



Geassocieerde Faculteit Toegepaste Ingenieurswetenschappen
Vakgroep Informatica

Internet of Things: Integratie in Drupal

door

Kobe WRIGHT
Stef DE WAELE

Promotoren: dr. ir. P. SIMOENS, prof. dr. ir. I. MOERMAN, dr. ir. J. HOEBEKE
Begeleiders: ing. J. ROSSEY, ir. F. VAN DEN ABEELE

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

Academiejaar 2012–2013



Geassocieerde Faculteit Toegepaste Ingenieurswetenschappen
Vakgroep Informatica

Internet of Things: Integratie in Drupal

door

Kobe WRIGHT
Stef DE WAELE

Promotoren: dr. ir. P. SIMOENS, prof. dr. ir. I. MOERMAN, dr. ir. J. HOEBEKE
Begeleiders: ing. J. ROSSEY, ir. F. VAN DEN ABEELE

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

Academiejaar 2012–2013

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

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

Geassocieerde Faculteit Toegepaste Ingenieurswetenschappen

Hogeschool Gent

Vakgroep Informatica

Samenvatting

Het Internet of Things (IoT), waarbij verscheidene soorten apparaten met het internet worden verbonden, wordt steeds belangrijker. Het biedt de mogelijkheid deze apparaten van op afstand te observeren en besturen op een gemakkelijke en dynamische manier.

Het succesvolle en wijd verspreide HTTP-protocol is echter te zwaar, het veroorzaakt te veel overhead om gebruikt te kunnen worden in sensornetwerken, waar de bandbreedte beperkt is. Bovendien wordt de nodige energie voor sensoren vaak geleverd door een batterij, dus is het belangrijk energieconsumptie te beperken. In ruil voor de betrouwbaarheid en robuustheid van het HTTP-protocol biedt het CoAP-protocol minder betrouwbare communicatie, maar het veroorzaakt slechts een minimale overhead en biedt een sensor de mogelijkheid op eigen initiatief data te sturen naar clients die geïnteresseerd zijn.

Het doel van deze masterproef bestaat erin een Drupal-module te ontwikkelen die het mogelijk maakt op een gebruiksvriendelijke en dynamische manier sensor data op te vragen en eventueel sensoren te configureren. Bovendien moeten nieuw aangesloten sensoren gedetecteerd worden. Dit alles moet gebeuren door enkel gebruik te maken van native CoAP-communicatie.

Trefwoorden

Internet of things, Drupal, CoAP, sensornetwerk

Inhoudsopgave

| | |
|---|------------|
| Lijst van gebruikte afkortingen | iii |
| 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? | 8 |
| 2.3 Werking van Drupal | 10 |
| 2.3.1 Drupal bouwstenen | 10 |
| 2.3.2 Pagina aanroepen | 13 |
| 2.4 jQuery in Drupal | 15 |
| 2.4.1 Met een formulier | 15 |
| 2.4.2 Met AJAX in jQuery | 16 |
| 2.4.3 Problemen | 16 |
| 3 CoAP | 17 |
| 3.1 Wat is CoAP? | 17 |
| 3.1.1 Berichtformaat | 18 |
| 3.2 Communicatiemogelijkheden | 22 |
| 3.2.1 Betrouwbaarheid | 22 |
| 3.2.2 Request/response model | 23 |
| 3.3 Extra Features | 24 |

| | | |
|----------|---|-----------|
| 3.3.1 | Observe | 24 |
| 3.3.2 | Resource discovery | 24 |
| 3.3.3 | Resource directory | 24 |
| 4 | CoAP library module | 25 |
| 4.1 | Doel | 25 |
| 4.1.1 | Essentiële, te ondersteunen functies | 26 |
| 4.1.2 | Optionele, handige functies | 27 |
| 4.2 | Implementatie | 28 |
| 4.2.1 | Proceduregericht | 28 |
| 4.2.2 | Objectgericht | 29 |
| 4.2.3 | Hooks voor notificaties | 30 |
| 4.2.4 | Opvangen van verloren berichten | 32 |
| 5 | CoAP resource module | 33 |
| 5.1 | Architectuur | 33 |
| 5.1.1 | Functionaliteit | 33 |
| 5.2 | Evolutie | 35 |
| 5.3 | Temperatuurmodule met HTTP/CoAP-proxy | 35 |
| 5.3.1 | Proxy | 36 |
| 5.3.2 | Achtergrondprocessen | 37 |
| 5.3.3 | Geschiedenis van opvragingen | 37 |
| 5.3.4 | Push-strategie: node.js | 38 |
| 5.3.5 | Pull-strategie | 39 |
| 5.4 | Temperatuurmodule met native CoAP | 40 |
| 6 | Arduino | 41 |

Lijst van gebruikte afkortingen

| | |
|-------|--|
| ACK | Acknowledgement |
| AJAX | Asynchronous JavaScript and XML |
| ASCII | American Standard Code for Information Interchange |
| CMS | Content Management System |
| CoAP | Constrained Application Protocol |
| CON | Confirmable |
| CR | Carriage Return |
| HTTP | HyperText Transfer Protocol |
| IoT | Internet of Things |
| IPv4 | Internet Protocol versie 4 |
| IPv6 | Internet Protocol versie 6 |
| LF | Line Feed |
| M2M | Machine-to-machine |
| NON | Non-confirmable |
| PHP | PHP Hypertext Preprocessor |
| RST | Reset |
| TKL | ToKen Length |
| TLV | Type Length Value |

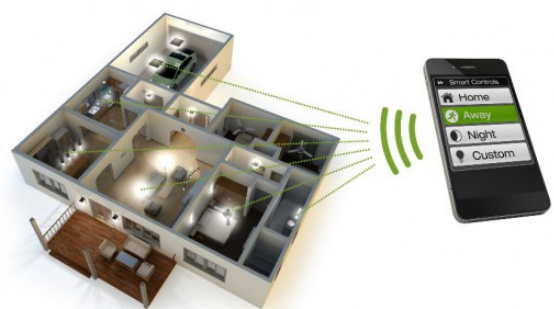
| | |
|-------|-----------------------------|
| UDP | User Datagram Protocol |
| URI | Uniform Resource Identifier |
| WOEID | Where On Earth ID |

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 je smartphone tot een wasmachine of zelfs je auto.

De komende jaren zal de omvang van het Internet exploderen omdat steeds meer van deze dagdagelijkse objecten ermee verbonden zullen worden. Het Internet zal evolueren naar een Internet of Things 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 blijkbaar 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 zijn niet geoptimaliseerd voor verkeer van en naar een sensornetwerk. Het HyperText Transfer Protocol (HTTP) bijvoorbeeld, biedt een zeer betrouwbare vorm van communicatie, maar om die betrouwbaarheid te realiseren is een aanzienlijke hoeveelheid overhead nodig. Het is deze nood aan een lightweight protocol dat aan de oorzaak ligt van de recente ontwikkeling van het CoAP protocol. Het spitst zich toe op communicatie met minimale overhead. Als gevolg moet het CoAP protocol inboeten aan betrouwbaarheid. Het maakt immers gebruik van het minder betrouwbare User Datagram Protocol (UDP). Daar waar HTTP juist betrouwbaarheid biedt, garandeert UDP niet dat alle pakketten effectief hun bestemming bereiken of dat die in volgorde aankomen.

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. 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 van de embedded devices. Meer specifiek willen we een bijdrage leveren aan de realisatie van webservices voor het IoT. Het vertrekpunt hiervoor zullen de recente Constrained Application Protocol (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 Drupal-module 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 berichtverkeer stuurt en afhandelt en een 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

Voor de eigenlijke ontwikkeling van de Drupal-module werden verscheidene boeken geraadpleegd, waarbij voor de implementatiedetails van het CoAP protocol, de CoAP drafts werden geraadpleegd.

In een eerste fase van deze masterproef zal de Drupal-module wel nog gebruik maken van een HTTP/CoAP proxy. Wanneer deze eerste fase geoptimaliseerd is, wordt de proxy uitgesloten en wordt er enkel nog gebruik gemaakt van het CoAP protocol.

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.

Na uitvoerig bestuderen van de CoAP drafts en het ontmantelen van CoAP berichten in Wireshark, werd geëxperimenteerd met CoAP berichten. 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 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.

Naast deze laatste ontwikkelingen die zich eerder toespitsen op de backend, werd ook vooruitgang geboekt aan de frontend. Een content type werd ontwikkeld zodat een gebruiker gemakkelijk inhoud kan toevoegen aan zijn/haar website die voorzien wordt door onze module.

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 een 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 wordt nog verder 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 gebruiken we een resource directory. 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.

Als laatste stap rest nog het samenvoegen van de frontend en backend. Als resultaat bekomt 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 de 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, in de latere fasen, 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.

In het vijfde hoofdstuk bespreken we de sensormodule.

Een mogelijk zesde hoofdstuk behandelt het ontwikkelen van een embedded device. Hiervoor

werd als basis de Arduino UNO gekozen.

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. [4]



Figuur 2.1: *Drupallogo*

Content Management System

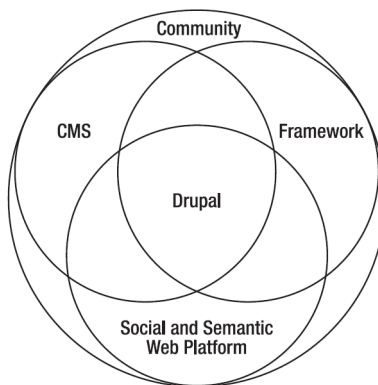
Drupal levert alle functies en mogelijkheden van een krachtig CMS. We denken meteen aan het kunnen inloggen en registreren van gebruikers, verschillende soorten gebruikers kunnen definiëren, verschillende niveaus van permissies,... Ook denken we aan het creëren, aanpassen, beheren, weergeven, categoriseren 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 enorm 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. Drupal's veelzijdigheid wordt nogmaals bewezen met het feit dat het zowel als frontend voor Java-gebaseerde applicaties kan optreden als backend voor AJAX of Flash-driven frontends.

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 te delen is. Op deze manier is het gemakkelijk voor verschillende websites om dezelfde data te delen en door te geven.



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.

Drupal is ontwikkeld in PHP en gebruikt ook een noemenswaardige hoeveelheid JavaScript (in de vorm van jQuery) voor de frontend. Voor het opslaan van content en configuratiegegevens gebruikt Drupal een relationele databank. 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 Drupal Core en een aantal out-of-the-box functies.



Figuur 2.3: *Basiswebsite van Drupal met Bartik-theme*

2.2 Waarom Drupal?

Eerst sommen we de vlakken op waar elke degelijke software goed op moet scoren:

- betrouwbaarheid en robuustheid
- efficiëntie
- flexibiliteit

Als we dit even vergelijken met de principes waarop Drupal gebouwd is:

- 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 bvb. XHTML en CSS.
- Low-resource demanding: om een goede prestatie te garanderen, maakt Drupal gebruik van low-profile codering (bvb. 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.

We zien dat Drupal aan de eerder vermelde voorwaarden voldoet.

Een bijkomend voordeel van Drupal is zijn grote gemeenschap die ondertussen uit al meer dan 630000 actieve gebruikers en ontwerpers bestaat die zich inspannen om Drupal steeds beter te maken. Dit aantal neemt elke dag toe. Dankzij die grote gemeenschap krijg je snel antwoord op je vragen dankzij de inspanning van andere Drupal leden.

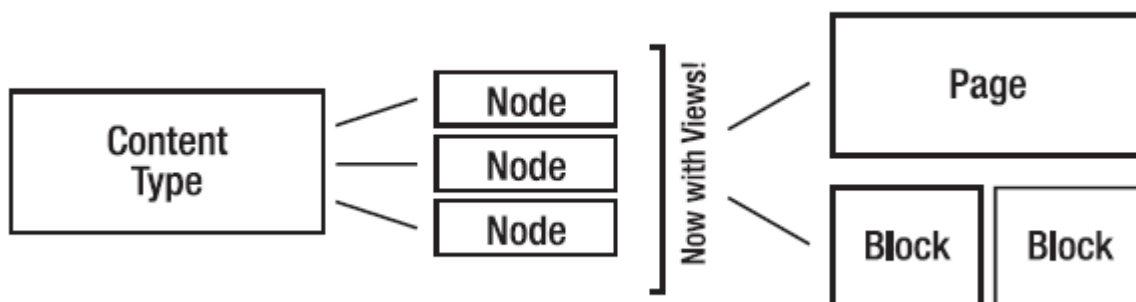
We bespreken nu enkele nadelen van Drupal:

- 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, de bootstrapcode 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-community. Dit is in veel gevallen een voordeel, maar wanneer je een probleem hebt, ben je niet zeker of er wel een oplossing voor bestaat.

- 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.

2.3 Werking van Drupal

Aan de hand van een voorbeeld proberen we de lezer een idee te geven van de werking van Drupal. Bekijk Drupal als een digitale postzegelsorteerder. De nodes zijn de postzegels en de content types zijn de verschillende soorten postzegels (postzegels van € 0,67, € 1,34, ...). Naast content types, 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, ...). 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. Zie Figuur 2.4.



Figuur 2.4: Grafische weergave van hoe Drupal content aanbiedt.

2.3.1 Drupal bouwstenen

We gaan dieper in op de begrippen die gebruikt worden in het voorbeeld en lichten enkele extra termen toe die ook gebruikt worden in Drupal.

Node

Nodes zijn de eenheden met inhoud in Drupal. Die inhoud kan vanalles zijn, van een nieuwsbericht tot een volledige pagina.

Block

Blocks zijn, zoals de naam al 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.

Content type

Het content type geeft het type van de node aan. Het biedt de mogelijkheid om verschillende soorten content te onderscheiden van elkaar omdat ze verschillend zijn van type. Het geeft aan wat een node allemaal mag of kan bevatten.

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, ...).

Users, Roles en Permissions

Users zijn de gebruikers van jouw website die zich aangemeld hebben. Alle user functionaliteit zoals registreren, inloggen, enz... zit reeds in de Drupal core. Roles geven weer tot welk type een gebruiker hoort. Meerdere gebruikers kunnen dezelfde role hebben, en een gebruiker kan meerdere roles hebben. Met permissions kan je aangeven welke privileges een user met een bepaalde role, mag of kan doen. Deze permissions worden gekoppeld aan een role zodat alle gebruikers die deze role hebben automatisch de privileges hebben van deze role. Drupal biedt standaard de Administrator role aan die alle privileges heeft.

Module

Modules bieden je de mogelijkheid om extra functionaliteit in te pluggen op je website. Modules zijn gratis te downloaden van de Drupal community en omdat deze community 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 2.5).

[+ 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 2.5: Organisatie van Drupalmodules

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.

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 community.

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 definiëren door de naam van de hook te laten voorafgaan door de naam van de module die de hook implementeert (zie figuur 2.6).

```
//Act on a node that is being assembled before rendering.  
function coap_resource_node_view($node, $view_mode, $langcode) {  
  if($node->type == 'coap_resource'){  
    $node->content['coap_resource_form'] = drupal_get_form('coap_resource_form', $node);  
  }  
  return $node;  
}
```

Figuur 2.6: Implementatie van *hook_node_view* door de module *coap_resource*

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.

2.3.2 Pagina aanroepen

Elke keer een webbrowser een Drupal pagina opvraagt, gebeuren er een reeks complexe stappen die als eindresultaat de gerenderde pagina hebben. Indien de pagina niet aanwezig is in de cache, worden deze complexe stappen gegroepeerd in twee delen: de bootstrap en de uitvoering van de page callback. De bootstrap bestaat steeds uit dezelfde reeks van acht verschillende fasen die we nader toelichten in. Na de bootstrap, wordt de page callback uitgevoerd. Indien de pagina wel in de cache aanwezig is, wordt er een subset van de bootstrapstappen uitgevoerd. Het hele proces wordt weergegeven in Figuur ??.

Initialisatie van de configuratie

Deze fase houdt onder andere in dat globale variabelen worden gezet, zoals de basis URL van de website. Deze configuratie gebeurt aan de hand van een configuratiebestand en aan de omgeving waarin de server zich bevindt.

Poging tot opvragen van een gecachte pagina

In deze fase tracht Drupal te vermijden dat de volledige bootstrap moet worden doorlopen. Wanneer de pagina al eens is opgevraagd en deze nog geldig is, wordt deze pagina getoond.

Initialisatie van de database abstraction layer

Basisklassen en functies worden geïnitieerd, maar nog geen verbindingen met de databank worden opgezet.

Initialisatie van het Variable System

In deze fase worden de waarden van variabelen uit de variabelentabel gehaald die zich in de databank bevindt en ingevuld bij de juiste naam. Als er modules zijn waarvan hooks moeten worden opgeroepen, worden deze ook ingeladen.

Initialisatie Session Handling

In deze fase wordt aan elke user een sessie gekoppeld. Een anonieme user krijgt geen sessie toegewezen tenzij er iets in de sessie moet worden opgeslagen.

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 de pagina.

De taal van de pagina bepalen

Als de website meerdere talen ondersteunt wordt in deze fase de gekozen taal van de user bepaald.

Laden van modules en initialisatie van het theme

Alle ingeschakelde modules worden ingeladen en het theme wordt geïnitieerd. Deze fase biedt ook nog de mogelijkheid om variabelen in te stellen die pas later in de request beschikbaar zijn.

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.

2.4.1 Met een formulier

In eerste instantie bevatte de module een formulier bestaande uit een textbox 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 de hook `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 de hook `hook_block_view` op van de weer-module door `weather_info_block_view()` op te roepen:
 - het ingegeven WOEID wordt opgehaald met behulp van de Drupal-functie `variable_get()`
 - 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 blok



Figuur 2.7: Weermodule

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. Een oplossing hiervoor is een proxy-script in PHP gebruiken. De AJAX call gebeurt dan naar het proxy script dat zich op de server en dus hetzelfde domein bevindt. Het proxy-script vraagt daar effectief de data op en geeft de uitvoer terug. De browser wordt dus eigenlijk om de tuin geleid.

Bron: <http://stackoverflow.com/questions/12683530/origin-http-localhost-is-not-allowed-by-access-control-allow-origin>

Hoofdstuk 3

CoAP

3.1 Wat is CoAP?

CoAP is een web transfer protocol speciaal ontwikkeld voor netwerkcomponenten die beperkt zijn in zowel geheugen als energieverbruik als machine-to-machine (M2M)communicatie. Naast het minimaliseren van de overhead concentreert CoAP zich ook op het automatiseren van taken. Het mechanisme van built-in discovery 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.

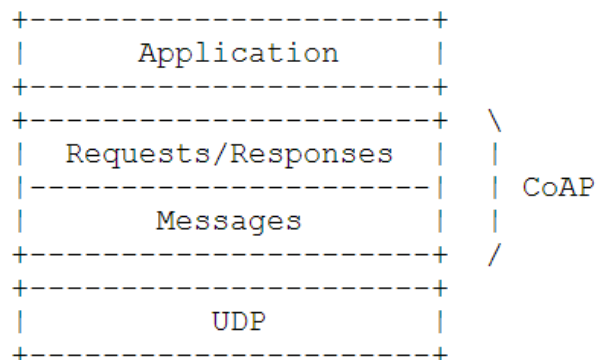
Het interactiemodel van CoAP is vergelijkbaar met het client/server model van HTTP. Hoewel, bij een CoAP implementatie voor M2M interacties kan een device bij de ene berichtuitwisseling client zijn, en bij de andere server. Een CoAP request is equivalent aan een HTTP request en wordt ook gestuurd van 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 Uniform Resource Identifier (URI). De server antwoordt met een response die onder andere een response code (bvb. 2.05 Content) bevat.

Verschillend met HTTP, gebeurt de berichtenuitwisseling asynchroon over een datagramgeöriënteerd transport. 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 verder). Wanneer zo'n bericht niet wordt bevestigd, wordt het bericht meermaals opnieuw

gestuurd volgens het exponential back-off mechanisme.

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.

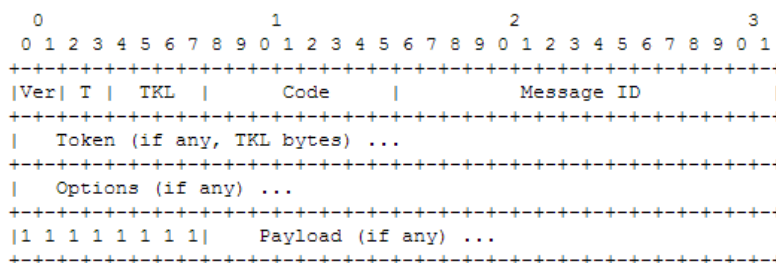
We kunnen CoAP ook in het 7-lagen model plaatsen. Logisch gezien hebben we een CoAP berichtenlaag die het UDP gedeelte en de asynchroniteit van de berichten afhandelt en een request/response laag die gebruik maakt van method en response codes (zie Figuur 3.1). Nochtans bestaat CoAP in werkelijkheid slechts uit één laag waarbij berichten-uitwisseling en het request/response mechanisme enkel en alleen door manipulatie van de header verwezenlijkt wordt.



Figuur 3.1: CoAP lagen (CoAP 14 draft)

3.1.1 Berichtformaat

Om een minimale overhead te realiseren worden de berichten zeer compact gehouden. We geven een kort overzicht van de onderdelen van het berichtformaat (zie Figuur 3.2) en



Figuur 3.2: Berichtformaat (CoAP 14 draft)

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 payload marker (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 zeer weinig is.

Header

Deze vier bytes worden opgedeeld in drie delen:

- Versiegetal (Ver): een 2-bit unsigned integer die de CoAP versie aangeeft.
- Typeaanduiding (T): een 2-bit unsigned integer die het berichttype aangeeft. De mogelijkheden zijn: Confirmable (0), Non-confirmable (1), Acknowledgement (2) of Reset (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 acknowledgement/Reset te linken aan berichten van het type confirmable/non-confirmable.

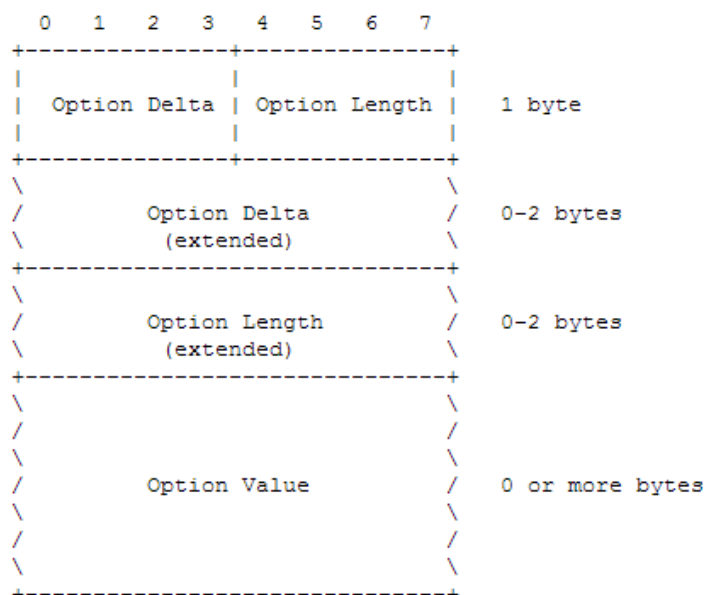
Token

Een token wordt gebruikt om een response te linken met 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. Elke request bevat een token 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 d.m.v. 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 Option Delta. 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 Option Numbers.



Figuur 3.3: CoAP optie (CoAP 14 draft)

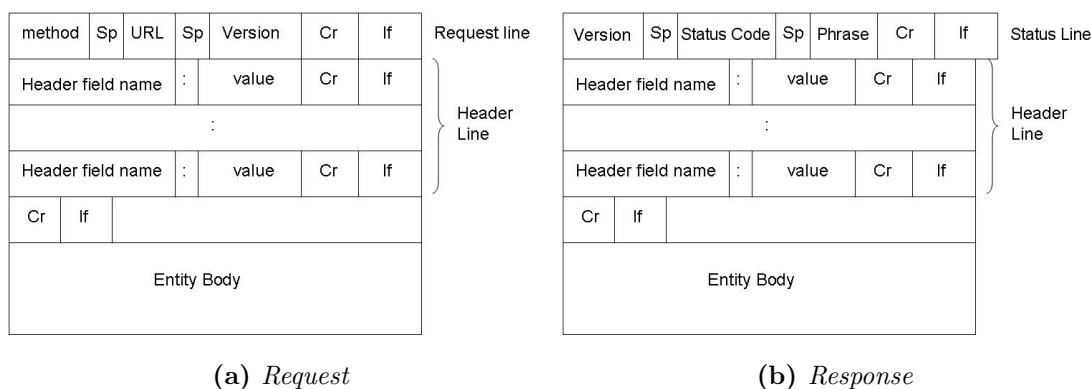
Verschil met HTTP

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.

Deze is opgedeeld in drie delen. Een requestlijn die op zijn beurt opgedeeld is in drie delen, gescheiden door een spatie. Het eerste deel is de methodenaam (GET, POST, HEAD, PUT of DELETE voor HTTP 1.1). Het tweede deel is de URL van de gevraagde resource en het laatste deel is het versienummer. Vervolgens zijn er een aantal headerlijnen die bijkomende opties voorstellen. De entity body wordt gebruikt door de POST methode om gegevens door te sturen en wordt gescheiden van de headerlijnen 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 status code.

Het is dus duidelijk dat een HTTP bericht aanzienlijk groter zal zijn dan een CoAP bericht.



Figuur 3.4: *HTTP Message Format*

3.2 Communicatiemogelijkheden

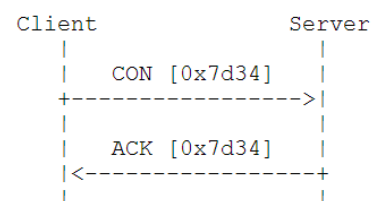
In deze paragraaf bespreken we de communicatiemogelijkheden van CoAP. We bekijken de variabele betrouwbaarheid van CoAP berichten en gaan na hoe het request/response model bij CoAP werkt aan de hand van voorbeelden.

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 implementeren indien nodig.

Confirmable berichten

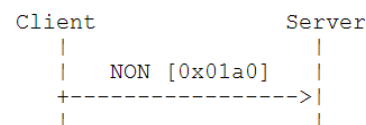
Wanneer we de betrouwbaarheid van de berichtenuitwisseling willen opdrijven, merken we de berichten als confirmable (CON). Een CON bericht moet door de server worden beantwoord met een Acknowledgement (ACK) bericht, dit ACK bericht moet hetzelfde message ID bevatten als het CON bericht waarop geantwoord wordt. Wanneer CON berichten niet worden beantwoord met een ACK bericht vóór een bepaalde timeout, wordt het bericht opnieuw verzonden. Bij het opnieuw verzenden wordt een exponential back-off mechanisme toegepast. Eerst wordt een timeout bepaald tussen een `ACK_TIMEOUT` en `ACK_TIMEOUT x ACK_RANDOM_FACTOR`, wanneer die timeout verstrijkt wordt het CON bericht opnieuw verzonden en de timeout verdubbeld. Wanneer de server niet in staat is het CON bericht te verwerken, wat betekent dat die zelfs geen geldige error response kan geven, antwoordt die met een reset (RST) bericht in plaats van een ACK bericht.



Figuur 3.5: *Betrouwbare berichtenuitwisseling (CoAP 14 draft)*

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-confirmable bericht (NON). 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. Wanneer een ontvanger niet in staat is het bericht te verwerken, opnieuw bedoelen we daarmee dat het geen geldige error response kan geven, zendt hij een RST naar de zender.

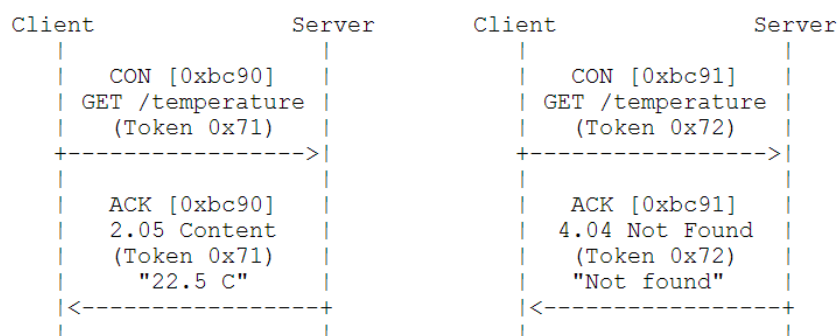


Figuur 3.6: Onbetrouwbare berichtuiwisseling (CoAP 14 draft)

3.2.2 Request/response model

Piggy-backed response

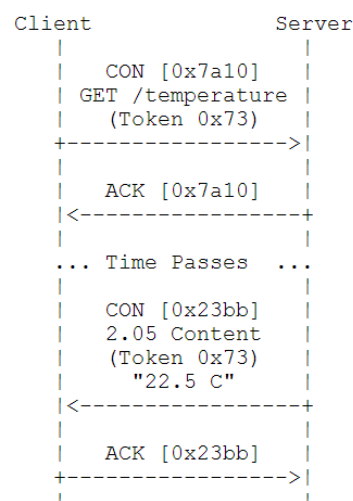
Wanneer een request met een CON bericht verstuurd wordt, is het mogelijk dat het antwoord meteen beschikbaar is bij de server. Indien dit het geval is, wordt het antwoord op de request meteen meegestuurd met het ACK bericht. Dit wordt een piggy-backed response genoemd. In Figuur 3.7 worden twee voorbeelden van GET requests met piggy-backed responses getoond. De ene is succesvol, de andere geeft een error response terug.



Figuur 3.7: Twee GET requests met piggy-backed responses (CoAP 14 draft)

Seperate 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 3.8).

3.3 Extra Features

3.3.1 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 te sturen naar de server dat een lege observe option bevat. Wanneer de client dan een ACK bericht terug krijgt met een observe option, 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. De server verwijdert de client dan uit de lijst van geïnteresseerden.

3.3.2 Resource discovery

3.3.3 Resource directory

Hoofdstuk 4

CoAP library module

In hoofdstuk 3 bekeken we beknopt hoe het CoAP protocol in mekaar zit en wat er allemaal mogelijk is. In dit hoofdstuk bekijken we hoe een eigen CoAP library in PHP werd ontwikkeld als Drupal module. De module stelt een Drupal gebruiker 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 is slechts functioneel en bijvoorbeeld te gebruiken in een andere module die dan de berichten kan omvormen tot een visuele representatie. Dit laatste is wat wij doen in de uiteindelijk te ontwerpen module, deze laatste wordt toegelicht in hoofdstuk 5.

4.1 Doel

Wanneer we trachtten CoAP berichten op te stellen en 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 een andere module. 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 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 moet kunnen

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 n functie is niet prioritair en dus bijkomstig. Het is uiterst belangrijk bij het ontwerp van deze library dat de gebruiker zich niet meer moet bekommeren om hexadecimale code die nodig is in een bericht, alle generatie van hexadecimale code zit vervat 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, te ondersteunen functies

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

- Opstellen van de header en token waarbij de gebruiker het type, de methode en het token opgeeft. De token lengte wordt afgeleid uit het token. Wanneer geen token gewenst is, kan de gebruiker een lege string meegeven.
- Een optie toevoegen aan het bericht waarbij de gebruiker slechts het optienummer en de waarde van de optie moet meegeven. De gebruiker hoeft zich hierbij geen zorgen te maken over het mechanisme waarbij de Option Delta gebruikt wordt, of over de lengte van de waarde, noch over de volgorde van de opties. Evenmin moet de gebruiker rekening houden met het feit dat wanneer de Option Delta of lengte van de waarde groter is dan 12, een mechanisme met extra bytes wordt gebruikt. Hier wordt allemaal rekening mee gehouden in de CoAP library.
- De payload toevoegen aan het bericht waarbij de gebruiker slechts de payload hoeft op te geven. Alweer neemt de CoAP library alle formaliteiten voor eigen rekening. 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 return waarde

van de functie die het bericht stuurt. Men hoeft zich niet te bekommeren om meerdere antwoorden die op willekeurige tijdstippen kunnen aankomen.

- Het sturen van een bericht met observe optie: Het antwoord dat hierop terug gestuurd wordt, bevat slechts één en is slechts een bevestiging dat de server de client heeft toegevoegd aan de lijst van geïnteresseerden. De server zal nu geregeld 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 4.2.3) hoe we deze probleemstelling in de library hebben opgevangen.

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 (versie 14).

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, handige functies

Sommige berichten worden 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 kunnen zijn. Dit zorgt ervoor een gebruiker bijvoorbeeld al een basisbericht kan opstellen en het daarna nog eigen wens kan aanpassen, wat

de gebruiker veel tijd 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 de kans krijgt om meteen een URI-path optie toe te voegen aan het bericht. De gebruiker hoeft hier enkel het desbetreffende URI-path toe te voegen. De optie wordt automatisch toegevoegd door de library op basis van de invoer. 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. Dit zorgt ervoor dat de server waarnaar het bericht wordt verstuurd, de client zal toevoegen aan de lijst met geïnteresseerden. 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. 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 mogelijk 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 2 klassen welke besproken worden in volgende puntjes.

CoAPFactory klasse

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 bericht object aanmaken geven een object terug van 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 subparagraaf 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 klasse

Deze klasse representeert het bericht zelf. Het bevat ten allen tijde het bericht in hexadecimale vorm. Het is deze klasse die de essentiële operaties (zie 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 dus een UDP socket geopend worden en in het geval van de observe functionaliteit moet de socket bovendien onderhouden worden, daar berichten later kunnen toekomen op initiatief van de server.

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 return waarde 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 kunnen 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 verkozen alternatief. De CoAP library definieert hierbij 3 hooks die een gebruiker moet implementeren. Één van de voordelen hiervan is dat de Drupal gebruiker normaal gezien al bekend is met het hook mechanisme van Drupal zelf. Het gebruikte mechanisme is namelijk analoog, de hook wordt geïmplementeerd door de naam van de module voor de hook te plaatsen. We sommen even de te implementeren hooks op, deze mogen leeg zijn:

- *hook_receive_response*: Deze hook wordt opgeroepen telkens wanneer een notificatie binnen komt. De CoAP library bouwt een response-object op van de klasse CoAP-Message 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 en 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. 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 gemaakt 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 het CoAP protocol gebruik maakt van UDP kunnen pakketten verloren gaan, dit moet dus opgevangen worden indien CON berichten verstuurd worden. In de CoAP draft (versie 14) wordt exponential backoff voorgeschreven. In de CoAP library werd die gecomplementeerd met de standaard waarden die eveneens in de CoAP draft (versie 14) 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 timeout waarde en een retransmission counter die bijhoudt hoeveel keer het CON bericht al opnieuw werd verstuurd. De timeout waarde 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 timeout 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 timeout waarde verdubbeld. De constanten werden in de CoAP library ingevuld met volgende waarden die voorgeschreven worden in de CoAP draft (versie 14):

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

Hoofdstuk 5

CoAP resource module

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 daarvoor technische kennis nodig te hebben van het CoAP protocol of programmatie in Drupal. We bekijken eerst globaal wat de module concreet moet aanbieden. Daarna bekijken we hoe de module in mekaar zal steken. Eens we dit weten, kunnen we overgaan tot de verschillende versies van de ontwikkelde module en de implementatie ervan.

5.1 Architectuur

5.1.1 Functionaliteit

De bedoeling van deze module is de eindgebruiker in staat te stellen om gemakkelijk sensoren toe te voegen aan zijn/haar website, ze te bekijken en te sturen. Belangrijk hierbij is dat elke Drupal gebruiker die een website maakt, deze kan gebruiken zonder extra kennis nodig te hebben over het CoAP protocol. Ook is het niet de bedoeling dat de eindgebruiker zelf nog moet programmeren in Drupal 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. Concreet houdt dit onder andere dat bij installatie van de module een content-type mee wordt geïnstalleerd, zodat de gebruiker slechts met enkele muisklikken het content-type kan toevoegen en onmiddellijk kan beginnen met het gebruik van de sensoren. Verder worden bij installatie ook alle benodigde databanktabellen gecreëerd.

De configuratie die de gebruiker moet doen zoals bijvoorbeeld het toevoegen van een resource, moet gebruiksvriendelijk verlopen. Hiervoor wordt gebruik gemaakt van resource discovery, een gebruiker hoeft enkel het IPv6-adres op te geven. De sensoren die aangesloten zijn op dat embedded device worden dan opgesomd voor de gebruiker, die dan de mogelijkheid krijgt de sensoren aan te vinken om ze toe te voegen aan de website.

Dit principe wordt zelfs verder doorgetrokken. Een gebruiker zal tevens de mogelijkheid krijgen om alle embedded devices in een subnetwerk op te vragen. Dit gebeurt aan de hand van een resource directory waarin alle well-known/core's van de embedded devices in het subnetwerk zich bevinden. De gebruiker krijgt dan de kans een embedded device te kiezen om dan resources te selecteren.

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, op voorwaarde dat de desbetreffende sensor de methode ondersteunt (dit wordt weergegeven bij het toevoegen van sensoren). Naast deze methodes kan een sensor ook nog observable zijn (zie hoofdstuk 3), de gebruiker moet dus ook in staat gesteld worden om sensoren te observeren.

Er moet een module ontwikkeld worden voor Drupal die de gebruiker in staat moet stellen sensoren te kunnen beheren, waarbij als onderliggend protocol CoAP wordt gebruikt. De module moet zo veel mogelijk out-of-the-box te gebruiken zijn, wat inhoudt dat zoveel mogelijk benodigdheden mee geïnstalleerd worden zonder extra tussenkomst van de gebruiker. Concreet betekent dit dat de benodigde content-types, zowel voor de gebruiker als de tabellen voor de databank, automatisch mee geïnstalleerd worden bij installatie van de module.

Het content-type aan gebruikerskant moet de mogelijkheid bieden om dynamisch aan te geven welke sensoren hij/zij wil beheren en in welk vorm de data moet aangeboden worden. De gebruiker kan zelf een URI opgeven, maar nieuwe sensoren moeten automatisch gedetecteerd worden en beschikbaar gesteld worden aan de gebruiker aan de hand van een resource directory.

Een gebruiker moet een waarde kunnen ophalen met een GET-request maar, indien de sensor dit ondersteunt, moet het ook mogelijk zijn zich in te schrijven bij de sensor om hem te observeren. Hierbij pusht de sensor op eigen initiatief de data naar de geïnteresseerden.

Wanneer een sensor de PUT-methode ondersteunt moet de gebruiker deze ook kunnen uitvoeren.

De gebruiker moet in staat zijn om:

- cores op te vragen;
- de vier REST-methodes op een resource aan te roepen;

5.2 Evolutie

We kunnen de evolutie van deze module opsplitsen in volgende stappen (niet klaar):

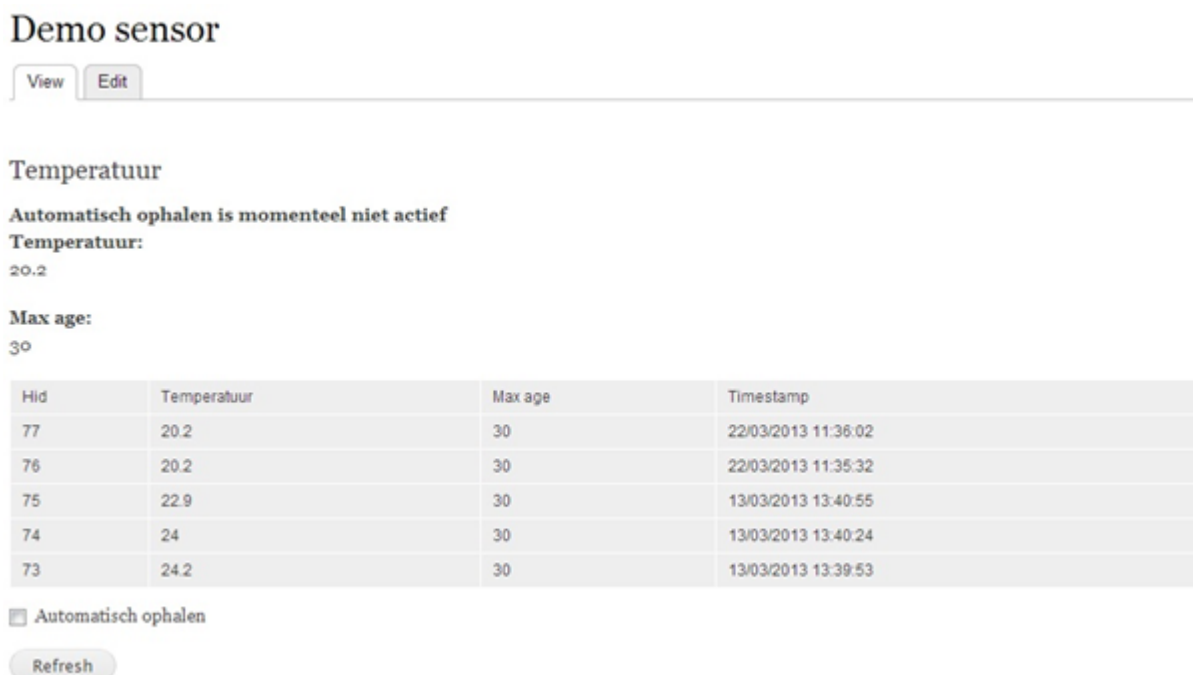
1. temperatuurmodule met HTTP/CoAP-proxy, GET en observe worden meteen ondersteund;
2. temperatuurmodule met native CoAP;
3. module maakt gebruik van externe CoAP library en ondersteunt verschillende content-types;
4. module ondersteunt resource discovery van n device en creëert een custom content-type bij installatie
5. module ondersteunt de drie andere REST methodes (PUT, POST en DELETE)
6. module ondersteunt resource discovery van meerdere devices en laat toe custom lijsten samen te stellen van resources verspreid over verschillende devices;
7. module ondersteunt het gebruik van een resource directory;
8. ...
9. finale module.

5.3 Temperatuurmodule met HTTP/CoAP-proxy

In deze paragraaf wordt besproken hoe de temperatuurmodule wordt 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.

De gebruiker klikt op een knop die het automatisch periodiek bevroegen van een embedded device initialiseert. 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 opgehaald en getoond aan de gebruiker.



The screenshot shows a web interface titled "Demo sensor". At the top, there are two buttons: "View" and "Edit". Below this, the section "Temperatuur" is displayed, followed by the status "Automatisch ophalen is momenteel niet actief". The current temperature is shown as "Temperatuur: 20.2". Below this, "Max age: 30" is indicated. A table with four columns (Hid, Temperatuur, Max age, Timestamp) lists recent data points. At the bottom, there is a checkbox for "Automatisch ophalen" and a "Refresh" button.

| 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 |

Figuur 5.1: *Temperatuurmodule met HTTP/COAP-proxy*

5.3.1 Proxy

Als proxy werd de coap.me website gebruikt van iMinds. Deze biedt de mogelijkheid om de 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.

5.3.2 Achtergrondprocessen

In een eerste fase van deze module werd er gebruik gemaakt van een formulier en werd bij het indienen van dit formulier, eerst de request uitgevoerd en dan gewacht op de response om de nieuwe pagina op te bouwen. Het spreekt voor zich dat dit geen goed oplossing is, daar de gebruiker zal moeten wachten op de pagina wanneer de communicatie traag is of misloopt, de gebruiker kan snel het idee krijgen dat er geen verbinding meer is met de website.

In het vorige hoofdstuk zagen we dat een mogelijke oplossing het gebruik van jQuery is. In deze situatie is jQuery echter geen goede optie, daar jQuery de databank van Drupal niet zomaar kan manipuleren. Hiervoor zouden een connectiestring, wachtwoord en dergelijke gegevens nodig zijn, en 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 in het verleden, 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, waardoor de gebruiker dus niet geconfronteerd wordt 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 opgeroepen worden met de response als argument.

5.3.3 Geschiedenis van opvragingen

Alles is nu voorhanden om een geschiedenis bij te houden van opgevragen waarden. Het achtergrondproces kan ongestoord op de server periodiek een GET-request uitvoeren en omdat dit op de server gebeurt kunnen de waarden gemakkelijk en dynamisch toegevoegd worden aan de databank.

Er rest ons nu enkel nog een manier te vinden om de client op de hoogte te brengen van

nieuwe waarden en hem van die waarden te voorzien.

Op deze manier staat het ophalen van waarden en die in de databank stoppen los van het tonen van diezelfde waarden aan de client. Dit zorgt ervoor dat een andere gebruiker evengoed ook de waarden kan bekijken, er is dus een n-naar-veel relatie.

5.3.4 Push-strategie: node.js

De mooiste oplossing en tevens n met het minste aandeel aan overhead, is een oplossing waarbij de server zelf op eigen initiatief data kan sturen naar de client. 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 node.js 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 free-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 wilt. Het is de bedoeling dat onze module zo veel mogelijk out-of-the-box bruikbaar is.

Vandaar dat wij geopteerd hebben geen gebruik te maken van node.js en dus ook niet van een push-strategie.

5.3.5 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.

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 terug stuurt.

Hetzelfde mechanisme wordt gebruikt in onze module. Wanneer de pagina geladen wordt bij de client, start een timer die periodiek een AJAX call uitvoert naar de Drupal server. Op de Drupal server is 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.

De callback-functie haalt dan de laatst toegevoegde rij op en stuurt volgende kolommen naar de client:

- Hid: een History-id die de opvragingen onderscheidt
- Temperatuur: de effectieve waarde in graden Celsius.
- Max age: De geldigheidsperiode van de waarde in seconden.
- Timestamp: Het tijdstip van opvragen.

In de jQuery-functie bij de client wordt dan het opgehaalde history-id vergeleken met de hoogste history-id in de tabel op de pagina. Als de opgehaalde waarde nieuwer is dan die op de pagina, wordt de nieuwe rij ingevoegd bovenaan de tabel op de pagina.

5.4 Temperatuurmodule met native CoAP

Hoofdstuk 6

Arduino

Het device zelf heeft ook een rol als proxy om de eindgebruiker het idee te geven dat de sensoren rechtstreeks aanspreekbaar zijn, terwijl eigenlijk het embedded device een vorm van controle invoert op het gebruik van kostbare bandbreedte. Een device kan namelijk meerdere sensoren omvatten (temperatuur, vochtigheid, etc). Bij het gebruik van deze module moeten sensoren automatisch gedetecteerd worden, wanneer deze fysiek op het embedded device worden aangesloten. Om de automatische detectie mogelijk te maken wordt gebruik gemaakt van een soort publieke directory, die de sensoren bevat onder vorm van een lijst. Wanneer een sensor aangesloten wordt op het device, moet de Drupal-module de eindgebruiker hiervan op de hoogte stellen en moet het mogelijk zijn om de sensor te configureren naar eigen wensen.

Bibliografie

- [1] K. Steenbergen, F. Janssen, J. Wellen, R. Smets, T. Koonen, “Fast wavelength-and-time slot routing in hybrid fiber-access networks for IP-based services”, in *IEEE LEOS Symposium*, Delft, The Netherlands, October 2000.
- [2] K. Nichols, V. Jacobson, L. Zhang, “A two-bit differentiated services architecture for the Internet”, *IETF RFC 2638*, July 1999.
- [3] <http://www.omniorb.org>
- [4] 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*. New York: Apress.

Lijst van figuren

| | | |
|-----|---|----|
| 1.1 | Internet of Things | 1 |
| 2.1 | Drupallogo | 6 |
| 2.2 | Wat is Drupal? | 7 |
| 2.3 | Basiswebsite van Drupal met Bartik-theme | 8 |
| 2.4 | Grafische weergave van hoe Drupal content aanbiedt. | 10 |
| 2.5 | Organisatie van Drupalmodules | 12 |
| 2.6 | Implementatie van hook_node_view door de module coap_resource | 13 |
| 2.7 | Weermodule | 15 |
| 3.1 | CoAP lagen (CoAP 14 draft) | 18 |
| 3.2 | Berichtformaat (CoAP 14 draft) | 18 |
| 3.3 | CoAP optie (CoAP 14 draft) | 20 |
| 3.4 | HTTP Message Format | 21 |
| 3.5 | Betrouwbare berichtuiwisseling (CoAP 14 draft) | 22 |
| 3.6 | Onbetrouwbare berichtuiwisseling (CoAP 14 draft) | 23 |
| 3.7 | Twee GET requests met piggy-backed responses (CoAP 14 draft) | 23 |
| 3.8 | GET request met seperate response (CoAP 14 draft) | 23 |
| 5.1 | Temperatuurmodule met HTTP/COAP-proxy | 36 |

Lijst van tabellen