelfs niet lukken wanneer de Drupal-server zich niet in hetzelfde subnetwerk bevindt. Men zou dan kunnen opteren voor *multicast*, maar dit is ongewenst in *constrained* netwerken. Wanneer men echter gebruik maakt van een resource *directory* kan men bovendien beslissen welke *embedded devices* publiek worden gemaakt, wat de beveiliging en belasting van het subnetwerk ten goede komt.

6.3 Conditional observe

Het principe van een *conditional observe* is een uitbreiding van de standaard CoAP *observe*. Het werd mede ontwikkeld door iMinds en wordt omschreven in een aparte *draft*, namelijk de CoAP *Conditional Observe draft* (huidig versie 3) [35].

Een *conditional observe* werkt net als een gewone CoAP *observe* met dat verschil dat een waarde pas naar de client wordt opgestuurd wanneer de waarde aan een bepaalde voorwaarde voldoet. Dit kan bijvoorbeeld een temperatuur zijn waarvan je pas op de hoogte wilt gesteld worden wanneer die hoger wordt dan twintig graden Celsius. Voor de *conditional observe* wordt optie nummer 18 gebruikt. Deze optie moet steeds vergezeld worden van een gewone *observe* optie (nr. 6) aangezien deze er een extensie van is.

Type	C/E	Name	Data type	Length	Default
18	E	Condition	uint	1-5 B	(none)

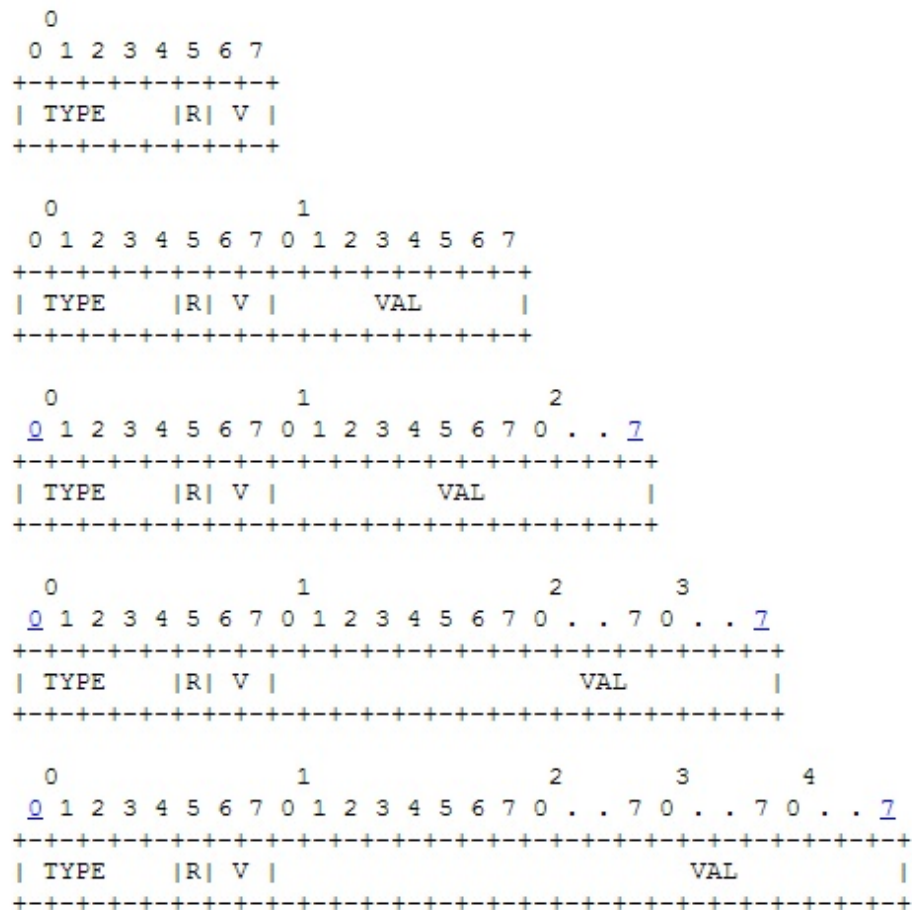
Figuur 6.1: *Conditional observe optie (Option Delta 18)*

De waarde kan variëren in lengte van één tot vijf bytes en bestaat uit de volgende onderdelen:

- TYPE: Beschrijft het type van de voorwaarde, bijvoorbeeld groter dan, kleiner dan, is

gelijk aan,...

- R: In een *request* duidt deze bit aan of de client de notificaties als CON (1) of NON (0) verkiest. In een *response* duidt deze bit aan of de server bereid is om in te gaan op het verzoek van de *client* om het betreffende soort berichten te sturen.
- V: Deze twee bits duiden aan wat het type van de voorwaarde is (*Integer*, tijdsduur in seconden of *float*).
- VAL: De waarde van de voorwaarde, de lengte hiervan kan variëren van één tot vier bytes.



Figuur 6.2: Conditional observe optie formaat

6.3.1 Keep-alive

Men kan gebruik maken van de *Keep-alive* optie om er zeker van te zijn dat de *client* nog deel uitmaakt van de lijst van *observers*. Het betreft hier optie 30.

Type	C/E	Name	Format	Length	Default
30	E	Keep-alive	Duration in s	1 B	(none)

Figuur 6.3: De *Keep-alive* optie met *Option Delta 30*

Wanneer een *client* een *Keep-alive* optie meestuurt, vraagt die aan de server te bevestigen dat de *client* nog tot de lijst met *observers* behoort. En dit telkens wanneer een bepaald interval, meegegeven door de *client* in de optie, verstrijkt en er in dat interval geen notificaties of enkel NON-berichten ontvangen zijn. De server stuurt dan een leeg bericht op, de voorwaarde werd immers niet voldaan.

6.4 Custom Entity

We gebruiken de *Entity API* al voor manipulatie van content. Wat er nog mee mogelijk is, is het ontwerpen van een eigen *entity*. Dit is toepasbaar op de waarden van resources. Momenteel worden waarden opgeslagen in de tabel `coap_resource_values`. Deze tabel wordt bij installatie aangemaakt en het manipuleren ervan wordt door onze eigen code afgehandeld. Als we een *entity* `coap_resource_value` maken, worden de waarden nog steeds in dezelfde tabel opgeslaan maar kunnen we de manipulatie ervan laten gebeuren door de functies die de *Entity API* aanbiedt. Een bijkomend voordeel is dat we *custom views* kunnen maken van *entities*. Gebruikers kunnen dan zelf *custom views* van waarden aanmaken. Een gebruiker kan dus bijvoorbeeld alle waarden van 5 mei tot 5 juni opvragen.

Een nuttig voorbeeld van het aanmaken van een eigen *entity* in code is beschikbaar in de module *Model Entities* [24].

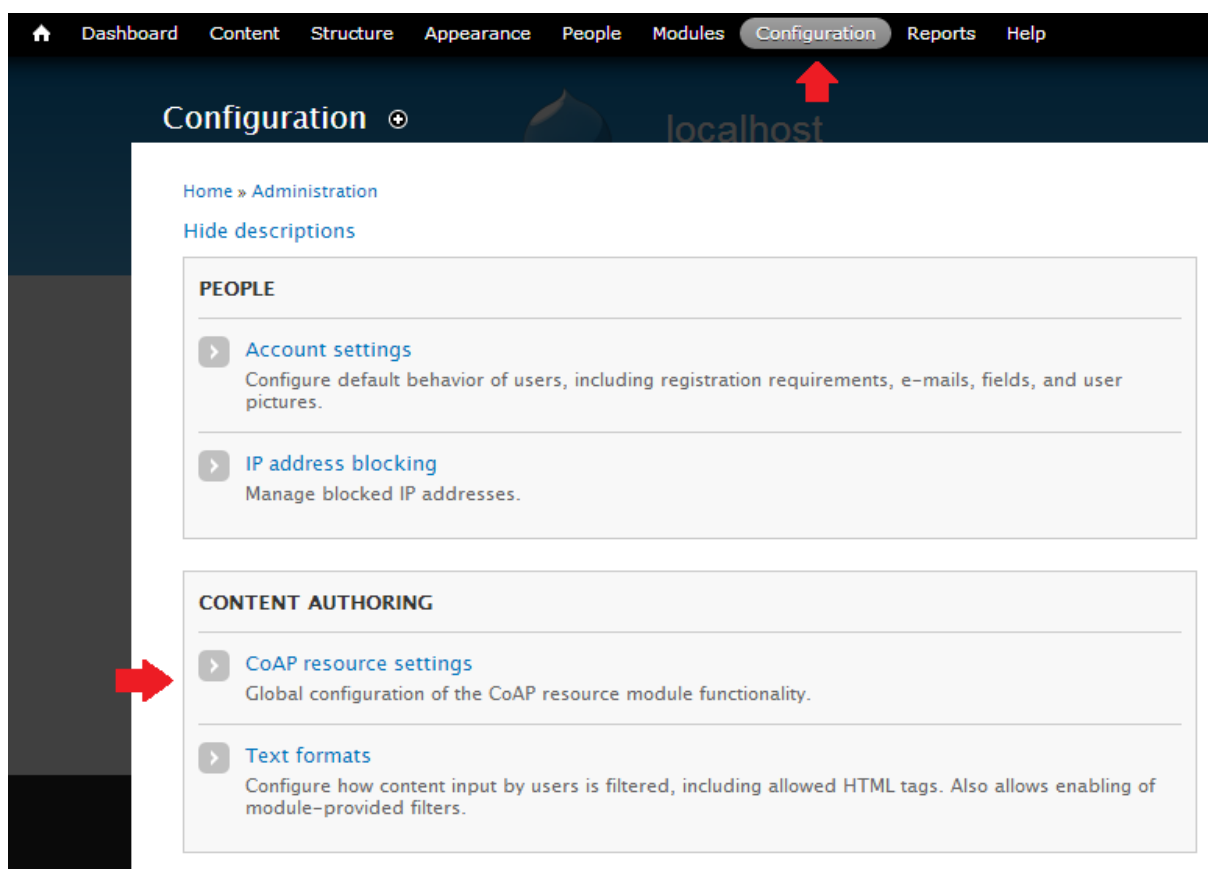
Een bijkomende uitbreiding is het gebruik van de module *Entity Reference* [23]. Omdat we zelf een tabel creëren zonder aan Drupal te zeggen dat deze een speciale betekenis heeft, staan we zelf in voor de visualisatie van de waarden. Maar door een eigen *entity* te maken kunnen we de waarden weergeven door gebruik te maken van een nieuw veld dat geïntroduceerd wordt door de *Entity Reference* module.

6.5 Interface Description

Het attribuut *if* (*interface description*) van het *CoRE Link Format* kan info bevatten over welke methodes deze resource ondersteunt. Momenteel wordt deze info opgeslagen maar nog niet gebruikt. Deze informatie kan gebruikt worden om de methodes (GET, PUT, POST en DELETE) niet voor alle resources beschikbaar te laten zijn. De verschillende waarden en hun betekenis die het attribuut kan aannemen vind je in de *CoRE Interfaces draft* [36].

6.6 Configuratie op maat

Bij installatie wordt een pagina voor configuratie voorzien. Deze kan je bereiken door eerst op 'Configuration' te klikken en dan op 'CoAP resource settings'. Enige configuratieinstellingen komen hier te staan. Om de pagina die getoond wordt aan passen moet je de functie `coap_resource_admin_settings()` in `coap_resource.module` aanpassen.



Figuur 6.4: Navigatie naar de configuratiepagina als deze voorzien wordt

6.7 View Modes

De content die we aanbieden wordt op elke pagina op dezelfde manier getoond. Door gebruik te maken van *view modes* kunnen we hier verandering in brengen. Wanneer content van het type `coap_resource` getoond wordt, kan je er verschillende functies op uitvoeren en zie je een *history*tabel. Maar soms zou het handig zijn als je enkel de *history*tabel ziet. Een complete *tutorial* met uitleg vind je online [30].

6.8 DNS

Het programma werkt enkel met IP-adressen. Door gebruik te maken van de PHP-functies `gethostbyaddr(string $ip_addr)`[31] en `gethostbyname(string $hostname)`[32] kunnen *hostnames* gebruikt worden i.p.v. hun respectievelijk IP-adres.

6.9 Help

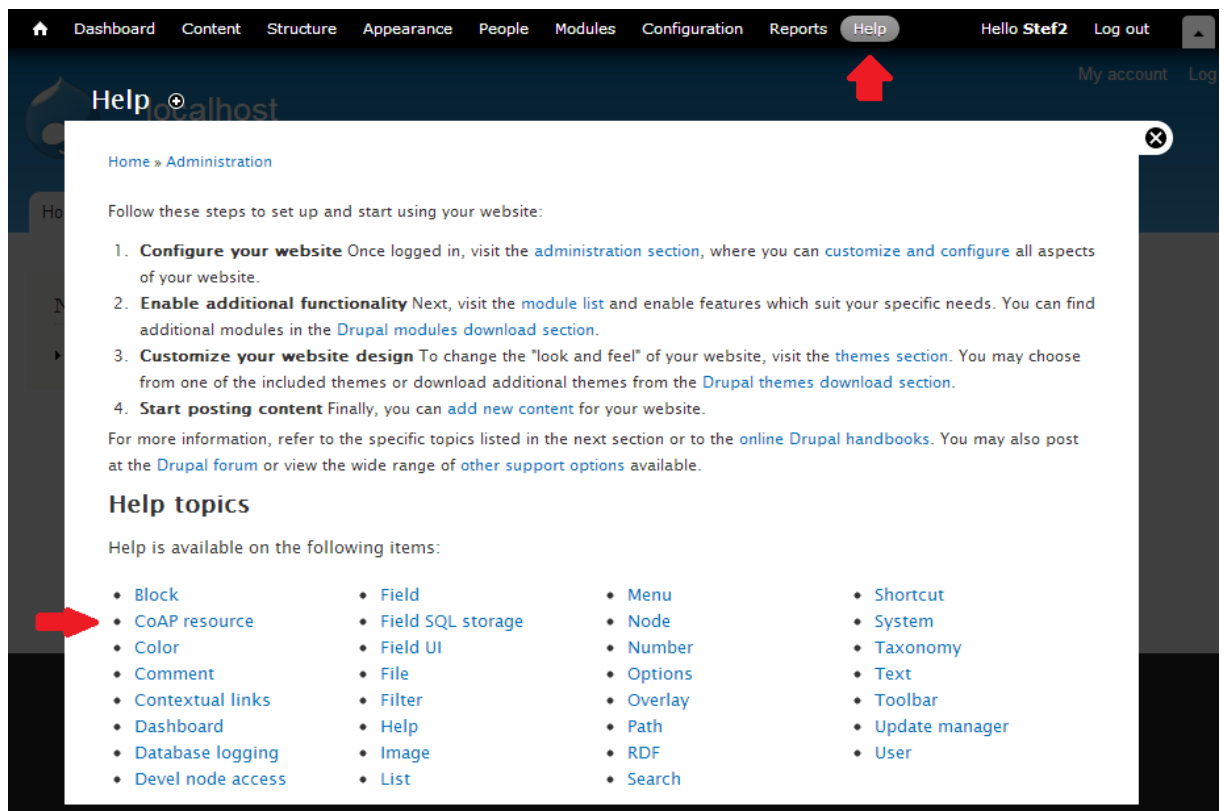
Een standaard hulppagina kan voorzien worden door *hook_help* te implementeren. Deze kan opgevuld worden met de handleiding.

Er kan ook gebruik gemaakt worden van de *Advanced Helpmodule* [19]. Wanneer deze module *enabled* is, kan je hulppagina's opslaan in HTML-formaat en deze in je moduledirectory zetten. Deze module maakt ook het gebruik van pop-ups mogelijk en laat de administrator van de website beslissen voor wie de hulp beschikbaar moet zijn. We laten het aan de lezer over om alle mogelijkheden van deze module te bekijken.

6.10 Block Options

Voor optie 28 (Size) en 27 (Block1) bieden we nog geen ondersteuning, momenteel worden ze genegeerd. Block1 hebben we reeds besproken in 3.2.3. De Size-optie wordt voor meerdere dingen gebruikt, maar ze hebben allemaal een verband met een schatting van de grootte van de resource representatie:

- In een request: om aan de server te vragen of hij een schatting meestuurt met zijn response. Hier moet als waarde 0 gebruikt worden.
- In een response met een Block2-optie: om de schatting van de server aan te geven.



Figuur 6.5: Navigatie naar de hulppagina als deze voorzien wordt

- In een response met een Block1-optie: om de schatting van de client aan te geven.

De *client* of server kan aan de hand van de schatting beslissen om grotere of kleinere *blocks* door te sturen. Deze optie is niet verplicht te ondersteunen dus moet er rekening gehouden worden met dat de server of *client* deze optie negeert.

6.11 Anonieme gebruikers

Momenteel worden anonieme gebruikers niet meer ondersteund. Er is wel een analyse uitgevoerd hoe ze best geïmplementeerd worden. Voor anonieme gebruikers is het beter om zelf geen gegevens naar de databank te schrijven. Dit om ervoor te zorgen dat de databank niet overspoeld wordt met data.

Anonieme gebruikers hebben als *user-ID* (uid) het getal 0. Anders dan gewone gebruikers die een uniek uid hebben kan je anonieme gebruikers hier niet op onderscheiden. Gebruikers hebben ook een *session-ID* (sid) dat wel uniek is per gebruiker. Nu moeten we de gegevens van de anonieme gebruiker bijhouden op een plaats die overal toegankelijk is maar dit mag niet de databank zijn. Daarom wordt er gebruik gemaakt van de globale sessievariabele [18] `$_SESSION`.

Deze variabele is een PHP-tabel en het ligt voor de hand deze te indexeren op de sid. Alle gegevens worden nu best opgeslagen en aangesproken op volgende manier: `$_SESSION[$sid]` met `$sid` een variabele die de sid van de gebruiker bevat.

Bibliografie

- [1] <https://addons.mozilla.org/en-US/firefox/addon/copper-270430/>, 21 mei 2013
- [2] http://api.drupal.org/api/drupal/developer!topics!forms_api_reference.html/7, 18 februari 2013
- [3] <http://api.drupal.org/api/drupal/includes!database!database.inc/group/database/7>
- [4] http://api.drupal.org/api/drupal/modules!field!field.crud.inc/group/field_crud/7, 13 juni 2013
- [5] <http://api.drupal.org/api/drupal/modules!field!field.module/group/field/7>, 30 mei 2013
- [6] http://api.drupal.org/api/drupal/modules!system!system.api.php/function/hook_enable/7, 18 februari 2013
- [7] http://api.drupal.org/api/drupal/modules!system!system.api.php/function/hook_install/7, 18 februari 2013
- [8] http://api.drupal.org/api/drupal/modules!system!system.api.php/function/hook_schema/7, 18 februari 2013
- [9] http://api.drupal.org/api/drupal/modules!system!theme.api.php/function/hook_preprocess_HOOK/7, 14 mei 2013
- [10] <http://coap.me>, 31 mei 2013
- [11] <http://datatracker.ietf.org/doc/draft-ietf-core-coap/>, 21 mei 2013
- [12] <http://datatracker.ietf.org/doc/draft-ietf-core-observe/>, 21 mei 2013
- [13] <https://developers.google.com/chart/>, 27 mei 2013
- [14] <http://developer.yahoo.com/weather/>, 18 februari 2013

- [15] <http://drupal.org/mission>, 31 mei 2013
- [16] <https://drupal.org/node/1169864>, 29 mei 2013
- [17] <http://drupal.org/node/1261744>, 27 mei 2013
- [18] <http://drupal.org/node/33513>, 27 mei 2013
- [19] http://drupal.org/project/advanced_help, 27 mei 2013
- [20] http://drupal.org/project/background_process, 19 februari 2013
- [21] http://drupal.org/project/block_refresh, 5 maart 2013
- [22] <http://drupal.org/project/entity>, 24 april 2013
- [23] <http://drupal.org/project/entityreference>, 27 mei 2013
- [24] <http://drupal.org/project/model>, 27 mei 2013
- [25] <http://drupal.org/project/progress>, 19 februari 2013
- [26] <http://drupal.org/project/references>, 24 april 2013
- [27] <http://drupal.org/project/views>, 27 mei 2013
- [28] <http://dsheiko.com/weblog/websockets-vs-sse-vs-long-polling>, 13 juni 2013
- [29] <http://hinrg.cs.jhu.edu/joomla/images/stories/coap-ipsn.pdf>, 13 juni 2013
- [30] pingv.com/blog/custom-drupal-7-view-modes-different-ways-see-your-content, 27 mei 2013
- [31] <http://php.net/manual/en/function.gethostbyaddr.php>, 27 mei 2013
- [32] <http://php.net/manual/en/function.gethostbyname.php>, 27 mei 2013
- [33] <http://stackoverflow.com/questions/12683530/origin-http-localhost-is-not-allowed-by-access-control-allow-origin>, 17 maart 2013
- [34] <http://tools.ietf.org/html/draft-ietf-core-block-11>, 21 mei 2013
- [35] <http://tools.ietf.org/html/draft-li-core-conditional-observe-03>, 21 mei 2013
- [36] <http://tools.ietf.org/html/draft-shelby-core-interfaces-05>, 21 mei 2013

-
- [37] <http://tools.ietf.org/html/rfc5988>, 27 mei 2013
- [38] <http://tools.ietf.org/html/rfc6690>, 21 mei 2013
- [39] <http://www.acquia.com>
- [40] <http://www.metachunk.com/blog/adding-module-path-drupal-7-theme-registry>, 14 mei 2013
- [41] <http://www.websocket.org/quantum.html>, 13 juni 2013
- [42] <http://www.wireshark.org>, 27 mei 2013
- [43] Melanon B., Luisi J., Ngyesi K., Anderson G., Somers B., Corlosquet S., Freudenberg S., Lauer M., Charlevale E., Lortan F., Nordin D., Szrama R., Stewart S., Strawn J., Travis B., Hakimzadeh D., Scavarda A., Albala A., Micka A., Douglass R., Monks R., Scholten R., Wolanin P., VanValkenburgh K., Stout G., Dolin K, Mars F., Boyer S., Gifford M., Sarahe C. (2011). *The Definite Guide to Drupal 7*. APress.
- [44] Tomlinson T. (Juni, 2010) *Beginning Drupal 7*. APress.
- [45] Travis B. (Februari, 2011) *Pro Drupal 7 for Windows developers*. APress.

Lijst van figuren

1.1	Internet of Things	1
2.1	Drupal logo	6
2.2	Wat is Drupal?	7
2.3	Basiswebsite van Drupal met Bartik-theme	8
2.4	Organisatie van Drupal modules	14
2.5	Laadcyclus van de pagina in Drupal	18
2.6	Voorbeeld van de laadcyclus van een pagina waar een node weergegeven wordt .	23
2.7	Field tabellen	25
2.8	Weermodule	28
3.1	CoAP-lagen (CoAP 17 draft)	31
3.2	Berichtformaat (CoAP 17 draft)	31
3.3	CoAP-optie (CoAP draft 16)	34
3.4	HTTP-berichtformaat	35
3.5	Berichtuitwisseling (CoAP 17 draft)	36
3.6	Twee GET-requests met piggy-backed responses (CoAP 17 draft)	37
3.7	GET-request met separate response (CoAP 17 draft)	37
3.8	Blokoptiewaarde - bytes en bits worden aangegeven bovenaan de figuur	38
5.1	Resourcewaarde opvragen met de CoAP-sensormodule in Drupal	55
5.2	Temperatuurmodule met HTTP/COAP-proxy	61
5.3	Block met HTML-formulieren die de gebruiker toelaat meerdere resources met één block te bevragen	65
5.4	CoAP-resource met volledige REST functionaliteit	66
5.5	Opsomming van resources aan de hand van resource discovery	68

5.6	EER-diagram van de door ons toegevoegde tabellen	69
5.7	Content toevoegen door op 'Add content' te klikken	73
5.8	Adding content of contenttype CoAP-resource	74
5.9	Visuele representatie van het contenttype CoAP-resource	75
5.10	Mechanisme om fouten te tonen aan de gebruiker	79
5.11	Adding content of contenttype CoAP-device	80
5.12	Variabele boodschap bij content van type CoAP-device	81
5.13	Visuele representatie van het contenttype CoAP-device	82
6.1	Conditional observe optie (Option Delta 18)	86
6.2	Conditional observe optie formaat	87
6.3	De Keep-alive optie met Option Delta 30	88
6.4	Navigatie naar de configuratiepagina als deze voorzien wordt	89
6.5	Navigatie naar de hulppagina als deze voorzien wordt	91

