Pervasive queries

Niels Clauwers

promotor :
Prof. dr. Kris LUYTEN

Eindverhandeling voorgedragen tot het bekomen van de graad
master in de informatica Human Computer Interaction

universiteit hasselt

Abstract

Deze thesis begint met een bespreking van een scenario waarin duidelijk wordt, dat werken in een pervasive omgeving voor verschillende problemen kan zorgen en pervasive querying een oplossing kan bieden. Dit scenario wordt als rode draad gebruikt doorheen de bespreking van de literatuurstudie.

Omdat deze thesis handelt over het queryen van een pervasive omgeving is het vanzelfsprekend dat we eerst beginnen met een kort overzicht over pervasive computing. We bespreken waaruit pervasive computing bestaat en welke problemen zich kunnen voordoen.

Daarna bekijken we enkele technologieën die gebruikt worden in het semantische web. Het semantische web is nauw verbonden met pervasive omgevingen. Het semantische web is een web van dingen die met elkaar verbonden zijn. De technologieën die hierin gebruikt worden, kunnen ook voor een pervasive omgeving gebruikt worden omdat hier de voorstelling van een "web" ook van toepassing kan zijn.

Ontologieën spelen een belangrijke rol in het semantische web en bij pervasive queryen. De samenwerking tussen verschillende ontologieën is dan ook een topic dat niet kan ontbreken. Er wordt hier besproken waar aandacht aan besteed moet worden indien er een samenwerking vereist is tussen verschillende ontologieën en hoe men deze samenwerking kan verwezenlijken.

Vervolgens bespreken we welke aspecten van het netwerk belangrijk zijn bij het pervasive queryen. We zien hier bijvoorbeeld waarom het netwerkgedeelte de grootste bottleneck is in een pervasive omgeving. Doordat het netwerk voor de grootste bottleneck kan zorgen, worden hier enkele optimalisaties besproken zodat het netwerk efficiënt gebruikt kan worden.

Ten slotte bespreken we de eigen implementatie waarin verschillende aspecten van de geziene literatuurstudie gebruikt worden. Hier worden ook enkele verbeteringen aangehaald die de implementatie ten goede zouden kunnen komen.

Voorwoord

Steeds meer mensen dragen verschillende technologische apparatuur. Ook worden de omgevingen verder uitgebreid met deze apparatuur. Kijk maar eens om je heen, hoeveel apparatuur er zich rondom je bevindt. Toekomstgericht wordt deze tendens alleen maar erger. We zijn zelfs op een punt gekomen dat we deze apparatuur niet meer zien, maar dat ze in de achtergrond verdwijnen. Hiervoor moet je maar eens naar het huis van de toekomst kijken.

Het grootste deel van deze apparaten worden voorzien om data op te slaan, andere apparaten aan te spreken, te verbinden met het internet enz. Er ontstaat dus een netwerk van computerapparatuur. We verliezen stilaan het overzicht van dit netwerk. Welke apparaten zitten er in mijn netwerk? Welk apparaat moet ik nu aanspreken? Hoe kan ik dat apparaat aanspreken? Dit zijn maar enkele vragen die een gebruiker zich kan stellen in een dergelijke pervasive omgeving.

Vandaar het onderwerp van mijn thesis: onderzoeken hoe we een vraag kunnen stellen aan een pervasive omgeving. Het belangrijkste aspect dat hierbij onderzocht moet worden is hoe we een vraag efficiënt kunnen oplossen, door alleen de noodzakelijke resources aan te spreken die de vraag volledig of gedeeltelijk kunnen beantwoorden. Om dit aan te pakken wordt gebruik gemaakt van een aantal technologieën die ook binnen het semantische web gebruikt worden.

De interesse voor netwerken en het web hebben er voor gezorgd dat dit onderwerp werd gekozen.

Clauwers Niels,
Student HCI
(UHasselt 2008-2009)

Dankwoord

Ten eerste wil ik mijn promotor Kris Luyten, co-promotor Karin Coninx en assistenten Geert Vanderhulst en Karoline Petermans bedanken voor de begeleiding tijdens mijn thesis.

Mijn vriendin Isabelle Hoebers, wil ik bedanken voor de steun en hulp bij het schrijven van deze thesis. Ook gaat er een dankwoord uit naar mijn familie, vrienden en collega's voor hun bijdrage en steun, zoals het testen van de implementatie.

Inhoudsopgave

1. Scenario 1
1.1. Vragen 1
1.2. Informatiebronnen 1
1.3. Pervasive Querying 2
1.4. Voorbeeld 2

2. Pervasive computing 4
2.1. Domein bespreking 4
2.2. Apparatuur 4
2.3. Communicatie 4
2.4. Interfaces 4
2.5. Beveiliging 5
2.6. Conclusie 5

3. Semantische Web 6
3.1. Uri 6
3.2. RDF 7
3.2.1. RDF Model 8
3.2.2. RDF Querying 8
3.3. Ontologie 9
3.4. Conclusie 10

4. Ontologie mediatie 11
4.1. Aanpak 11
4.1.1. Ontologie mapping 11
4.1.2. Ontologie alignment 12
4.1.3. Ontologie merging 13
4.2. Mediatie scenario's 13
4.2.1. Query rewriting 13
4.2.2. Instance translation 14
4.2.3. Instance mediation 15
4.3. Alignment beschrijving 16
4.3.1. Subsumption relaties 17
4.3.2. Value restriction 17
4.3.3. Type conversion 18
4.3.4. Path equation 18
4.3.5. Mapping patterns 19
4.4. Matching procedure 19
4.4.1. String distance metric 19
4.5. Voorbeelden 20
4.5.1. Semantisch gebied 20
4.5.2. Observer 21
4.6. Conclusie 23

5. Queryen 24
5.1. Gedistribueerd queryen 26
5.2. Architectuur 27
5.3. Query afhandeling 28
5.3.1. Mutant queries 30
5.4. Heterogene database systemen 32
5.4.1. Resource selection 33

5.5. Conclusie 38

6. Eigen implementatie 39
6.1. Omschrijving 39
6.2. Omgeving 39
6.3. Framework 40
6.3.1. Mediator 40
6.3.2. Resource service 43
6.4. Query afhandeling 43
6.4.1. Mediator 44
6.5. Interface 48
6.5.1. Client 48
6.5.2. Locaties 51
6.5.3. Rfid reader 52
6.6. Ontologieën 53
6.6.1. Merged ontologie 53
6.6.2. Local ontologie 53
6.6.3. Agenda ontologie 54
6.6.4. General ontologie 54
6.7. Mogelijke uitbreidingen 54
6.7.1. Optimalisatie 54
6.7.2. Automatisatie ontologie alignment 55
6.7.3. P2P 55
6.8. Conclusie 55

7. Conclusies 56

Lijst van gebruikte figuren

Figuur 1: Voorbeeld uri 7
Figuur 2: Voorbeeld rdf triple 7
Figuur 3: Uitbreiding resource 7
Figuur 4: rdf graaf 8
Figuur 5: SPARQL voorbeeld 9
Figuur 6: Ontologie mapping 12
Figuur 7: ontologie merging 13
Figuur 8:query rewriting 14
Figuur 9: Instance translation 15
Figuur 10: Instance mediation 16
Figuur 11: subsumption relatie 17
Figuur 12: Value restriction example 17
Figuur 13: Type conversion 18
Figuur 14: Path equation 18
Figuur 15: Mapping pattern [16] 19
Figuur 16: Ontology sphere [24] 21
Figuur 17: Observer architectuur [23] 22
Figuur 18: Observer query procedure [23] 22
Figuur 19: Query architectuur 24
Figuur 20: Query processing stappen [18] 24
Figuur 21: Interne data structuur [18] 25
Figuur 22: Gedistribueerd queryen 26
Figuur 23: Mediator 27
Figuur 24: Query shipping 28
Figuur 25: data shipping 29
Figuur 26: Hybrid shipping 29
Figuur 27: verwerking MQP [20] 30
Figuur 28: Boom van operatoren [20] 31
Figuur 29: vergelijking pipelined vs mqp [20] 31
Figuur 30: Wrapper 32
Figuur 31: Query graaf 34
Figuur 32: Index 34
Figuur 33: Statements 35
Figuur 34: Geen overlap 35
Figuur 35: Overlapping bij één resource 36
Figuur 36: Overlapping bij twee resources 36
Figuur 37: Replicatie van de data 36
Figuur 38: formule [26] 37
Figuur 39: formule [26] 37
Figuur 40:formule waarden [26] 37
Figuur 41: Rfid reader en tag 40
Figuur 42: Voorbeeld mapping 41
Figuur 43: Voorbeeld triple pattern 42
Figuur 44: Voorbeeld mapping 42
Figuur 45: Overzicht framework 42
Figuur 46: Resource information voorbeeld 43
Figuur 47: Voorbeeld query 43
Figuur 48: Predicaat triple pattern 1 44

Figuur 49: Alignments uit mapping document 45
Figuur 50: Subquery compatibel aan localOntology.....45
Figuur 51:Subquery compatibel aan generalDataOntology.....45
Figuur 52: Resource information voorbeeld.....46
Figuur 53: Algemeen rdf model.....46
Figuur 54: Query voor ontologie op te halen.....47
Figuur 55: Resultaat vertaling.....47
Figuur 56: Resultaat 47
Figuur 57: Userinterface client.....48
Figuur 58: Menu.....49
Figuur 59: Set ip optie 49
Figuur 60: Switch User optie 49
Figuur 61: Titelbalk met informatie 49
Figuur 62: Leave message interface.....50
Figuur 63: Bericht ontvangen 50
Figuur 64: Agenda functie 51
Figuur 65: Interface locaties 52
Figuur 66: Rfid reader interface.....52
Figuur 67: Overzicht merged ontologie.....53
Figuur 68: Overzicht local ontologie.....54
Figuur 69: Overzicht agenda ontologie 54
Figuur 70: Overzicht general ontologie.....54

1. Scenario

Brandweerkorpsen hebben vaak nood aan veel verschillende informatie die hen kan begeleiden om een ramp bij te staan. Bijvoorbeeld de ramp in Gellingen op 30 juni 2004. Als een dergelijke oproep binnenkomt bij de brandweer zijn er verschillende vragen die beantwoord moeten worden en liefst zo snel mogelijk. In volgende sectie worden een aantal vragen aangehaald die men zich kan stellen.

1.1. Vragen

De eerste vraag die men zich kan stellen: Wat is de aard van de ramp? Afhankelijk van het antwoord op deze vraag kunnen er bijkomende vragen gesteld worden.
Stel dat een huis of industriegebouw in brand staat. Zijn er giftige gassen die vrij kunnen komen? Liggen er bewoonde huizen rondom? Zitten er nog mensen vast in het gebouw? Zo ja, zijn dit mensen met een bepaalde eigenschap zoals doven of mindervalide? Moet de pers geïnformeerd worden?
Stel dat er een zwaar ongeluk gebeurd is op de snelweg. Hoeveel auto's zijn hierbij betrokken? Moet de snelweg misschien afgezet worden? Moet de politie ingeschakeld worden? Zijn er vrachtwagens bij betrokken die gevaarlijke bestanddelen vervoeren?
Bij beide voorbeelden kan het ook nuttig zijn om te weten of er manschappen nodig zijn die gespecialiseerd zijn in een bepaald domein (reddingen in brandende gebouwen, experts in gasexplosies, …). Is er bepaalde apparatuur nodig bij de interventie (snijmachine, luchtflessen, helikopter, …)?
Ook het aantal beschikbare manschappen en het aantal nodige manschappen kan belangrijke informatie zijn. Misschien moeten er extra manschappen/kazernes opgeroepen worden.

We zien dus dat er dringend nood is aan veel informatie. De antwoorden op deze vragen kunnen verkregen worden door een aantal informatiebronnen te raadplegen. Deze informatie is ook niet altijd in één en dezelfde database beschikbaar. Het kan dus zijn dat er verschillende gedistribueerde informatiebronnen gebruikt moeten worden. Het is zelfs mogelijk dat er ter plekke informatie gewonnen moet worden. Je kunt je voorstellen dat het niet altijd mogelijk is om op voorhand te weten hoeveel mensen er betrokken zijn bij een ramp. Ook wordt niet alle informatie op dezelfde manier opgeslagen. In volgende sectie wordt een overzicht gegeven van mogelijke informatiebronnen die gecontacteerd kunnen worden.

1.2. Informatiebronnen

Iedere kazerne houdt een lokale database bij waarin informatie wordt opgeslagen zoals: hoeveel manschappen er werken, hoeveel manschappen en apparatuur er beschikbaar zijn.

In vorige sectie hebben we ook gezien dat er nood is aan informatie zoals waar het ongeluk gebeurd is en welke gebouwen er rondom liggen. Ligt de ramp midden in een bewoond gebied of een industriegebied? Om een antwoord te geven op deze vraag kan er bijvoorbeeld een overzichtelijk plan getoond worden van het gebied zodat de operatoren duidelijk kunnen zien welke maatregelen er getroffen moeten worden. De informatie die nodig is om dit plan op te bouwen kan bijvoorbeeld gehaald worden uit een gemeentelijke database. Deze database houdt ook bij welke mensen in ieder huis wonen en of deze een bepaalde eigenschap hebben. Dit plan kan eventueel aangevuld worden met belangrijke leidingen en informatiepunten. Stel

dat er dicht bij een brand, een gasleiding loopt dan moet de leverancier gewaarschuwd worden om de toevoer af te sluiten.

Ook het weer kan een belangrijke rol spelen. Gaat het regenen of stormen? Van welke kant komt de wind? De windrichting is bijvoorbeeld van groot belang bij bosbranden. Deze informatie kan dan weer verkregen worden bij weerstations.

Het kan ook voorkomen dat er ter plekke pas geconstateerd kan worden hoeveel mensen bij de ramp betrokken zijn. Deze informatie kan eventueel door specialisten ingevuld worden op hun pda, gsm of laptop. Ook omstanders kunnen belangrijke informatie doorspelen aan operatoren die deze informatie opslaan in hun databanken.

Dit zijn een aantal voorbeelden van informatiebronnen die gecontacteerd kunnen worden om de brandweerkorpsen en eventueel andere korpsen, zoals de politie of ziekenhuizen, bij te staan bij een ramp. Ook is hier duidelijk geworden dat deze informatiebronnen niet allemaal op dezelfde manier gecontacteerd kunnen worden en ze ook niet alle informatie op dezelfde manier opslaan.

Er is dus nood aan een transparante manier van werken zodat bevoegde gebruikers deze informatie kunnen opvragen zonder dat ze zich moeten afvragen welke informatiebronnen ze moeten contacteren of hoe ze deze moeten contacteren. Alle technische details moeten verborgen blijven voor de gebruikers. Zij moeten enkel de relevante informatie op hun scherm krijgen.

1.3. Pervasive Querying

Pervasive querying kan voor deze transparante manier van werken zorgen. Pervasive querying maakt het mogelijk om een vraag, die geformuleerd is in een bepaalde taal, naar een mediator te sturen. Deze vraag wordt dan behandeld en de nodige informatiebronnen worden geraadpleegd. Uiteindelijk zal de gebruiker een antwoord op zijn vraag terug krijgen van de mediator.
Dit is een simplistische beschrijving van wat Pervasive querying doet. Doorheen de thesis zal duidelijk worden welke stappen er ondernomen worden en welke technologieën hierbij belangrijk zijn. Op het einde van de thesis zal de implementatie uitgelegd worden en wordt duidelijk welke stappen een query ondergaat, van vraagstelling tot antwoord.

1.4. Voorbeeld

Stel dat er een oproep binnenkomt van een hangaar die in brand staat. Via het adres van deze hangaar wordt er opgevraagd tot welk bedrijf deze hangaar behoort. Ook wordt er een plan opgevraagd van de omgeving. We komen er achter dat het bedrijf gevaarlijke stoffen produceert en in de hangaar opslaat. Verder zien we op het plan dat er op 5 kilometer afstand een bewoonbaar gebied bevindt. Bij het opslaan van dergelijke gevaarlijke bestanddelen en een bewoonbaar gebied op korte afstand moet er rekening gehouden worden met de gezondheid van de bewoners. De weertoestand van de volgende uren wordt opgevraagd en men kan constateren dat de wind in de richting van het bewoonbare gebied staat. Door deze informatie kan men de nodige media contacteren om de bevolking te waarschuwen, dat ze ramen en deuren gesloten moeten houden. Vervolgens wordt er gekeken welke brandweerkorpsen in de buurt gelegen zijn en of deze korpsen voldoende manschappen en apparatuur beschikbaar hebben om deze brand te blussen, rekening houdend met de

apparatuur die nodig is bij eventuele gevaarlijke bestanddelen. De korpsen worden gewaarschuwd en vertrekken onmiddellijk.

2. Pervasive computing

Samengevat betekent pervasive computing de aanwezigheid van computerverwerking zonder het besef van de gebruiker, maar op een manier dat het op een intuïtieve manier aangewend kan worden [27][28]. Alle apparatuur en resources zijn met elkaar verbonden en werken met elkaar, zodat er verschillende services aan de gebruiker kunnen worden aangeboden. Deze services kunnen dan geraadpleegd worden om een bepaalde taak af te handelen zonder te moeten nadenken over hoe of welke apparatuur of resources hij hiervoor nodig zou moeten hebben.

Deze beschrijving van pervasive computing kunnen we ook terugvinden in het scenario. In het scenario hebben we gezien dat het noodzakelijk is om op een transparante manier alle informatiebronnen te contacteren, om zo een antwoord te krijgen op een vraag. De gebruiker hoeft zich niet af te vragen welke resources hiervoor aangewend moeten worden en het gebruik hiervan.

2.1. Domein bespreking

Pervasive computing heeft betrekking tot drie overlappende gebieden in het ICT domein [29], namelijk:

- computing of apparatuur
- communicatie
- user interfaces

2.2. Apparatuur

Apparatuur kan in verschillende vormen voorkomen zoals een pda, gsm of embedded processoren. Ze kunnen onderverdeeld worden in drie categorieën:

- Sensoren, deze kunnen bepaalde zaken detecteren, bijvoorbeeld rfid reader.
- Processoren die berekeningen of verwerkingen uitvoeren.
- Actuators, deze apparaten reageren op een bepaalde input, bijvoorbeeld een projector die een beeld toont.

2.3. Communicatie

In een pervasive omgeving communiceren al deze apparaten met elkaar via een bepaald medium. Dit kan een draadloos (bv.BlueTooth of WiFi) of een bedraad (bv. Ethernet) medium zijn. Deze apparaten kunnen zo aangepast worden, dat de juiste vorm van het medium gebruikt wordt in bepaalde situaties.

2.4. Interfaces

Via de user interface kunnen gebruikers interageren met de pervasive omgeving. Het is het contactpunt tussen gebruiker en pervasive computing environment. Er onderscheiden zich drie vormen van human computer interaction, namelijk:

- actief
- passief
- of dwingend

Bij actieve interactie kunnen gebruikers bepaalde apparaten of resources rechtstreeks aanspreken via de user interface. In het scenario zien we ook dat een gebruiker een interface kan gebruiken om een vraag te stellen (bijvoorbeeld een plan opvragen waar de ramp zich bevindt. Deze vraag wordt dan achterliggend naar een mediator gestuurd die de vraag afhandelt en op zijn beurt de nodige informatiebronnen contacteert.

Passieve user interfaces verdwijnen in de omgeving, vaak beseffen gebruikers niet dat ze aan het interageren zijn met een dergelijke interface. We vinden passieve interactie bijvoorbeeld terug bij bepaalde vormen van rfid tracking. Gebruikers dragen een rfid tag, deze wordt door een rfid reader gelezen wanneer ze in de buurt komen en dit zonder dat de gebruiker het beseft.

Dwingende user interfaces zijn bijvoorbeeld interfaces die uit zichzelf reageren op een bepaalde input. Zo kan bijvoorbeeld een interface het licht doven in een ruimte als er niemand aanwezig is.

2.5. Beveiliging

In een pervasive omgeving waar alle apparatuur en resources met elkaar communiceren, is beveiliging een belangrijk punt. In deze sectie bespreken we dit punt kort.

Veel beveiligingsproblemen verschillen bijna niet van de problemen die zich kunnen voordoen bij traditionele computing, maar vloeien voort uit het gebruik van apparatuur met beperkte berekeningskracht die vaak in pervasive omgevingen gebruikt worden [28]. Apparatuur zoals pda's, gsm's, rfid tags, embedded processoren, etc. hebben beperkte cpu kracht, geheugen, bandbreedte, bereik en stroomvoorziening. Een voorbeeld als gevolg van deze beperkte berekeningskracht is encryptie. Een cpu kan te weinig kracht hebben om binnen aanzienbare tijd een zwaar geëncrypteerd bestand te decrypteren.

In het scenario van deze thesis kan het ook zijn dat er beveiliging nodig is. Het is bijvoorbeeld niet de bedoeling dat iedereen zomaar kan opvragen welke personen zich in een bepaald gebouw bevinden. Terwijl deze informatie wel van groot belang kan zijn voor brandweerkorpsen.

2.6. Conclusie

Pervasive querying vindt plaats in een pervasive computing environment. Daarom ook dit hoofdstuk waar pervasive computing kort werd toegelicht. De computing in een pervasive omgeving verdwijnt naar de achtergrond. De gebruiker interageert met deze omgeving zonder het te beseffen. Pervasive querying gaat deze omgeving aanspreken om een antwoord te krijgen op een bepaalde query. In het volgende hoofdstuk gaan we een aantal technologieën bespreken die hierbij gebruikt kunnen worden.

3. Semantische Web

Het onderwerp ‘Semantisch web’ wordt aangehaald omdat er een aantal technologieën gebruikt worden die ook bij pervasive querying gebruikt kunnen worden. Ook het begrip semantisch web slaat terug op deze thesis. Zo zal duidelijk worden dat een pervasive omgeving ook een web is van samenhangende dingen.

Het World Wide Web (WWW), zoals wij het vandaag kennen, linkt miljoenen, al dan niet heterogene documenten met elkaar. Je kunt het WWW zien als één grote database.
Als we een specifiek document of website zoeken kunnen we een zoekmachine raadplegen (bv. Google). Deze geeft aan de hand van gegeven zoektermen of sleutelwoorden een lijst van mogelijke relevante verwijzingen [1]. Het probleem is echter dat de computer niet weet welke semantiek er achter deze verwijzingen (in dit geval website of document) schuilt. Als hij dit wel zou weten, zou hij ons veel beter kunnen helpen met het zoeken naar informatie [2].

Semantiek is gerelateerd aan syntax. Je kunt een statement verwoorden in een bepaalde syntax, bijvoorbeeld in het Nederlands. Maar wat is nu de precieze betekenis van dit statement? Met behulp van het semantische web kunnen we aan een statement een semantiek hangen die verwerkt kan worden door een computer.

Het WWW is een web van documenten, het semantische web is een web van dingen. Deze dingen kunnen alles voorstellen van documenten, tot mensen, tot plaatsen etc. Het semantische web gaat deze dingen ook relateren met elkaar. We krijgen dus een web van informatie die verwerkt kan worden door computers [3].
Om dit alles te kunnen realiseren zijn er een aantal technologieën die ons hierbij helpen. In de volgende hoofdstukken worden een aantal van deze technologieën besproken.

3.1. Uri

Uri staat voor Uniform Resource Identifier en wordt binnen het semantische web gebruikt om resources te identificeren [4]. Een resource wordt gezien als het onderwerp naar waar we verwijzen en hiervoor gebruiken we een uniform systeem van identifiers.
Een url is een subvorm van uri en staat voor Uniform Resource Locator. Deze identifiers worden gebruikt om een website te bezoeken. Een url identificeert niet alleen een resource, maar zorgt ook voor de lokalisatie ervan.

Uri’s zijn gedecentraliseerd, niemand controleert het gebruik ervan. Iedereen kan een uri aanmaken voor eender welke resource. Er kunnen zelfs verschillende uri’s bestaan voor éénzelfde resource. Wanneer we kijken naar een url, verwijst deze naar de resource, maar vertelt ook hoe we deze resource kunnen bereiken. Een url doet dus dubbel dienst, wat een discussiepunt vormt binnen het semantische web (The semantic web identification problem [5]). Een uri geeft louter een naam,een identifier, voor een resource [4].

In het scenario wordt er gesproken over manschappen, uitrusting, locaties, etc. . Deze zaken kunnen allemaal geïdentificeerd worden met een uri. Stel dat er een manschap bestaat, genaamd Niels Clauwers, deze kan geïdentificeerd worden met een uri. In **Figuur 1** zien we de identificatie van deze persoon in rdf met behulp van de uri 'http://brandweerkorps.owl#Person1'.

```
<rdf:Description rdf:about="http://generalData.owl#Person1">
<j.0:hasSpeciality>None</j.0:hasSpeciality>
<j.0:hasName>Niels Clauwers</j.0:hasName>
<j.0:hasRfidTag>E004010008B1B4B40000</j.0:hasRfidTag>
<rdf:type rdf:resource="http://thesis/generalData.owl#Person"/>
</rdf:Description>
```

Figuur 1: Voorbeeld uri

3.2. RDF

In vorige paragraaf hebben we gezien dat we een resource kunnen identificeren via een uri. Door deze uri te gebruiken, maken we een referentie naar de desbetreffende resource. In het semantische web is het de bedoeling, dat de semantiek van een resource verwerkt kan worden door een computer. Daarom zou het gemakkelijk zijn als we over een resource zouden kunnen praten die door de computer verwerkt kan worden.
Rdf of Resource description framework zorgt ervoor dat je een statement kan opstellen die door een computer verwerkt kan worden [4]. Op **Figuur 1** zie je een voorbeeld waar rdf wordt gebruikt om een resource te beschrijven.

Rdf wordt in eerste instantie gebruikt om informatie weer te geven over web resources [6]. Op deze manier kan je dan metadata van de resource weergeven zoals de titel of de maker van de resource. Maar door een 'web resource' te generaliseren kan je ook metadata weergeven over andere resources die niet via het web te verkrijgen zijn. Je gaat met behulp van rdf een resource beschrijven door middel van eigenschappen en eigenschapwaarden. Dit gebeurt door gebruik te maken van uri's voor de resource, de eigenschappen en eigenschapwaarden. Door middel van een voorbeeld wordt deze uitleg concreet. Volgende statements halen we uit **Figuur 1**.

Stel we hebben het volgende statement "Person1 heet Niels Clauwers". Dit statement kunnen we voorstellen in een rdf triple. We krijgen dan bijvoorbeeld:

```
<http://thesis/generalData.owl#Person1>  <http://thesis/generalData.owl#hasName>  'Niels Clauwers'
```

Figuur 2: Voorbeeld rdf triple

Je ziet dat zowel het onderwerp (Person1) als de eigenschap (hasName) voorgesteld worden door uri's. De waarde wordt voorgesteld door een literal. We kunnen nog meer zeggen over de resource, je kunt bijvoorbeeld ook de specialiteit weergeven. We krijgen dan:

```
<http://thesis/generalData.owl#Person1>  <http://thesis/generalData.owl#hasSpeciality>  'None'
```

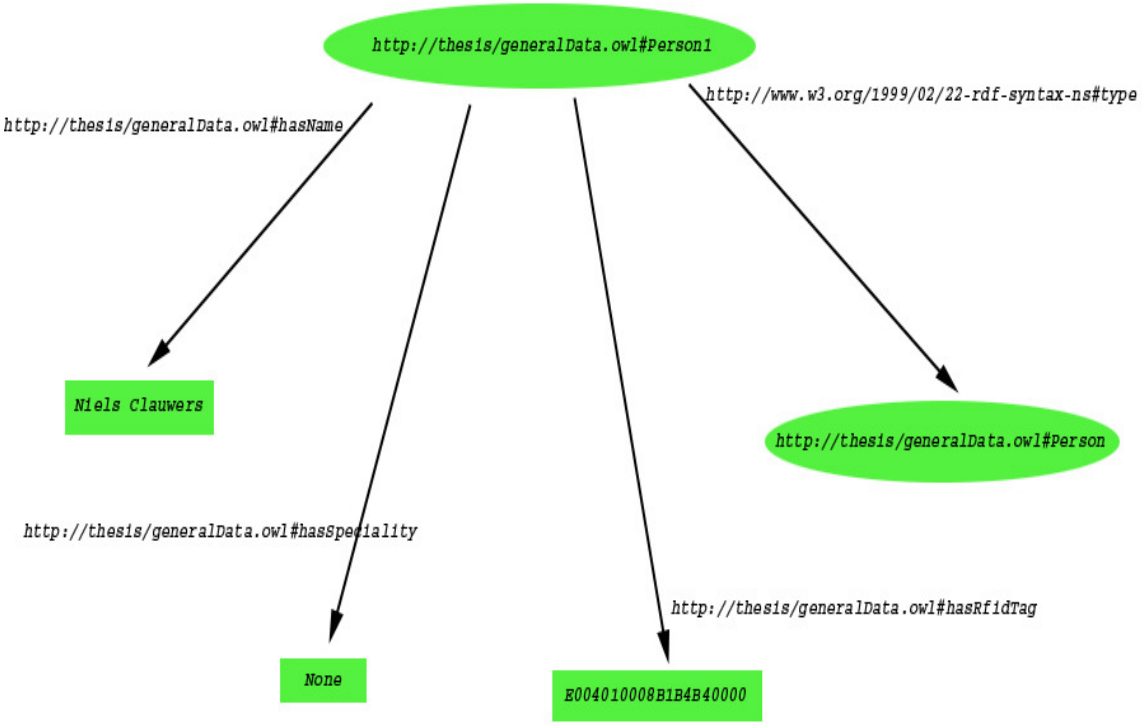
Figuur 3: Uitbreiding resource

Deze triples bestaan uit drie delen: een onderwerp, een predikaat en een object. De statements handelen over het onderwerp. De eigenschap of karakteristiek van het onderwerp is het

predikaat. De waarde van de eigenschap noemen we het object. Uit de vorige statements halen we dus het onderwerp ‘Person1’, de predikaten ‘hasName’ en ‘hasSpeciality’, en de objecten ‘Niels Clauwers’ en ‘None’.

3.2.1. RDF Model

In het rdf model worden de rdf triples voorgesteld als een graaf. Het subject en object worden voorgesteld als een node en het predikaat als een ark [6]. Voor het voorbeeld uit **Figuur 1** krijgen we volgende graaf:



Figuur 4: rdf graaf

Zo komt iedere statement overeen met één ark met in het begin en het einde een node. De objecten moeten geen uri’s zijn, maar kunnen ook voorgesteld worden door literals zoals in ons voorbeeld voor de naam of specialiteit. Uri’s worden voorgesteld door een ellips en literals worden voorgesteld door een rechthoek.

Doordat bepaalde objecten, zoals de klasse ‘Person’, voorgesteld worden als uri’s zijn dit ook resources en kunnen we deze graaf verder uitbreiden met meerdere rdf statements, waarbij deze objecten dan subjecten worden van de nieuwe statements. Zo zien we dat rdf bijdraagt tot het semantische web. Men kan via rdf gemakkelijk een web van dingen voorstellen.

3.2.2. RDF Querying

Rdf en uri’s zorgen ervoor dat we op een gedecentraliseerde manier kunnen praten over resources die verwerkt kunnen worden door computers. De verwerking van rdf houdt ook in dat deze door computers ondervraagd moeten kunnen worden. Hiervoor bestaat er een reeks

van querytalen o.a. ilrt squishql, Intellidimension RDFQL, RDFPath, triple, sparql, etc. [7]. De werking leggen we uit aan de hand van Sparql omdat dit een W3C [9] standaard is en deze taal ook in de implementatie van dit eindwerk gebruikt wordt.

Het rdf model is een gerichte, gelabelde graaf opgebouwd uit rdf triples [8]. Deze graaf kunnen we met sparql ondervragen door middel van een graaf pattern of query graaf. Deze basic graaf pattern kan je vergelijken met rdf triples. Het verschil is dat het subject, predikaat of object vervangen kan worden door een variabele. We krijgen een match wanneer bepaalde rdf termen uit de graaf vervangen kunnen worden door de variabelen. Het resultaat is een subgraaf. De syntax is afgeleid van de SQL syntax. Zo is er bijvoorbeeld ook een select en where clausule.

```
SELECT ?name
WHERE
{
  <http://thesis/generalData.owl#Person1> <http://thesis/generalData.owl#hasName> ?name.
}
```

Figuur 5: SPARQL voorbeeld

In Figuur 5 vind je een voorbeeld sparql query voor de rdf graaf uit Figuur 4. In deze query vragen we naar de naam (hasName) van 'Person1'. Het resultaat dat we terug krijgen is: 'Niels Clauwers'. In het voorbeeld zie je dat er een pattern wordt meegegeven waarvan het object een variabele is. Een match wordt gevonden indien deze pattern overeenkomt met een deelgraaf van de rdf graaf uit Figuur 4.

3.3. Ontologie

Een ontologie is een algemeen vocabulaire van een bepaald domein dat door onderzoekers gebruikt kan worden om informatie betreffende dat domein met elkaar uit te wisselen [10]. Het bevat definities van bepaalde concepten, die door een machine geïnterpreteerd kunnen worden uit dat domein, samen met de relaties tussen deze concepten.

Een ontologie wordt opgebouwd uit concepten van het domein. Deze concepten worden voorgesteld als klassen binnen de ontologie. Een klasse kan ook opgedeeld worden in subklassen. Deze subklassen zijn meer specifiek dan de superklassen. We hebben bijvoorbeeld een klasse 'fruit' een mogelijke subklasse zou dan 'rood fruit' zijn.
Deze concepten (of klassen) kunnen ook eigenschappen hebben, deze worden als properties, roles of slots beschreven binnen de ontologie. Eigenschappen kunnen ook beperkingen hebben, dit noemt men role restrictions of facets.
Van deze klassen en properties kunnen we instanties nemen. Een verzameling van instanties wordt ook wel de knowledge base genoemd van de ontologie. Van de klasse 'fruit' kunnen we bijvoorbeeld een instantie nemen, 'banaan' genoemd.

Als we terugkeren naar ons scenario zien we dat er verschillende soorten informatiebronnen zijn (kazerne, weerstation, gemeentelijke database, pda specialist). Voor iedere informatiebron kan er een aparte ontologie opgesteld worden omdat deze informatiebronnen andere termen kunnen gebruiken voor bepaalde resources, ze handelen dan ook over een ander domein. Een kazerne kan bijvoorbeeld data opslaan in verband met zijn manschappen. Voor een manschap kan hij de term 'Persoon' gebruiken. Een specialist kan bijvoorbeeld op zijn pda ingeven hoeveel betrokkenen er bij een ramp zijn. Voor deze betrokkenen kan hij ook de term 'Persoon' gebruiken. Hoewel deze ontologieën dezelfde termen gebruiken hebben ze

toch semantisch een andere betekenis. Zo zijn er voor alle informatiebronnen ontologieën opgesteld.
Indien een vraag beantwoordt moet worden, kunnen deze knowledge bases gecontacteerd worden. Om een query te kunnen sturen naar een bepaalde knowledge base, moet deze query de termen uit de desbetreffende ontologie gebruiken. Indien dat niet gebeurd zal er geen antwoord gegeven kunnen worden op de vraag. Een operator kan bijvoorbeeld vragen naar de 'windrichting' voor de volgende 5 uur. De lokale weerstations slaan deze informatie misschien niet op als 'windrichting' maar in het Engels als 'wind direction'. Wanneer de operator de vraag stelt: 'Wat is de windrichting voor de volgende 5 uur?' dan zal hier geen antwoord op gevonden worden omdat het lokale weerstation niet weet dat 'windrichting' hetzelfde betekent als 'wind direction'. Dit probleem kan aangepakt worden met behulp van ontologie mediatie wat besproken wordt in hoofdstuk 4.

3.4. Conclusie

In dit hoofdstuk hebben we besproken wat het semantische web precies voorstelt en welke technologieën hierbij gebruikt kunnen worden. We hebben ook gezien dat een pervasive omgeving eigenlijk ook een samenhangend web is van dingen. Een belangrijk onderdeel in het semantische web zijn ontologieën. In het voorbeeld uit sectie 3.3 waar de operator de windrichting wil opvragen, hebben we gezien dat er zich een mismatch probleem kan voordoen. Dit probleem kan aangepakt worden met behulp van ontologie mediatie dat besproken wordt in hoofdstuk 4.

Tot nu toe hebben we de omgeving en een aantal bruikbare technologieën besproken. De volgende logische stap is het queryen van deze omgeving, waar gebruik wordt gemaakt van deze technologieën. In de volgende hoofdstukken wordt dit uitgelegd samen met de problemen die zich kunnen voordoen.

4. Ontologie mediatie

Bij het queryen van verschillende databases bestaat er de mogelijkheid dat deze databases steunen op verschillende ontologieën (zie voorbeeld sectie 3.3). Het probleem, zoals in het voorbeeld van sectie 3.3, wordt ook wel het mediatie probleem genoemd.
Het mediatie probleem kan beschreven worden in één zin: Gegeven twee ontologieën die beiden een set van entiteiten beschrijven, vindt de relaties die bestaan tussen beide ontologieën [13] [15]. Teruggaand naar ons voorbeeld, bestaat er een relatie tussen de entiteit 'windrichting' en 'wind direction'.
Er is dus een noodzaak voor een taal die de semantische links tussen deze verschillende ontologieën beschrijft. Het document of resultaat van deze beschrijving kan dan gebruikt worden voor verschillende scenario's. In sectie 4.2 worden enkele voorbeelden gegeven.

4.1. Aanpak

Ontologie mediatie bestaat uit drie hoofdtakken [14], namelijk:

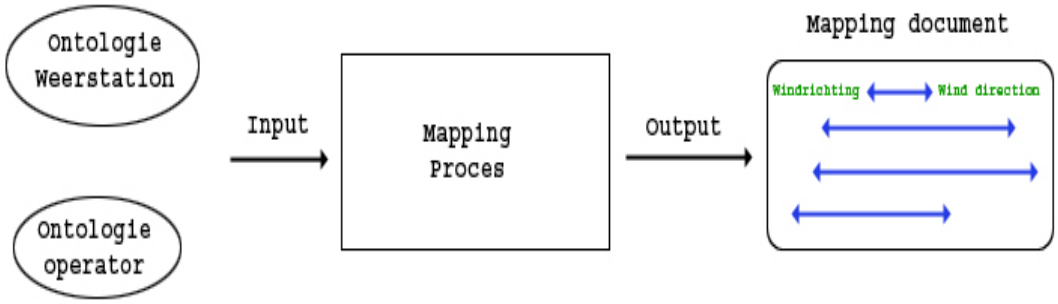
- ontologie mapping
- ontologie alignment
- ontologie merging

Bij ontologie mapping gaat het vooral om de overeenkomsten tussen de verschillende ontologieën vast te leggen. Ontologie alignment gaat deze overeenkomsten (half)automatisch proberen vast te leggen. Ontologie merging zorgt voor een nieuwe ontologie die gebaseerd is op de andere ontologieën. Ontologie alignment is dus een schakel die gebruikt kan worden bij ontologie mapping en merging.

4.1.1. Ontologie mapping

De output van het ontologie mapping proces is een ontologie mapping [14]. De input bij dit proces zijn twee ontologieën en de output is een specificatie van de semantische overeenkomsten. De eerste stap bestaat uit het vinden van semantische overeenkomsten. Deze worden dan gerepresenteerd in een mapping document met behulp van een mapping taal. In de laatste stap worden deze overeenkomsten gebruikt. In sectie 4.2 worden enkele voorbeelden gegeven waarvoor we een dergelijk mapping document kunnen gebruiken.

Het voorbeeld uit sectie 3.3 zou dan de ontologieën van het weerstation en de ontologie die de operator hanteert, kunnen gebruiken als input voor het mapping proces. De output van dit document kan dan gebruikt worden om de link te leggen tussen 'windrichting' en 'wind direction'. Met behulp van deze link kan gedetecteerd worden, dat de operator vraagt naar de 'wind direction' van de volgende 5 uur. Dit proces is gevisualiseerd op Figuur 6.



Figuur 6: Ontologie mapping

4.1.2. Ontologie alignment

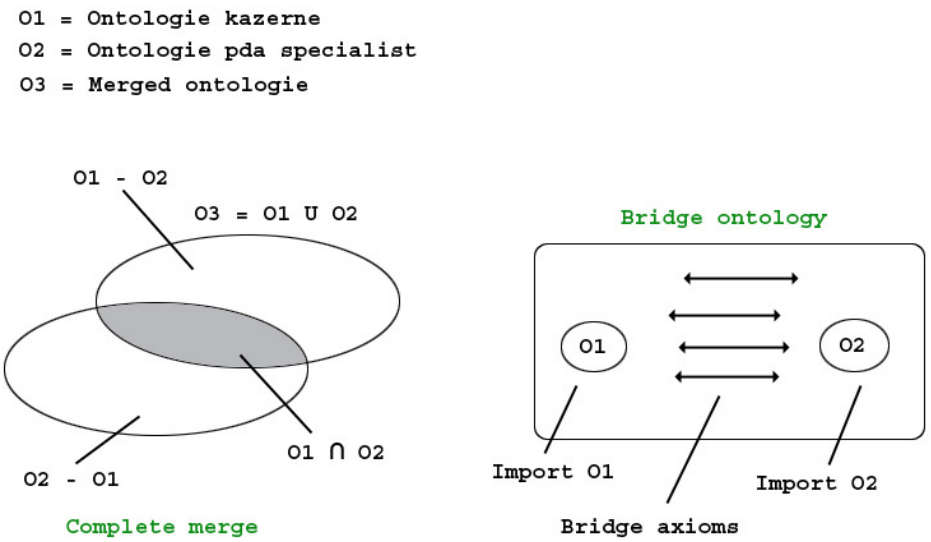
In 4.1.1 hebben we gezien dat de eerste stap het zoeken van overeenkomsten tussen twee ontologieën omvat. Het zoeken van deze overeenkomsten gebeurt via ontologie alignment. Deze stap wordt ook wel de match operator genoemd [14]. Deze operator zullen we ook terugzien bij de mediatie scenario's (sectie 4.2). De input van deze operator zijn twee ontologieën en de output zijn de overeenkomsten tussen deze twee ontologieën. Bij alignment komt het erop neer dat er een afstand berekend wordt tussen de entiteiten [15]. Een goede match is een match waarbij deze afstand zo klein mogelijk is.

Algoritmes die deze match operator implementeren zijn grotendeels ingedeeld in twee groepen, namelijk: schema-based en instance-based algoritmen [14].
Schema-based algoritmen kijken naar overeenkomsten tussen eigenschappen en relaties van de concepten uit de ontologieën.
Instance-based algoritmen gaan kijken naar de instanties die bij de concepten horen of hier overeenkomsten gevonden kunnen worden.
Verder wordt er nog een onderscheid gemaakt tussen element-level en structure-level matching [14].
Element-level matching controleert of er overeenkomsten zijn tussen properties of relaties van de concepten zoals de naam.
Bij structure-level matching wordt er gekeken naar overeenkomsten in de structuur van de ontologieën.

In [15] wordt er een andere opdeling gemaakt, die de opdeling uit [14] grotendeels aanvult. In [15] hebben we een string-based matching en een matching met een woordenschat.
Bij string-based kan men bijvoorbeeld kijken naar de labels van de entiteiten.
Terminologische matching met een woordenboek kijkt welke relaties er gelegd kunnen worden tussen een term en andere woorden uit de woordenschat zoals synoniemen of hyponiemen. We hebben ook nog interne en externe structuurvergelijking.
Bij interne structuurvergelijking wordt er naar de interne structuur gekeken van entiteiten. Zo kan men bijvoorbeeld kijken welke waarden ingevuld zijn voor een bepaalde property.
Bij externe structuurvergelijking wordt er gekeken naar de relaties van entiteiten ten opzichte van andere entiteiten.
[15] beschrijft ook een extentional vergelijking en een semantische vergelijking.
Bij extentional comparison wordt er gekeken naar de extensie van entiteiten (instanties).
Bij semantische vergelijking worden de interpretaties van entiteiten met elkaar vergeleken.

4.1.3. Ontologie merging

De output bij ontologie merging is een nieuwe ontologie die gebaseerd is op twee andere ontologieën. Deze nieuwe ontologie is de unie van de twee input ontologieën [14]. Er zijn twee verschillende aanpakken bij ontologie merging.
De eerste aanpak maakt een nieuwe ontologie van een aantal input ontologieën.
Bij de tweede aanpak wordt er geen nieuwe ontologie gecreëerd, maar worden de input ontologieën geïmporteerd en worden de overeenkomsten gespecificeerd door middel van bridge axioma's. Er wordt een soort view gecreëerd, genaamd de bridge ontology.
Op Figuur 7 worden beide aanpakken gevisualiseerd uitgaande van ons scenario.



Figuur 7: ontologie merging

4.2. Mediatie scenario's

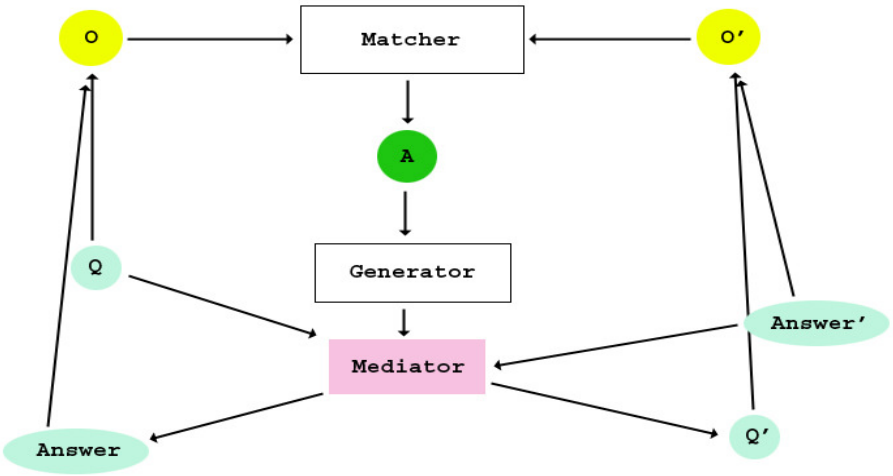
Er bestaan verschillende scenario's waarin ontologie mediatie nodig is [11]. In dit hoofdstuk beschrijven we enkele scenario's.

4.2.1. Query rewriting

Een applicatie kan naar een resource, die conform is aan een source ontologie o, een query q, sturen. Deze resource kan de query behandelen indien deze geschreven is in termen van de source ontologie o. Wanneer de applicatie echter dezelfde query wilt evalueren over een andere resource en deze resource is conform aan een andere ontologie o', dan moet de query q herschreven worden naar query q'. Deze nieuwe query moet geschreven zijn in termen van ontologie o'. Voordat we de query q kunnen herschrijven naar q' hebben we de overeenkomsten nodig tussen beide ontologieën o en o'. De overeenkomsten kunnen verkregen worden met behulp van een matching proces. De resultaten van een dergelijk matching proces is een alignment A. Op Figuur 8 ziet u dit proces weergegeven, toegepast op ontologieën die kunnen voorkomen in ons scenario. De resultaten die terugkomen van deze nieuwe resource, kunnen ook mediatie nodig hebben in de vorm van instance transformatie (zie sectie 4.2.3) of andere instance mediatie technieken [11].

Teruggaand naar ons voorbeeld waar de windrichting moet opgevraagd worden. Stel dat er twee weerstations zijn die beiden een andere term gebruiken voor ‘windrichting’. Indien de operator zijn vraag om de windrichting voor de volgende 5 uur naar waarstation A stuurt en deze nadien naar weerstation B wilt sturen, waar er een andere term voor ‘windrichting’ wordt gebruikt zoals ’wind direction’, dan moet deze query herschreven worden. Met behulp van ontologie mediatie kan deze query automatisch herschreven worden zodat de operator hier geen rekening mee moet houden en zijn oorspronkelijke query kan versturen.

O = Ontologie weerstation A
O’ = Ontologie weerstation B

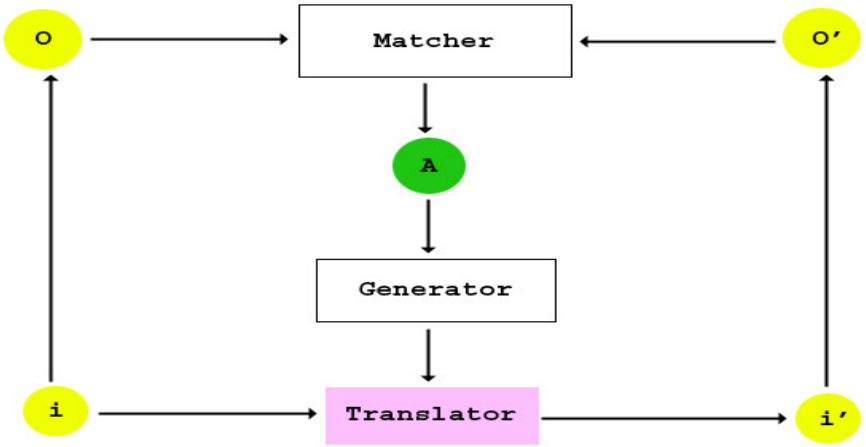


Figuur 8:query rewriting

4.2.2. Instance translation

Het kan zijn dat een applicatie een instantie van ontologie o moet gebruiken in een context met ontologie o’. Hiervoor moet de instantie vertaald worden naar concepten uit ontologie o’. Ook bij deze procedure wordt er gebruik gemaakt van een matching proces dat resulteert in een alignment A. Na het transformeren van een instantie van ontologie o naar o’ moet er nog gecontroleerd worden of de getransformeerde instantie al bestaat in de context met ontologie o’. Deze controle wordt opgevangen bij instance mediation [11]. Dit proces wordt weergegeven op afbeelding Figuur 9 toegepast op het voorbeeld van ons weerstation.

O = Ontologie weerstation A
O' = Ontologie weerstation B

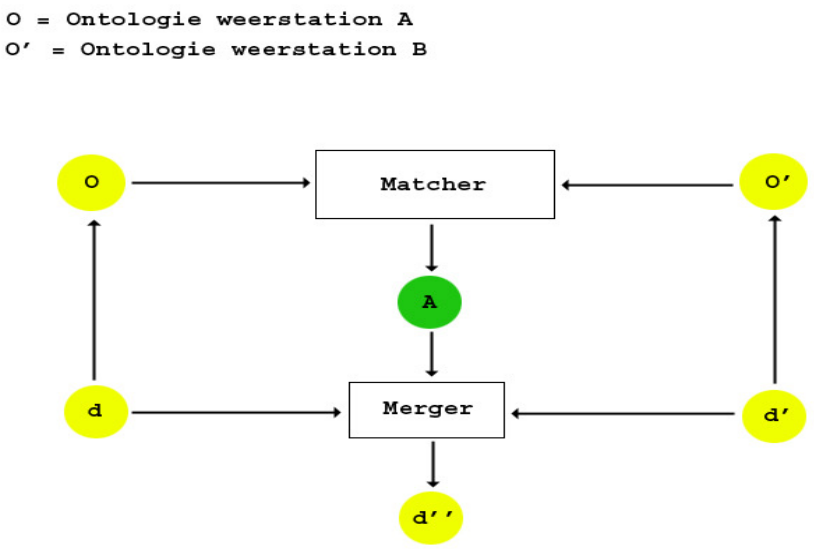


Figuur 9: Instance translation

Wanneer de operator uit ons voorbeeld de windrichting vraagt aan een weerstation die voor het element 'windrichting' het element 'wind direction' gebruikt. Dan gaat dit weerstation ook een antwoord terugsturen waarvan een instantie is genomen van het element 'wind direction'. We krijgen dan een antwoord in de aard van 'Winddirection with direction east'. De applicatie van de operator verstaat echter de term 'winddirection' niet en kan dit antwoord dus niet interpreteren. Deze klasse moet dus geconverteerd worden naar een term uit de ontologie die de applicatie wel verstaat. Via instance translation krijgen we dan een antwoord 'Windrichting met richting oosten'. Dit antwoord verstaat de applicatie wel.

4.2.3. Instance mediation

Zoals vermeld in sectie 4.2.2 wordt instance mediation gebruikt om te controleren of twee instanties naar dezelfde entiteit verwijzen. Deze procedure bestaat uit twee stappen [11]. De eerste stap bestaat uit de bevestiging dat twee instanties naar dezelfde entiteit verwijzen. In de tweede stap worden deze twee instanties gemerged naar één instantie waarbij de properties uit beide aparte instanties gecombineerd worden. Deze procedure wordt weergegeven in Figuur 10.



Figuur 10: Instance mediation

Ook hier wordt er weer gebruik gemaakt van een alignment A die voortvloeit uit een matching proces.

Ook dit laatste voorbeeld kunnen we mappen naar ons scenario. Stel dat een operator gegevens moet opvragen over een gebouw dat in brand staat. Door een bepaalde query op te sturen krijg hij gegevens over dit gebouw terug van verschillende knowledge bases. Knowledge base A stuurt een instantie terug van de klasse 'Gebouw' waarbij een aantal gegevens zijn ingevuld zoals de naam, het adres en het aantal verdiepingen. Knowledge base B stuurt ook een instantie terug van de klasse 'Gebouw' waarbij een aantal gegevens zijn ingevuld zoals het adres en het aantal inwoners. Hoe kan de applicatie, die de antwoorden ontvangt, weten dat deze instanties over hetzelfde gebouw gaan? Via instance mediation wordt dit gecontroleerd en worden de gegevens geaggregeerd. Dit kan bijvoorbeeld gecontroleerd worden door naar het adres te kijken. Indien beide adressen hetzelfde zijn kan men ervan uitgaan dat we over hetzelfde gebouw spreken.

4.3. Alignment beschrijving

In paragraaf 4.2 hebben we een aantal scenario's gezien waar ontologie mediatie nodig kan zijn. In deze scenario's wordt er gebruik gemaakt van een alignment beschrijving. Met deze beschrijving moet het mogelijk zijn om een aantal overeenkomsten expressief te kunnen weergeven [11].
In dit hoofdstuk worden een aantal vereisten aangehaald waaraan deze beschrijving moet voldoen. Een dergelijke alignment beschrijving moet ook aanwezig zijn in ons scenario om de overeenkomsten tussen de verschillende ontologieën vast te leggen. Deze vereisten worden in de volgende hoofdstukken duidelijk gemaakt met voorbeelden die zouden kunnen voorkomen in ons scenario. In deze voorbeelden maken we gebruik van ontologie A die de ontologie van het weerstation A voorstelt en ontologie B die de ontologie van het weerstation B voorstelt. Ontologie A maakt gebruik van de klasse 'Weer' en ontologie B maakt gebruik van de klasse 'Weather. Deze stellen de klasse voor die de toestand weergeven van het weer.

4.3.1. Subsumption relaties

De klasse ‘Weer’ en ‘Weather’ hebben beide een property voorzien om de windrichting aan te geven, namelijk: ‘windrichting’ en ‘wind direction’. Deze kunnen respectievelijk ingevuld worden met ‘noord’, ‘oost’, ‘zuid’, ‘west’ en ‘nord’, ‘east’, ‘south’, ‘west’. De property ‘windrichting’ kan ondergebracht worden onder property ‘wind direction ’ van ontologie B. In dit geval spreken we van een subsumption relatie. Deze relatie wordt weergegeven op Figuur 11.

Figuur 11: subsumption relatie

Deze figuur geeft weer dat voor iedere instantie x van de klasse ‘Weer’ uit ontologie A moet vertaald worden als een instantie van klasse ‘weather’ uit ontologie B. Verder zien we nog dat de waardes ‘noord’, ‘oost’, ‘zuid’ en ‘west’ vertaald moeten worden naar respectievelijk ‘nord’, ‘east’, ‘south’ en ‘west’.

4.3.2. Value restriction

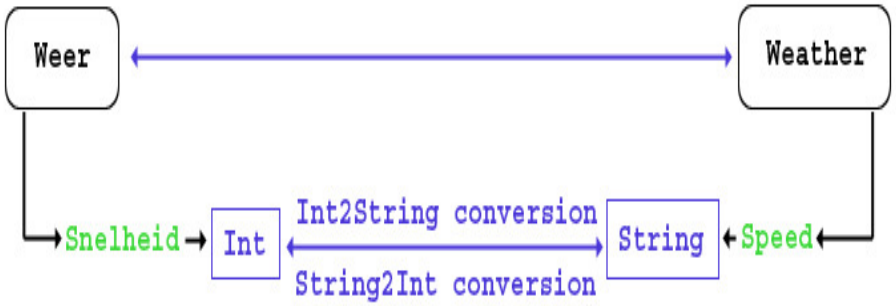
Verdergaand op ons voorbeeld van de weerstations. Stel dat we een instantie hebben van de klasse ‘krachtige wind’ uit ontologie A. Dit is een soort wind waarvan de snelheid hoger ligt dan 39 km/h. In ontologie B kunnen we een wind typeren als zijnde krachtig door de property ‘speed’ in te vullen met een waarde 39 of hoger. We beperken dus de instantie van klasse ‘weather’ uit ontologie B met behulp van een property waarde. De instanties uit ontologie B, van de klasse ‘weather’, kunnen gerelateerd worden met instanties uit ontologie A, van de klasse ‘krachtige wind’. Deze beperking gebeurd met behulp van een property waarde, vandaar de term value restriction. Op Figuur 12 wordt dit voorbeeld visueel weergegeven.

Figuur 12: Value restriction example

Deze figuur geeft weer dat iedere instantie van de klasse ‘krachtige wind’ van ontologie A, vertaald moet worden naar een instantie van klasse ‘weather’, van ontologie B, waarvan de property ‘speed’ is ingevuld met een waarde 39 of hoger.

4.3.3. Type conversion

De klasse ‘Weer’ in ontologie A heeft een property ‘snelheid’ die de snelheid van de wind voorstelt. Deze property wordt ingevuld met een integer waarde. De klasse ‘Weather’ uit ontologie B heeft ook een property die de snelheid van de wind voorstelt, namelijk ‘speed’. Deze property wordt ingevuld met een string waarde. In deze overeenkomst moet dus opgenomen worden dat het type int uit ontologie A geconverteerd moet worden naar een string type voor de ontologie B. Op Figuur 13 wordt deze overeenkomst weergegeven.

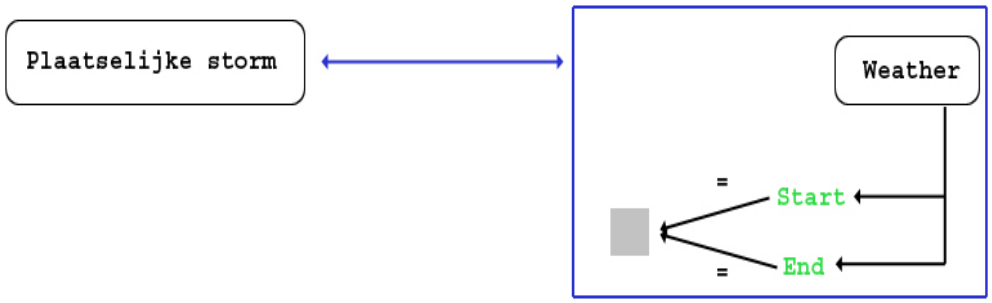


Figuur 13: Type conversion

Deze figuur geeft dus weer dat een waarde die ingevuld wordt voor de property ‘snelheid’ van het type Int is en dat je deze moet vertalen naar het type String indien je deze waarde wilt invullen voor de property ‘Speed’.

4.3.4. Path equation

Stel dat we in ontologie A een klasse ‘Plaatselijke Storm’ hebben. Deze klasse stelt stormweer voor dat ontstaat op een bepaalde locatie en eindig op dezelfde locatie. Een dergelijke klasse is niet beschikbaar in ontologie B. We kunnen de klasse ‘Plaatselijke Storm’ echter wel correleren met de klasse ‘Weather’ uit ontologie B door een beperking te leggen op de ‘Weather’ klasse. Indien we de properties ‘start’ en ‘end’ naar dezelfde entiteit laten verwijzen, kunnen we zeggen dat de instantie overeenkomt met een instantie van de klasse ‘Plaatselijke Storm’ uit ontologie A. Dit voorbeeld van een path equation wordt weergegeven op Figuur 14.



Figuur 14: Path equation

Alle instanties van de klasse ‘Plaatselijke storm’ kunnen vertaald worden naar een instantie van de klasse ‘Weather’ als er een beperking wordt gelegd op de properties ‘Start’ en ‘End’. De waarde die ingevuld wordt voor deze properties moeten verwijzen naar de zelfde entiteit. Deze entiteit kan bijvoorbeeld een instantie ‘Hamont ’zijn van de klasse ‘Locatie’.

4.3.5. Mapping patterns

In deze paragraaf hebben we gezien welke soorten relaties er gelegd kunnen worden tussen twee ontologieën. Net zoals bij software engineering hebben we ook hier patterns die beschrijven welke soort relaties er tussen twee ontologieën gelegd kunnen worden [14] [16]. Op Figuur 15 zien we een voorbeeld van een dergelijk pattern.

Name: Class by Attribute Mapping
Problem: The extension of a class in one ontology corresponds to the extension of a class in another ontology, provided that all individuals in the extension have a particular attribute value.
Solution: *Solution description:* A mapping is established between a class/attribute/ attribute value combination in one ontology and a class in another ontology. *Mapping syntax:* **mapping** ::= classMapping(*direction* A B attributeValueCondition(P o))
Example: classMapping(Human Female attributeValueCondition(hasGender “female”))

Figuur 15: Mapping pattern [16]

Deze pattern verteld dat de extensie van ontologie A overeenkomt met de extensie van ontologie B als en slechts als de instanties van de overeenkomende klassen een bepaalde waarde hebben van een attribuut.

4.4. Matching procedure

In sectie 4.1 hebben we gezien dat we een matching procedure nodige hebben die de overeenkomsten kan vastleggen tussen twee verschillende ontologieën. In deze sectie geven we hiervan een voorbeeld.
Er bestaan verschillende manuele aanpakken om een match vast te leggen tussen twee ontologieën. Er bestaan ook automatische methoden om een match vast te leggen. Toch zal de tussenkomst van de gebruiker altijd nodig zijn indien men een semantisch correctie overeenkomst nodig heeft. Deze methodes kunnen wel gebruikt worden om de matching procedure te vereenvoudigen. Ze kunnen bijvoorbeeld een aantal voorstellen geven die de gebruiker dan kan accepteren of aanpassen.

4.4.1. String distance metric

Deze methode wordt louter als voorbeeld gegeven. Er bestaan nog meer algoritmes die de afstand tussen twee klassen kunnen berekenen.

Bij deze methode is het de bedoeling dat klassen en properties van verschillende ontologieeën met elkaar worden vergeleken met behulp van een string distance metric. Dit wordt ook wel terminological matching genoemd [12]. Deze methode is niet heel doeltreffend, maar kan wel gebruikt worden als een eventuele initiële stap in het matching proces. Er wordt hier gewerkt

vanuit het standpunt dat gelijkaardige entiteiten gemodelleerd worden met een gelijkaardige naam. De oudere string distance metric methoden voldoen niet aan de vereisten die huidige systemen, die gebaseerd zijn op ontologieën, nodig hebben. Een string distance metric die bruikbaar is voor het matchen van ontologieën moet voldoen aan een aantal voorwaarden [12]:
Snelheid: De matching procedure moet snel uitkomst bieden. Zeker in realtime applicatie.

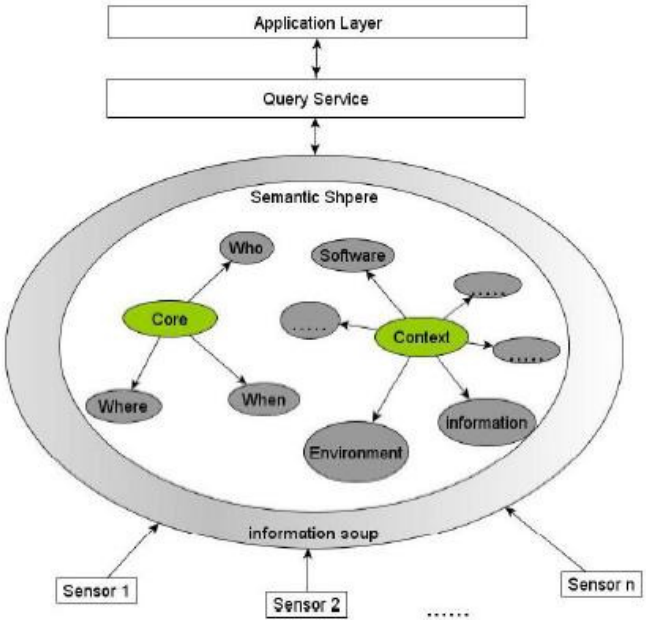
- Stabiel: String distance metric methoden maken vaak gebruik van een threshold. Deze threshold bepaalt of twee termen aan elkaar gelijk zijn of niet. Met stabiliteit wordt bedoeld dat een string distance metric methode optimale resultaten blijft leveren ook al verschilt de threshold een klein beetje van de optimale threshold.
- Intelligent: Het kan voorkomen dat een string distance metric twee semantisch verschillende termen met elkaar vergelijkt maar toch een grote overeenkomst berekent omdat de strings sterk op elkaar lijken. Neem bijvoorbeeld de namen “luis” en “muis”. Deze twee zijn semantisch heel verschillend, maar de strings verschillen maar één letter van elkaar.
- Discriminerend: Stel dat een term uit ontologie o overeenkomt met verschillende termen uit ontologie o’ volgens de string distance metric en dit met dezelfde distance metric. Een discriminante string distance metric methode zal er voor zorgen dat het zelden dezelfde waarde zal toekennen wanneer een term uit ontologie o vergeleken wordt met verschillende termen uit ontologie o’. Op deze manier komt het algoritme nooit of zelden voor het probleem te staan dat hij niet de juiste keuze kan maken.

4.5. Voorbeelden

In deze sectie geven we een tweetal voorbeelden, die het mediatie probleem aanpakken.
De eerste aanpak gaat zelf een ontologie opstellen waarop andere applicaties hun data kunnen inhaken. De tweede aanpak gaat terminologische relaties tussen de verschillende termen vast leggen. Je kunt het zien als een soort mapping document (zie 4.1.1).

4.5.1. Semantisch gebied

In [23] gaan we ervan uit dat er al verschillende ontologieën bestaan en er een samenwerking tussen deze ontologieën nodig is. In [24] gaan ze ook van het probleem uit dat gelijkaardige informatie anders voorgesteld kan worden in verschillende domeinen. De oplossing die ze hier voorstellen begint vanuit het standpunt van een core ontologie en is ook vooral toegespitst op pervasive omgevingen. Een overzicht van de architectuur uit [24] zie je op Figuur 16.



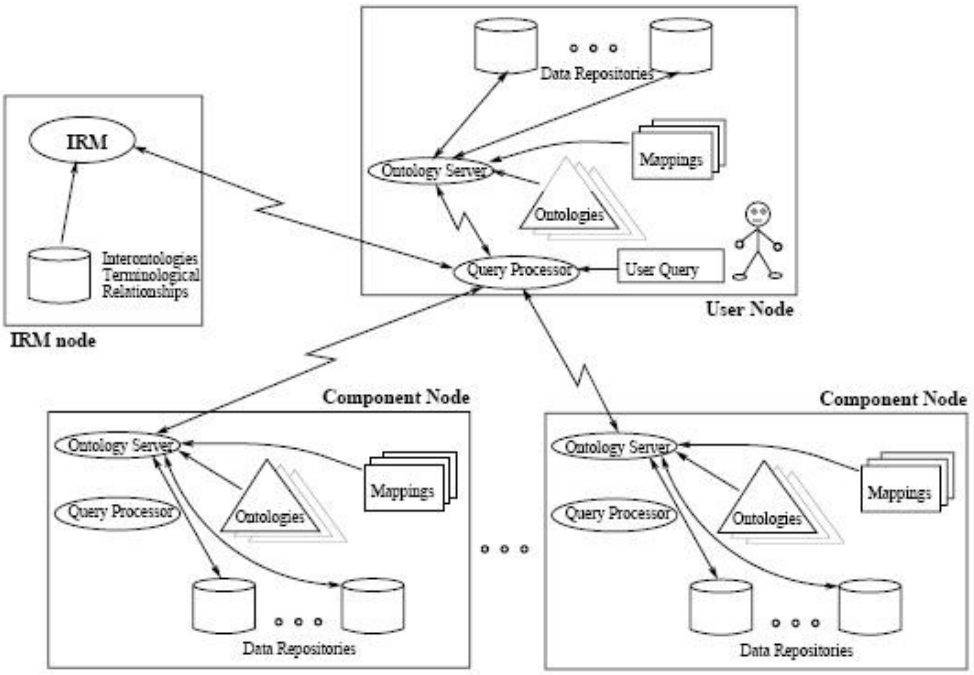
Figuur 16: Ontology sphere [24]

Bij deze aanpak vertrekken ze vanuit een core ontologie die bestaat uit een aantal fundamentele concepten, namelijk Who, Where and When. Deze core concepten kunnen dan door de verschillende sensoren en entiteiten in de pervasive omgeving gebruikt worden als hooks. De where ontologie wordt gebruikt om het concept van locatie binnen de omgeving te beschrijven. De when ontologie wordt gebruikt om het concept van tijd weer te geven en de who ontologie om het concept van een agent die zicht binnen de omgeving bevindt te beschrijven. Een voorbeeld van zo een agent is een sensor of een persoon. Wanneer een sensor nu data genereert haakt hij de data in deze core ontologieën. Een rdf reader die bijvoorbeeld een tijdstip samen met een tag id genereert slaat dit tijdstip op in de when ontologie en de tag id in de who ontologie. Andere soorten sensoren die gelijkaardige informatie genereren slaan hun data op dezelfde manier op. Deze manier zorgt ervoor dat er een samenwerking is tussen data van verschillende applicatie contexten [24].

4.5.2. Observer

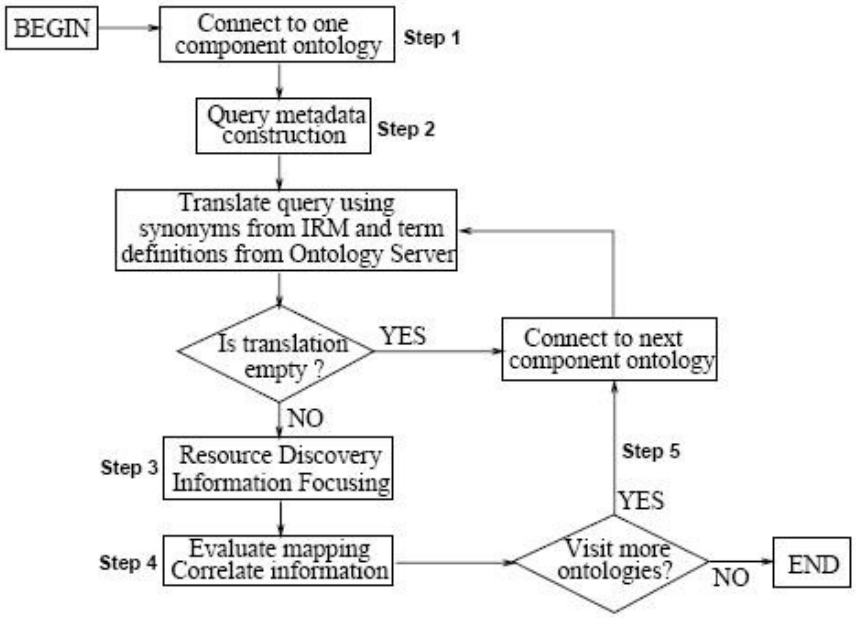
Het is ook mogelijk om een algemene ontologie te creëren die alle verschillende onthologieën van de verschillende repositories integreert in één ontologie. Deze methode is echter zeer complex en ook niet evident om de consistentie tussen de verschillende termen te behouden. Het is makkelijker om een loosely coupled approach te gebruiken. Deze aanpak zal dan de terminologische relaties tussen de verschillende termen vastleggen [23]. Het probleem wordt zo gereduceerd tot een probleem waar we alleen de synoniem relaties tussen de verschillende termen moeten kennen in plaats van alle verschillende termen in alle onthologieën.

Observer [23] is een loosely coupled architectuur die de boven beschreven problemen aanpakt. Op Figuur 17 zie je een overzicht van de Observer architectuur.



Figuur 17: Observer architectuur [23]

Een belangrijke component op deze afbeelding is de IRM. Irm staat voor Interontology Relationships Manager en zorgt voor de opslag van alle relaties die er bestaan tussen de verschillende termen. Op Figuur 18 zie je een overzicht van de query procedure die gevolgd wordt bij Observer.



Figuur 18: Observer query procedure [23]

In de eerste stap verbindt de gebruiker zich met zijn ontologie. Dit wil zeggen dat de gebruiker zich verbindt met de termen die binnen die ontologie gebruikt worden. In de tweede stap stelt de gebruiker zijn query op waarbij hij gebruik maakt van de terminologie uit zijn

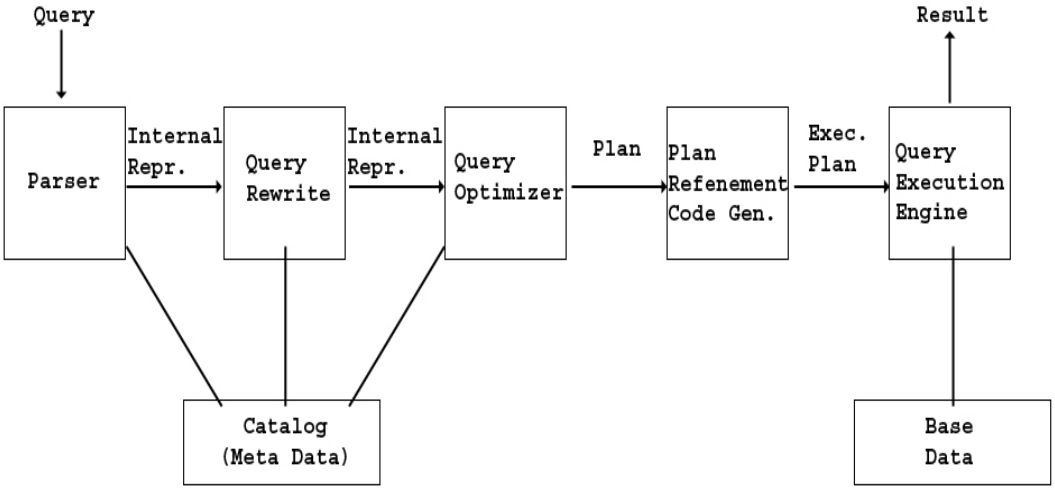
ontologie. Dan wordt er binnen de irm gekeken of er synoniemen bestaan voor bepaalde termen uit de query. Indien er synoniemen bestaan wordt de query vertaald zodat deze begrepen kan worden door de andere repositories die onderworpen zijn aan de desbetreffende ontologie. De volgende logische stap is het ophalen van de informatie uit de repositories. Indien de gebruiker aan het einde niet tevreden is met het antwoord kunnen er nog meer repositories bezocht worden indien deze aanwezig zijn.

4.6. Conclusie

In dit hoofdstuk hebben we gezien dat bij de communicatie tussen verschillende ontologieën verschillende problemen kunnen opduiken. Ontologie mediatie kan dan als oplossing dienen om deze problemen op te vangen. De volgende stap is het effectief queryen van de verschillende knowledge bases in de pervasive environment. De problemen die dan kunnen opduiken, zoals knowledge bases die op verschillende ontologieën gebaseerd zijn, kunnen dan opgevangen worden met behulp van ontologie mediatie. In de volgende hoofdstukken wordt het queryen en het gedistribueerd queryen besproken.

5. Queryen

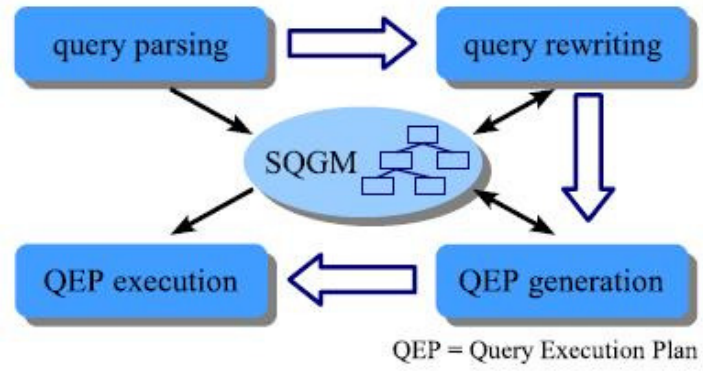
Voordat we beginnen met gedistribueerde queries, kijken we eerst naar de architectuur van query processing. De architectuur die we hier behandelen kan zowel voor relationele databases als rdf repositories gebruikt worden. Ook is dit niet de enige mogelijke aanpak, er bestaan nog andere mogelijke oplossingen [17].



Figuur 19: Query architectuur

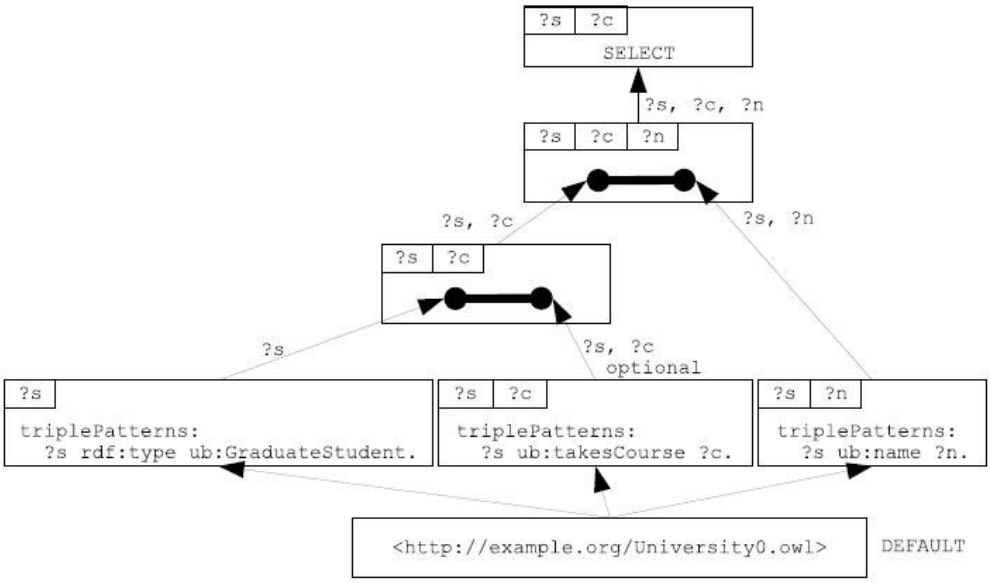
Op Figuur 19 zie je een overzicht van het verloop van een query. Dit verloop gebeurt in verschillende stappen waar we nu verder op in zullen gaan [17].

In de eerste stap wordt de query overgedragen naar een query parser. Deze parser gaat de syntax van de query controleren en de query omvormen tot een interne representatie. Deze interne representatie zorgt ervoor dat de query gemakkelijk behandeld kan worden door de opvolgende stappen. Deze interne datastructuur van de query is dan ook heel belangrijk en een goede structuur kan er voor zorgen dat een query sneller afgehandeld kan worden [18].



Figuur 20: Query processing stappen [18]

Op Figuur 20 zie je duidelijk hoe belangrijk de interne datastructuur (SQGM) is. Sqgm staat voor sparql query graph model en is de interne data structuur die door sparql wordt gebruikt.



Figuur 21: Interne data structuur [18]

Op Figuur 21 zien we een mogelijke voorstelling van een interne datastructuur die bij query processing gebruikt kan worden. Deze graaf voorstelling van een query bestaat uit noden en verbindingen. De noden stellen de operatoren voor en de verbindingen de data flow. Bij de noden heb je nog een onderscheid tussen providing operatoren en consuming operatoren. De providing operatoren zorgen voor de data waarop een consuming operator operaties kan uitvoeren. Deze consuming operatoren zorgen in hun beurt dan weer voor output data.

De volgende stap in de procedure is het herschrijven van de query. Dit herschrijven bestaat uit het uitvoeren van optimalisaties ongeacht de fysische staat van de database. Met fysische staat bedoelen we bijvoorbeeld de aanwezigheid van indices, de tabel grootte, etc. Mogelijke optimalisaties zijn bijvoorbeeld het verwijderen van overtollige predikaten, het ontnestelen van subqueries, etc. .

De derde stap bestaat uit het optimaliseren van de query. Deze optimalisaties zijn wel afhankelijk van de fysische staat van de database. Zo wordt er bijvoorbeeld bepaald welke indices er gebruikt worden. De uitkomst van deze stap zijn een reeks van query plannen. Met behulp van een cost estimation model wordt dan het beste plan uitgekozen. Een mogelijke cost estimation model hangt aan iedere operator een mogelijke kost. Deze kost kan bijvoorbeeld bestaan uit de cpu tijd die nodig is voor het behandelen van de operator, de transmissie tijd over een netwerk, etc. De maatstaf die gebruikt wordt door een dergelijk model om het beste plan te verkiezen wordt de objective function genoemd [19].
Deze kosten worden opgeteld en het goedkoopste plan wordt dan verkozen. In sectie 6.3.1 komen we terug op het optimaliseren van een query omdat dit toch wel een heel belangrijke stap is binnen het proces en zeker voor gedistribueerd queryen.

In de vierde stap wordt het verkozen plan omgevormd naar een uitvoerbaar plan. Dit kan bijvoorbeeld de generatie naar assembler taal zijn. In sommige gevallen worden er nog optimalisaties uitgevoerd die in de derde stap niet uitgevoerd zijn.

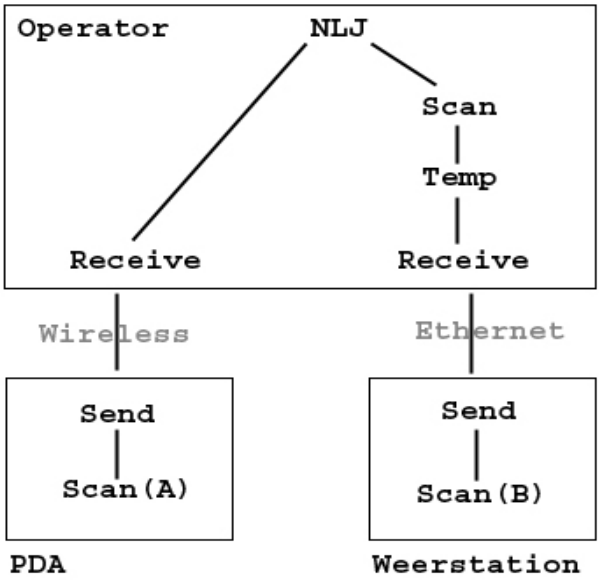
Op Figuur 19 zien we dat de query uiteindelijk terecht komt bij de query execution engine.

Deze engine bestaat uit implementaties om alle mogelijke operatoren (join, sort, scan, …) uit te kunnen voeren.

Figuur 19 toont nog één element dat niet bespoken is, namelijk de catalog. Binnen de catalog is allerlei informatie opgeslagen die gebruikt wordt om de query te parsen, te herschrijven en te optimaliseren. Deze informatie bestaat uit het schema van de database, fysische informatie van de database, etc.

5.1. Gedistribueerd queryen

In sectie 5 hebben we gezien hoe een query kan worden verwerkt tot een antwoord in een gecentraliseerd database systeem (lokaal). In deze sectie kijken we hoe een query behandeld kan worden in een gedistribueerde omgeving.
Het behandelen van een query in een gedistribueerde omgeving kan je vergelijken met het afhandelen van een query in een gecentraliseerde omgeving waarbij je rekening moet houden met enkele fundamentele verschillen. Zo moeten er eerst en vooral een soort van send en receive operatoren geïntroduceerd worden [17]. Deze operatoren zorgen ervoor dat er een data flow mogelijk is tussen de gedistribueerde sites. In ons scenario moet de operator ook verschillende gedistribueerde informatiebronnen contacteren. Sommigen moeten misschien draadloos (pda) gecontacteerd worden terwijl andere via het netwerk (weerstations) gecontacteerd kunnen worden.



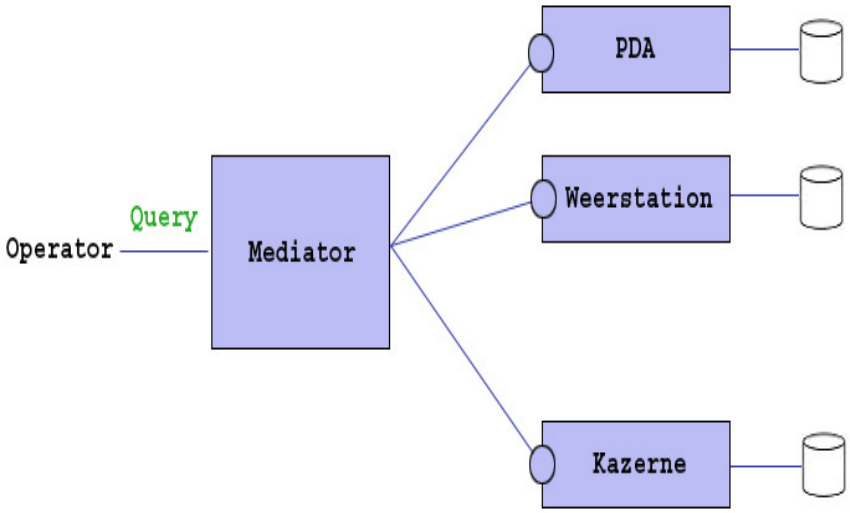
Figuur 22: Gedistribueerd queryen

Op Figuur 22 zie je dat data scans op de pda en bij het weerstation gebeuren. De resultaten van de scans worden naar de operator gestuurd die de rest van de operatoren op de data uitvoert. Deze send en receive operatoren worden vaak geïmplementeerd aan de hand van row blocking [17]. In de literatuur zal je vaak terug vinden dat het netwerk gedeelte, het zenden van berichten, vaak de bottleneck is binnen het gedistribueerd queryen. Row Blocking zorgt voor een verlichting van deze overhead. Zo worden records niet één voor één doorgestuurd maar in blokken. Op deze manier creëer je minder berichten die moeten doorgestuurd worden.

In tegenstelling tot gecentraliseerde query afhandeling moet er in een gedistribueerde omgeving beslist worden welke sites aangesproken moeten worden om op een efficiënte manier tot een antwoord te komen voor de query. Deze stap gebeurt binnen de query optimalisatie stap waar we nog op terugkomen (zie 6.3.1). Deze stap wordt ook wel het collection selection probleem [21] genoemd of database selection.

5.2. Architectuur

Stel je hebt verschillende gedistribueerde databases en een query voor deze databases. Een mogelijke oplossing om de query te behandelen zou zijn een kopie te nemen van de gedistribueerde databases en deze kopieën samen te voegen in een centrale database. De query kunnen we dan loslaten op deze centrale database. Deze aanpak wordt een geïntegreerd database systeem genoemd [19]. Wanneer bepaalde fragmenten van de originele database in één of meerdere aparte databases wordt opgeslagen spreken we over een parallel database systeem. Deze aanpak is vooral gunstig voor de performantie van de query afhandeling omdat delen van de query in parallel behandeld kunnen worden in de verschillende databases.
Nog een andere soort van gedistribueerde database systemen zijn de federated database systemen. Bij deze soort zijn er verschillende autonome database systemen die men kan aanspreken via één aanspreekpunt. Dit aanspreekpunt, ook wel de mediator genoemd, zorgt ervoor dat je een antwoord krijgt op de query door de verschillende databases aan te spreken op een transparante manier. Zowel de federated als het parallelle database systeem kunnen aan de hand van dezelfde gedistribueerde architectuur besproken worden [19].



Figuur 23: Mediator

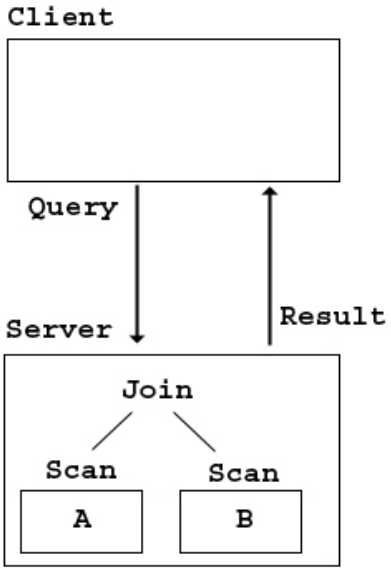
Op Figuur 23 zien we deze gedistribueerde architectuur die in ons scenario gebruikt wordt. We zien op deze figuur duidelijk de mediator die tussen de databases en de gebruiker staat. De mediator moet er voor zorgen dat de juiste databases gebruikt worden die mogelijke (gedeeltelijke) antwoorden kunnen bevatten voor de query. Hij moet de query ook opdelen en de juiste delen naar de juiste databases sturen. Uiteindelijk moet hij de resultaten mergen tot één antwoord. Om de query op een zo efficiënt mogelijke manier te kunnen behandelen kan de mediator gebruik maken van informatie over de verschillende databases. Hier komen we nog op terug wanneer we spreken over query optimalisatie (zie 6.3.1). Via de mediator krijgen we een transparante manier van werken. De operator moet alleen de mediator contacteren, van hieruit worden de gedistribueerde databases aangesproken. De operator moet dus geen

rekening houden met hoe en waar hij de gedistribueerde databases kan vinden. Hij moet ook geen rekening houden met hoe de databases hun data opslaan of welke databases gecontacteerd moeten worden om een antwoord te krijgen op de query.

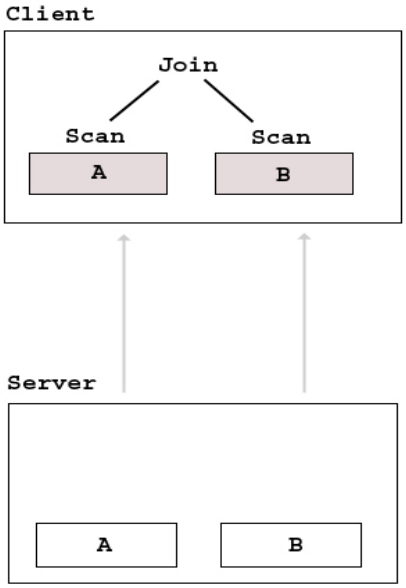
5.3. Query afhandeling

Een volgende vraag die we ons kunnen stellen is: “Waar verwerken we de query?”. De meeste voorkomende architectuur die gebruikt wordt is de client-server architectuur [17]. De cliënt zendt zijn aanvraag naar een server die dan een antwoord terug stuurt naar de cliënt.

De meest voor de hand liggende oplossing is query shipping. Bij deze methode wordt de query van de cliënt naar de server gestuurd. De server handelt de query af en stuurt een antwoord terug naar de cliënt. Deze methode is schematisch voorgesteld op Figuur 24.

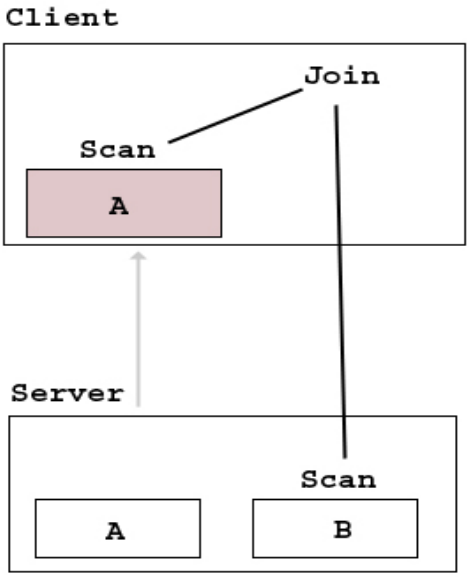


Figuur 24: Query shipping



Figuur 25: data shipping

Wanneer de data die nodig is om de query op te lossen wordt gestuurd naar de cliënt, spreken we over data shipping. Deze methode is schematisch terug te vinden op Figuur 22.
Bij data shipping wordt de data aan de cliënt kant in het geheugen of op schijf opgeslagen en bijgehouden voor eventuele opvolgende queries.
Er bestaat nog een methode die de voordelen van beide methodes samenbrengt. Op Figuur 26 zie je een schematische voorstelling van de hybrid shipping methode.



Figuur 26: Hybrid shipping

Bij de hybrid shipping methode kunnen de query operatoren zowel aan de cliënt kant als de server kant verwerkt worden en kan er data caching gebruikt worden aan de cliënt kant.

In het scenario wordt een client-server architectuur gebruikt, waar de client een query stuurt naar de server (mediator). Maar er komt nog een laag bij waar de mediator andere databases aanspreekt. De gebruiker (operator) heeft hier echter geen weet van.

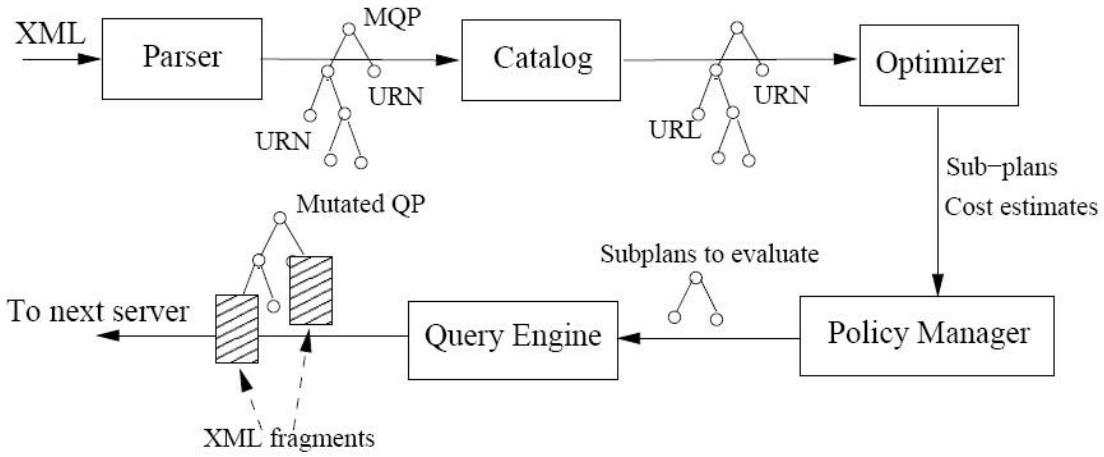
5.3.1. Mutant queries

Mutant queries worden hier uitgelegd om aan te tonen dat er nog meer manieren bestaan om queries op een gedistribueerde manier op te lossen. Deze uitleg is dus informatief en wordt niet gebruikt in de implementatie en ook niet in het scenario, hoewel het zou kunnen.

We hebben nu drie methodes bekeken over de manier van query afhandeling. De meest flexibele methode is hybrid shipping. Met deze methode kan de query makkelijker geoptimaliseerd worden omdat er gekozen kan worden waar bepaalde query operatoren uitgevoerd worden. We zijn hier dus niet gebonden aan één bepaalde verwerkingsmethode.

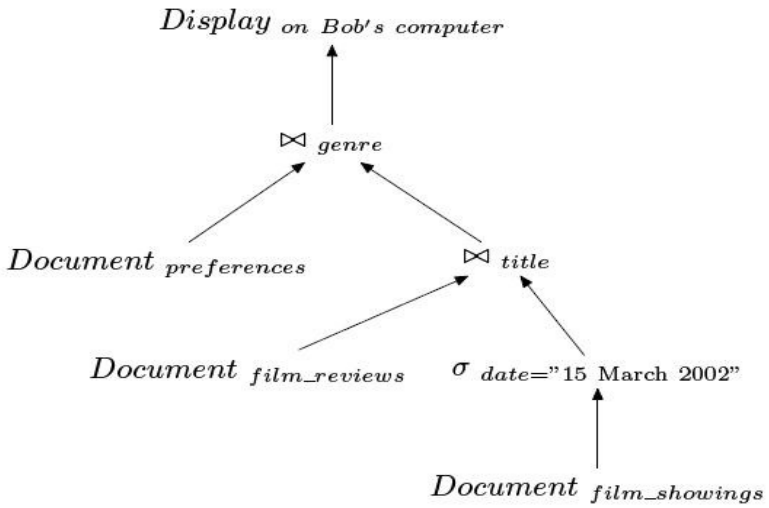
In sectie 6.3.1 gaan we nog zien dat er veel informatie, over de remote sites, nodig is om een goede query optimalisatie uit te voeren. Deze informatie moet ook geregeld geüpdatet worden en houdt ook geen rekening met het eventuele uitvallen van remote servers.
Tot nu toe hadden we altijd een centraal deel dat er voor zorgde dat heel het query plan juist uitgevoerd werd. Bij mutant query's wordt dit centrale aspect geschrapt en wordt ervoor gezorgd dat query's op een niet gecentraliseerde manier geoptimaliseerd kunnen worden [20]. Ook past deze methode zich aan, aan netwerk condities. Het uitvallen van een server zal er dus niet voor zorgen dat heel het query plan geannuleerd wordt.

Mutant query's zijn origineel ontwikkeld voor op het internet [20], maar kunnen ook geadopteerd worden voor andere soorten gedistribueerde netwerken. Ook bij mutant queries wordt er een query plan opgesteld. Deze wordt geserialiseerd in een xml document.



Figuur 27: verwerking MQP [20]

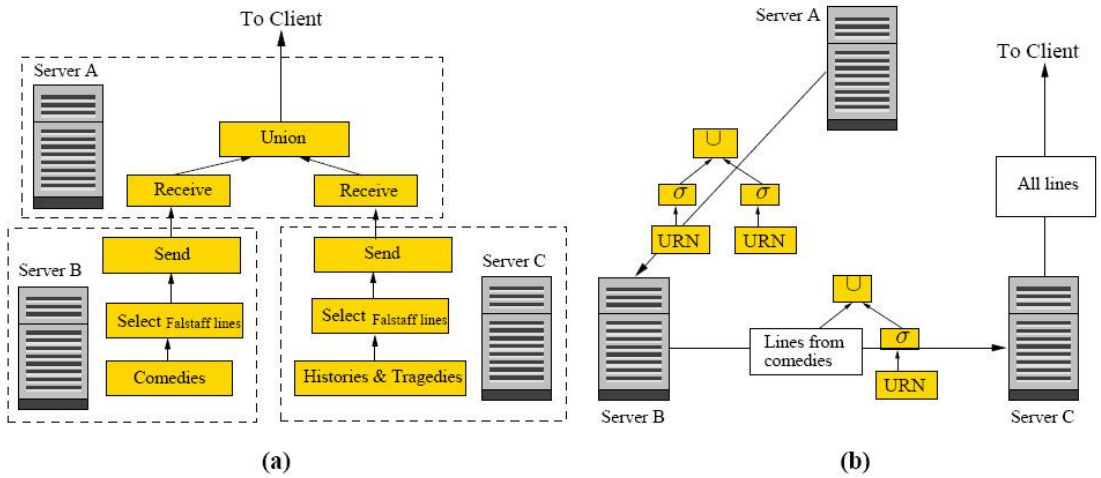
Op Figuur 27 kan je zien hoe een server een mutant query plan (mqp) verwerkt. In de eerste stap wordt het mqp geparsed naar een boom van operatoren en constante data [20]. Een voorbeeld van zo een boom van operatoren kan je terug vinden op Figuur 28.



Figuur 28: Boom van operatoren [20]

Zoals je ziet wordt er in de mqp gebruik gemaakt van urn's. Dit zijn verwijzingen naar documenten. Iedere server houdt in zijn locale catalog, informatie bij over de urn's, zoals waar ze de documenten kunnen terug vinden of welke servers meer weten over dat bepaalde document. De server haalt de documenten op waarvan hij de locatie kent en het plan wordt dan naar de optimizer gestuurd. Deze optimizer gaat het plan opnieuw optimaliseren, hij gaat subplannen genereren die lokaal geëvalueerd kunnen worden samen met hun kosten. De policy manager beslist dan of een subplan uitgevoerd gaat worden of niet. Hij kan bijvoorbeeld beslissen dat een plan niet uitgevoerd wordt omdat de server het op dat moment te druk heeft. Deze subplannen worden dan naar de query engine gestuurd die de subplannen evalueert. Deze subplannen worden binnen het query plan vervangen door hun resultaten. Op deze manier krijgen we een nieuw mqp die naar de volgende server gestuurd kan worden. De volgende server is dan een server die iets meer weet over één van de documenten naar waar verwezen wordt. Deze informatie is terug te vinden in de locale catalog van de server. Wanneer het hele query plan geëvalueerd is wordt het resultaat gestuurd naar het ip adres dat in de mqp staat.

Op Figuur 29 kan je een schematische vergelijking zien tussen de traditionele pipelined methode en de mqp methode.



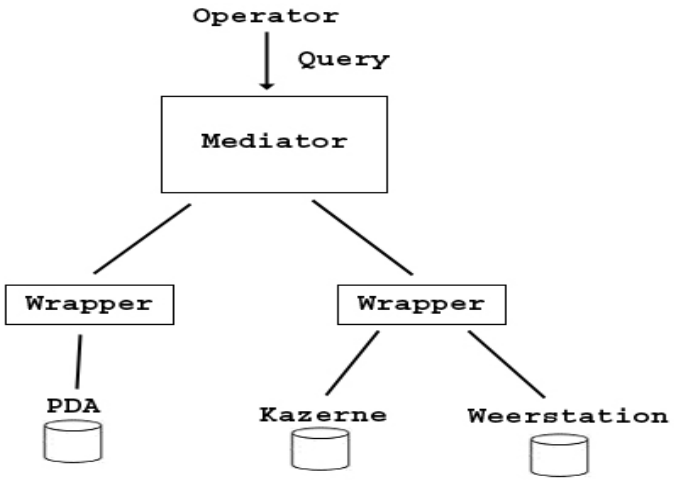
Figuur 29: vergelijking pipelined vs mqp [20]

Mutant query's zijn dus een mogelijke aanpak tegen de veranderende omstandigheden binnen een gedistribueerde omgeving. Dit is echter niet de enige methode, in [22] wordt de methode SwAP beschreven wat voor Scalable and adaptable query processor staat. Bij deze methode worden de operatoren in een query plan opnieuw geordend naar gelang veranderingen in het netwerk.

Zoals aangehaald in het begin van deze sectie wordt deze manier van aanpak niet gebruikt in de implementatie of het scenario. Met als gevolg dat ons aanpak niet let op de status van het netwerk. Indien de mediator uitvalt, kan de client geen queries meer versturen. Als er gedistribueerde databases uitvallen kan het zijn dat de client een leeg antwoord terug krijgt omdat hij niet alle nodige informatie kan opvragen die nodig is. Dit wil echter niet zeggen dat als één gedistribueerde resource uitvalt de client nooit een antwoord terug krijgt. De database die uitvalt, bevat niet altijd de nodige informatie en soms hebben andere gedistribueerde databases dezelfde data. Zolang er één database online is die de nodige data bevat zal de gebruiker een antwoord terug krijgen.

5.4. Heterogene database systemen

Tot nu toe hebben we het gehad over homogene systemen. Dit zijn systemen die dezelfde soort data behandelen en semantisch homogeen zijn. Bij heterogene systemen kunnen de databases andere soorten data behandelen [17]. Database A kan bijvoorbeeld afbeeldingen bewaren terwijl database B een relationele database is. Deze databases kunnen ook beperkingen hebben op het vlak vak uitvoerbare operaties. Database A kan bijvoorbeeld joins uitvoeren terwijl database B alleen maar kan scannen. Semantisch kan er ook een verschil zijn tussen de verschillende databases. Database A kan voor een bepaalde term een andere definitie hanteren dan database B, hier komen we op terug wanneer we het hebben over data integratie. Bij heterogene systemen kunnen de databases ook allemaal een andere api hebben voor ze aan te spreken.
Heterogene database systemen hebben inherent last van een aantal problemen. Deze problemen kunnen opgevangen worden door een wrapper. De wrapper zorgt ervoor dat deze heterogene databases transparant aangesproken kunnen worden.



Figuur 30: Wrapper

Er wordt een wrapper voorzien voor iedere database die een andere aanpak vereist. De wrapper vertaald de aanvragen van de mediator zodat deze begrepen wordt door de databases. Het resultaat van de database wordt dan weer door de wrapper vertaald zodat de mediator het resultaat verstaat. Wrappers spelen ook een belangrijke rol in het optimalisatie proces en kunnen complex zijn. Deze wrappers spelen dus als vertaler tussen de query zender en de database. Op Figuur 30 kunnen we zien hoe het in het scenario opgelost kan worden. De kazerne en het weerstation hanteren dezelfde aanpak en het volstaat dus om één wrapper te voorzien voor deze twee resources. De pda hanteert een andere aanpak en hiervoor moet dus een nieuwe wrapper voor voorzien worden.

Bij heterogene databases zijn er drie stappen die uitgevoerd moeten worden om een query te behandelen [26]. Eerst en vooral moeten de juiste databases geselecteerd worden die aangesproken gaan worden om een (gedeeltelijk) antwoord te verkrijgen op de. Dit wordt ook resource selection (zie 6.3.1) genoemd. Vervolgens moet de query voor iedere geselecteerde database vertaald worden naar een query die verstaanbaar is voor de geselecteerde database. Het herformuleren van de query hangt sterk af van het matchen van de ontologieën onderling (Zie 6.4). Deze herformulering van de query kan gebeuren door een wrapper of door de mediator.
De laatste stap houdt in dat de antwoorden van de databases gemerged moeten worden tot één antwoord.

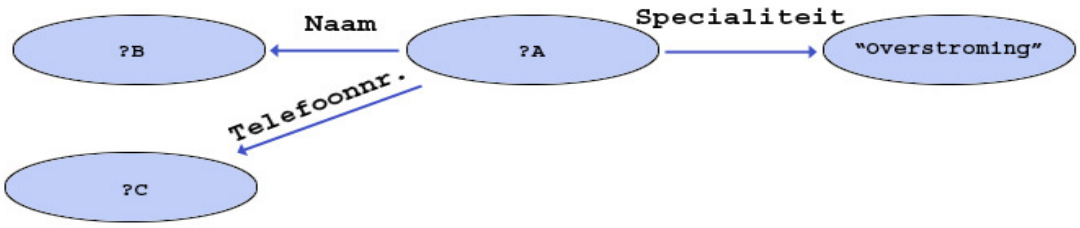
5.4.1. Resource selection

Het netwerk gedeelte in een gedistribueerde omgeving is de grootste bottleneck en dus ook de eerste kandidaat voor optimalisatie [19]. In de query optimalisatie stap worden er een aantal source queries gegenereerd. Source queries zijn queries die naar de gedistribueerde databases gestuurd worden met als bedoeling de resultaten van deze source queries samen te mergen tot een uiteindelijk antwoord. Om tot een goede optimalisatie te komen mogen de antwoorden van deze source queries niet leeg zijn. Er mogen ook geen antwoorden in zitten die niet tot het uiteindelijke antwoord behoren. En er mogen geen duplicaat antwoorden inzitten die bijvoorbeeld al gegenereerd zijn door andere source queries.

In hoofdstuk 3 hebben we het gehad over het semantische web waar rdf in besproken werd. Daarom tonen we in deze sectie een aantal optimalisaties voor het queryen van rdf databases. Dit wil natuurlijk niet zeggen dat deze gedachtegangen niet bruikbaar zijn voor bijvoorbeeld relationele databases.

5.4.1.1. Geen index

De makkelijkste optimalisatie is geen optimalisatie, deze methode wordt ook wel eens de naïeve methode genoemd [19]. In hoofdstuk 3.2 hebben we gezien hoe een rdf graaf is opgesteld en hoe het rdf model in elkaar steekt. Bij deze methode hebben we geen informatie beschikbaar die ons verteld hoe de rdf graaf verdeeld is over de verschillende gedistribueerde repositories. Het gevolg hiervan is dat we om een volledig antwoord set te verkrijgen van alle gegenereerde source queries, we alle triple patterns van de query graaf moeten evalueren over alle gedistribueerde repositories.



Figuur 31: Query graaf

Stel dat er ergens een overstroming plaatsvindt en een operator een specialist zoekt die deze ramp kan begeleiden. Op Figuur 31 zie je hiervoor een mogelijke query graaf. In deze query graaf zitten 3 triple patterns:

- (?A, Specialiteit, “Overstroming”)
- (?A, Naam, ?B)
- (?A, Telefoonnr., ?C)

Om nu een volledige antwoordset te verkrijgen moeten we iedere triple pattern evalueren over alle repositories. Indien we twee repositories hebben krijgen we uiteindelijk zes source queries die geëvalueerd moeten worden. We komen aan zes omdat iedere triple pattern geëvalueerd moet worden over iedere repository.

Bij deze methoden treden er twee problemen op. Enerzijds wordt iedere triple pattern van de query graaf geëvalueerd over iedere rdf repository. Deze methode is dus niet haalbaar bij een groot aantal repositories. Anderzijds worden er geen source queries gecombineerd die voor dezelfde repository bedoeld zijn. De joins worden allemaal uitgevoerd bij de mediator. Deze aanpak is dus ook niet haalbaar voor grote input queries. Deze aanpak is dus niet aan te raden voor ons scenario omdat er hier veel repositories kunnen zijn.

5.4.1.2. Property Index

Deze resource selection methode maakt gebruik van een index van triple patterns [19]. Deze index houdt bij welke repositories mogelijke instanties hebben van een triple pattern. We hebben al gezien dat een triple pattern bestaat uit een object, predikaat en subject. We kunnen in deze index een lijst bijhouden van alle mogelijke predikaten samen met de mogelijke repositories. Een mogelijke index kan er dan uitzien zoals op Figuur 32.

Property	Remote Repository
Naam	1, 2
Telefoonnr.	1, 2
Specialiteit	2

Figuur 32: Index

Als we de query graaf uit Figuur 32 gebruiken komen we tot een totaal van vijf source queries die geëvalueerd moeten worden. Dit is een vermindering van één source query. Dit is niet zo heel veel, maar naar maten het aantal predikaten en repositories stijgt, kan dit snel toenemen.

We kunnen deze methode nog verder uitbreiden door niet alleen het predikaat op te nemen in de index maar bijvoorbeeld ook het object. Door een combinatie van object en predikaat kunnen we tot een nog preciezere index komen.

Met deze optimalisatie hebben we dus het eerste probleem aangepakt van de naïeve methode. Niet iedere triple pattern wordt geëvalueerd over iedere repository.

5.4.1.3. Graaf index

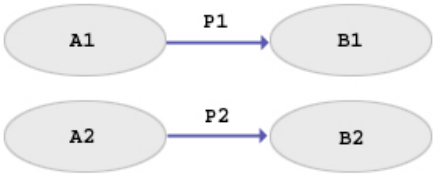
In paragraaf 6.3.1.2 hebben we gezien dat we met behulp van een index tabel de graaf query kunnen opdelen in verschillende source queries die dan naar de desbetreffende repositories gestuurd kunnen worden. Bij ontvangst van de resultaten worden deze gejoint met elkaar tot een volledig antwoord. We zien bij deze aanpak dat er geen moeite gedaan wordt om eventueel joins door de repositories te laten verwezenlijken.
Bij deze optimalisatie wordt er hiervoor wel een poging gedaan. Om deze optimalisatie succesvol uit te kunnen voeren is er extra informatie nodig in de catalog, namelijk inter-store overlap informatie [19].

Voordat we kijken naar een voorbeeld om de joins door de repositories uit te late voeren, kijken we eerst wat een inter-store overlap precies is.
In Sectie 3.2 hebben we gezien wat een statement precies betekend binnen RDF. Een statement kan je zien als een instantie van een triple pattern. Bij gedistribueerde rdf data zijn alle statements gefragmenteerd over de verschillende repositories. De vraag is nu welke statements we met elkaar kunnen linken [19]. Stel we hebben twee statements namelijk statement S1 en S2 (zie Figuur 33).

Figuur 33: Statements

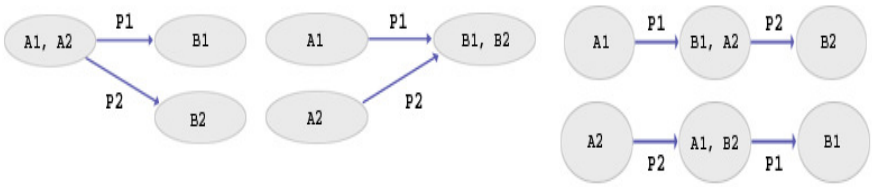
Deze twee statements kunnen op verschillende manieren met elkaar gelinkt worden tot één graaf.

Bij geen overlapping krijgen we het volgende resultaat (zie Figuur 34).



Figuur 34: Geen overlap

Wanneer er één resource uit iedere statement met elkaar overlapt, heb je vier verschillende mogelijkheden (zie Figuur 35).



Figuur 35: Overlapping bij één resource

Bij een overlapping van twee resources zijn er twee verschillende mogelijkheden (Figuur 36).

Figuur 36: Overlapping bij twee resources

Er bestaat ook de mogelijkheid dat er niet alleen een resource overlapping is, maar ook een property overlap. We krijgen hier dezelfde scenario's als voorheen. Alleen bij de overlapping van twee resources in combinatie met een property overlap spreken we over een replicatie van de data over de repositories (zie Figuur 37).

Figuur 37: Replicatie van de data

Wanneer één van de voorgaande overlapping plaatsvindt binnen dezelfde repository spreken we over een intra-store overlap [19]. Wanneer de overlap plaatsvindt over meerdere repositories spreken we over een inter-store overlap [19].

Als alle data overlappingen van de repositories in de catalog beschikbaar zijn, kunnen we kijken welke triple patterns uit de graaf query we kunnen joinen en welke overbodig zijn [19]. Met voldoende instantie overlap informatie in de catalog kunnen we dus veel onnodige queries schrappen en enkel de nodige queries naar de repositories sturen.

5.4.1.4. Cori

De Cori methode [26] wordt hier kort toegelicht omdat dit een automatische methode is om aan resource selection te doen. Deze wordt echter niet gebruikt in de implementatie van deze thesis. Het is ook niet echt haalbaar deze methode in een pervasive omgeving te gebruiken, maar dit wordt later duidelijk.
Deze methode gaat ervan uit dat een query gesteld wordt over een target schema met attribuut waarden A1, . . . ,Aq. Een query bestaat dus uit een aantal attributen met een waarde aanvast gekoppeld, q = {A1 = v1, . . . ,Aq = vq}. Alle attributen A_i komen uit het schema. Verder zijn er ook een aantal gedistribueerde resources waarop men een query kan loslaten. Alle resources hebben hun eigen schema die ook weer uit een aantal attributen bestaan. Deze resources kunnen eenvoudige of complexe queries verwerken. Een simpele query bestaat uit een reeks waarden bv. $q_s = \{v_1, ..., v_m\}$ en een complexe query bestaat uit een reeks attributen waar een waarde aanvast gekoppeld zit zoals query q.

De Cori methode gaat ervan uit dat er een aantal sample record beschikbaar zijn van de verschillende resources. Deze sample records worden bekomen door een random query los te laten op de resources. Op deze manier kunnen we een benadering van de content van de resource berekenen. Dit wordt ook wel eens de information resource sampling genoemd. Dit is ook meteen de reden waarom de Cori methode niet haalbaar is voor een pervasive omgeving. Stel je een omgeving voor met honderden sensoren, dan is het niet mogelijk om alle databases van deze sensoren te queryen voor sample records.

De volgende stap in deze methode is het berekenen van een resource score voor iedere resource. Deze score zegt hoe dicht een resource gerelateerd is met de query. De volgende formule is een aangepaste versie van de Cori methode [26]. Neem de volgende query:

$$q = \{A_1 = v_1, \ldots, A_q = v_q\}$$

Deze complexe query wordt eerst vertaald naar een simpele query q'. Voor alle resources wordt nu een score berekend die de mate waarin de resource gerelateerd is met de query voorstelt. Kort samengevat is een resource sterk gerelateerd met een query als de resource veel termen uit de query bevat. Wanneer een term voorkomt in veel resources dan is dit geen goede term om een onderscheid te maken tussen een gerelateerde resource of een niet gerelateerde resource. De formule voor de berekening van deze score ziet er als volgt uit (zie Figuur 38).

$$G(q, \mathcal{R}_i) = \frac{\sum_{v_k \in q'} p(v_k|\mathcal{R}_i)}{|q'|}$$

Figuur 38: formule [26]

|q'| stelt het aantal termen voor die in de geherformuleerde simpele query staan. Een belangrijk onderdeel uit de formule vind je terug op Figuur 39.

$$p(v_k|\mathcal{R}_i) = T_{i,k} \cdot I_k \cdot w_k$$

$$T_{i,k} = \frac{df_{i,k}}{df_{i,k} + 50 + 150 \cdot \frac{cw_i}{\overline{cw}}}$$

$$I_k = \frac{\log\left(\frac{|\mathscr{R}|+0.5}{cf_k}\right)}{\log\left(|\mathscr{R}| + 1.0\right)}$$

Figuur 39: formule [26]

Op Figuur 40 vind je terug wat de waarden in de formule voorstellen.

w_k is the weight of the term in the query;
$df_{i,k}$ is the number of records in the approximation of $\mathcal{R}_i$ containing value v_k;
cw_i is the number of values in the approximation of $\mathcal{R}_i$;
$\overline{cw}$ is the mean value of all the cw_i;
cf_k is the number of approximated resources containing value v_k;
$|\mathscr{R}|$ is the number of the resources.

Figuur 40:formule waarden [26]

$T_{i,k}$ uit de formule stelt het aantal keren voor dat een term uit de query voorkomt in een bepaalde resource. I_k zorgt ervoor dat een term die veel voorkomt in verschillende resource, en dus niet discriminerend genoeg is, niet te zwaar doorweegt. W_k stelt het gewicht voor dat aan een bepaalde term gehangen wordt.

Zoals eerder aangehaald is de Cori methode niet haalbaar binnen een pervasive omgeving. Verder lijkt deze methode op een keyword search zoals google of altavista gebruikt. Deze methode legt dus een nadruk op de syntactische overeenkomsten van termen en niet zozeer de semantische overeenkomsten. Stel dat je op zoek bent naar een document over leraren. In je query komt de term “leraar” voor. Deze methode zal dus geen Engelse resources selecteren omdat hier de term “leraar” als “tutor” voorkomt.

5.5. Conclusie

In het eerste deel van dit hoofdstuk is in grote lijnen uitgelegd welke weg een query aflegt van vraagstelling tot resultaat. In het scenario zien we dat een operator verschillende queries kan versturen. Deze queries kunnen naar gedistribueerde informatiebronnen verstuurd worden, waar ze opgelost moeten worden. Afhankelijk van het framework dat gebruikt wordt bij de gedistribueerde informatiebronnen, zullen de queries op een gelijkaardige manier, zoals hier uitgelegd, opgelost worden. We houden hier enkel rekening met het lokaal oplossen van queries. Indien we een query volledig gedistribueerd willen oplossen komt er heel wat meer bij kijken. In het volgende hoofdstuk bekijken we hoe een query op een gedistribueerde manier opgelost kan worden en welke problemen hierbij komen kijken.

In het tweede deel hebben we besproken hoe gedistribueerd queryen verloopt en welke problemen hierbij kunnen opduiken. Verder hebben we ook besproken welke optimalisaties er mogelijk zijn zodat er geen onnodige gedistribueerde resources gecontacteerd moeten worden. Deze optimalisaties zijn van groot belang, zeker bij pervasive omgevingen, omdat hier het netwerk gedeelte de bottleneck is.

6. Eigen implementatie

6.1. Omschrijving

Voor de implementatie van deze thesis heb ik gekozen voor een applicatie waarbij zoveel mogelijk gebruik wordt gemaakt van de besproken literatuur studie. Het is resultaat is een applicatie waarmee gebruikers “getracked” kunnen worden. Met tracken bedoelen we dat we kunnen zien waar een bepaalde persoon zich bevindt binnen een voorgedefinieerde omgeving. Het is ook mogelijk om berichten binnen de omgeving achter te laten voor andere gebruikers. Als laatste functie is er een soort agenda functie toegevoegd. Deze functies worden verder uitgelegd in dit hoofdstuk.

Met behulp van deze applicatie kan de gebruiker, net zoals in het scenario, een antwoord krijgen op zijn specifieke vragen. Mogelijke vragen die de gebruiker zich kan stellen:

- Waar zit een bepaalde gebruiker?
- Welke afspraken heeft een gebruiker?
- Bevindt de gebruiker zich binnen de omgeving?
- Is de gebruiker aanwezig op zijn afspraak?

6.2. Omgeving

In 7.1 hebben we gezien dat een gebruiker zich een aantal vragen kan stellen. Via de applicatie kan hij hierop een antwoord krijgen. Om een antwoord te krijgen op deze vragen moet de applicatie de gedistribueerde resources contacteren. Zo heeft iedere locatie binnen de omgeving een databank waarin de gebruikers, die zich op die locatie bevinden, geregistreerd worden. In deze databanken worden ook de berichten opgeslagen die achter gelaten worden door andere gebruikers. Iedere gebruiker heeft ook zijn eigen agenda databank waarin zijn afspraken opgeslagen worden. Deze databank kan op de pc, pda of laptop van de gebruiker staan. De gedistribueerde databases bestaan uit rdf databases of knowledge bases en zijn conform aan een bepaalde ontologie. Naast deze knowledge bases worden er nog een aantal databanken voorzien die de verschillende ontologieën voorstellen en een databank met meta-informatie betreffende de gebruikers en de locaties.

Gebruikers kunnen op een locatie geregistreerd worden met behulp van rfid technologie. Via een rfid tag kan een gebruiker zichzelf identificeren. De locaties worden van een rfid reader voorzien. Wanneer een gebruiker zijn rfid tag voor de reader houdt, wordt de gebruiker op deze locatie geregistreerd.

Om het gedistribueerd queryen mogelijk te maken wordt de omgeving voorzien van een mediator. Deze mediator kan de gebruiker aanspreken om zijn queries door te sturen. Deze mediator en de gedistribueerde databanken worden voorzien van een service die aangesproken kunnen worden via een uniek ip adres. Deze service zorgt ervoor dat de queries verstuurd kunnen worden.

Omdat we maar één rfid reader voor handen hadden, is er de mogelijkheid voorzien om deze rfid reader softwarematig te kunnen koppelen aan verschillende locaties. Via deze weg kunnen we een verplaatsing van een persoon toch simuleren.
Op onderstaande afbeelding zien we de rfid reader (links) samen met een rfid tag (rechts) (zie Figuur 41).

Figuur 41: Rfid reader en tag

Deze rfid tag wordt op een PDA gekoppeld en kan zo de tags lezen en versturen naar de service van de locatie.

6.3. Framework

In deze sectie wordt toegelicht uit welke delen het framework is opgebouwd en waarvoor ze dienen. In sectie 6.4 wordt er een voorbeeld query gegeven en wordt stap voor stap uitgelegd hoe deze query afgehandeld wordt. Het zal dan duidelijk worden hoe de verschillende delen in het framework met elkaar werken.

6.3.1. Mediator

De mediator wordt aangesproken door de gebruikers die een pervasive query willen versturen. Hij zal deze query aanpakken, verwerken en een antwoord terug sturen naar de gebruiker. Hoe deze verwerking precies gebeurt, zien we in sectie 6.4. Om een antwoord terug te kunnen sturen heeft de mediator informatie nodig uit andere resources.

De mediator heeft twee bestanden nodig om te kunnen functioneren, namelijk:

- ‘alignments.xml’ en
- ‘resourceInformation.xml’

Het bestand ‘alignments.xml’ stelt het mapping document voor dat vereist is voor de vertalingen, een beschrijving van dit document kan je terug vinden onder sectie 7.3.1.1. Het bestand ‘resourceInformation.xml’ bevat informatie betreffende de verschillende gedistribueerde resources, een beschrijving van dit document vind je terug in sectie 7.3.1.2.

6.3.1.1. Mapping document

Dit document wordt opgeslagen onder de naam 'alignments.xml' en wordt ingelezen door de mediator. Dit mapping document wordt gebruikt voor verschillende scenario's. Deze scenario's zijn beschreven in sectie 4.2.

Het ontologie mediatie probleem in deze implementatie lossen we in twee stappen op.
De eerste stap omvat een merge (zie 4.1.3) van de verschillende ontologieën waarop de gedistribueerde knowledge bases gebaseerd kunnen zijn. Deze merge is de unie van alle ontologieën.
In de tweede stap mappen (zie 4.1.1) we de verschillende ontologieën met de merge. Omdat we drie ontologieën hebben en dan de merged ontologie, krijgen we drie alignments:

- merged ↔ local
- merged ↔ general
- merged ↔ agenda

Het nadeel van deze aanpak is echter dat als er een nieuwe ontologie geïntroduceerd wordt, je de merged ontologie moet aanpassen. Je kunt dit oplossen door een aanpak te gebruiken gelijk de 'Observer' methode (zie 4.5.2). Dit heeft echter ook zijn nadelen. Met de Observer methode moet je connecteren met een bepaalde ontologie waardoor je beperkt bent tot de woordenschat van die ontologie. Via de merged aanpak heb je een overzicht van alle mogelijke woorden. Er is hier gekozen voor een merged aanpak omdat het nodig is een volledig overzicht te hebben.

Queries die door de gebruiker verstuurd worden naar de mediator moeten dus geschreven worden tegen de merged ontologie. Deze query wordt dan opgedeeld in verschillende source queries (zie 6.3.1) en vertaald naar de desbetreffende ontologie. Deze vertaling gebeurt dan met behulp van het mapping document.
Antwoorden die de mediator terug ontvangt, worden terug vertaald tegen de merged ontologie.

De alignment beschrijving die in deze thesis gebruikt wordt, maakt gebruikt van een eigen taal die door de mediator geïnterpreteerd kan worden. Onderaan zie je een voorbeeld van een mapping van de term 'Person' tussen de ontologieën local en merged.

```
<alignment>
        <sourceEntity>http://thesis/local.owl#Person</sourceEntity>
        <sourceOntology>localOntology</sourceOntology>
        <targetEntity>http://thesis/mergedOntology.owl#Person</targetEntity>
        <targetOntology>mergedOntology</targetOntology>
</alignment>
```

Figuur 42: Voorbeeld mapping

Als er een query bij de mediator binnenkomt met de term http://thesis/mergedOntology.owl#Person, en deze moet vertaald worden naar de local ontologie dan wordt deze term vervangen door http://thesis/local.owl#Person.

In sectie 6.3.1.2 zien we dat een index document gebruikt kan worden ter optimalisatie zodat geen onnodige knowledgebases moeten worden aangesproken. Het mapping document dat wij gebruiken wordt ook gebruikt als een soort index document. In sectie 7.4 zien we dat een query opgedeeld wordt in kleinere queries of source queries. Deze source queries worden naar

de remote resources verstuurd. Via het index document wordt gekeken of het wel zin heeft om deze source query te versturen naar de remote resource. Indien de remote resource geen antwoord kan geven op de source query heeft het dus geen zin. Dit wordt gecontroleerd door te kijken of het subject, predikaat en object van de source query vertaald kan worden naar termen die voortkomen in de ontologie van de remote resource.
Stel we hebben volgende triple pattern:

```
?x <http://thesis/mergedOntology.owl#hasAlias> ?alias
```

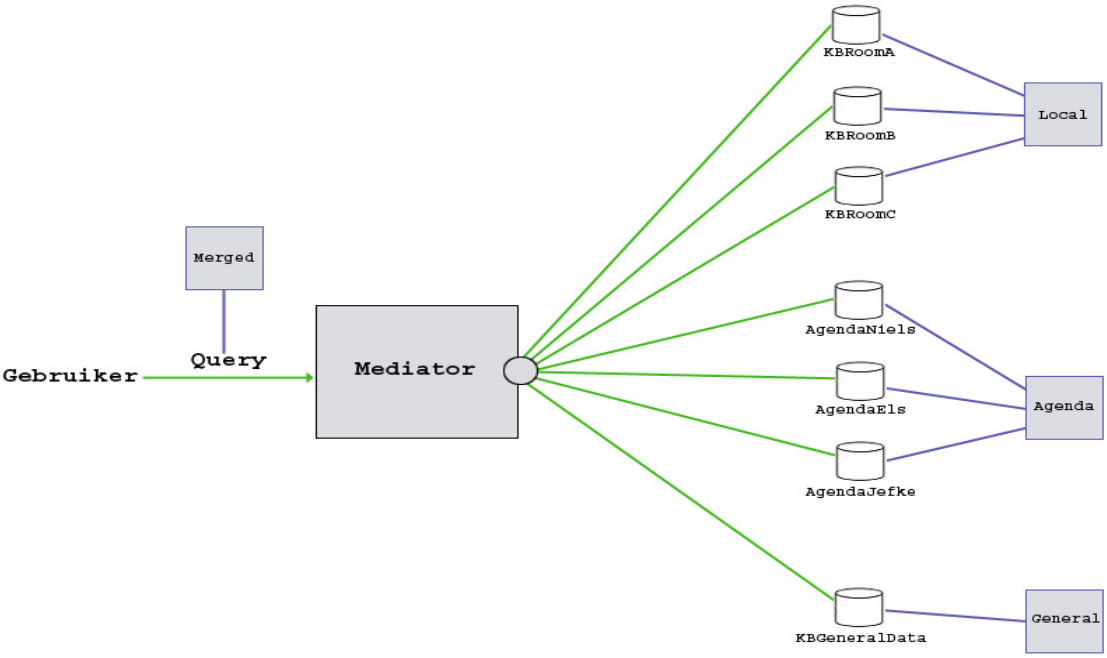
Figuur 43: Voorbeeld triple pattern

In deze triple pattern komt de term http://thesis/mergedOntology.owl#hasAlias voor. Door de mapping die te zien is op Figuur 44 kunnen we afleiden dat deze term te vertalen is naar de term http://thesis/generalData.owl#hasAlias afkomstig uit de ontologie genreralDataOntology.

```
<alignment>
<sourceEntity>http://thesis/generalData.owl#hasAlias</sourceEntity>
        <sourceOntology>generalDataOntology</sourceOntology>
        <targetEntity>http://thesis/mergedOntology.owl#hasAlias</targetEntit>
        <targetOntology>mergedOntology</targetOntology>
</alignment>
```

Figuur 44: Voorbeeld mapping

Indien zowel het subject, object en predikaat te vertalen zijn naar de ontologie generalDataOntology wil dit zeggen dat de remote resources die deze ontologie ondersteunen, deze query mogelijk kunnen beantwoorden. De source query kan dan naar deze remote resources verstuurd worden.



Figuur 45: Overzicht framework

Figuur 45 Geeft een overzicht van de gedistribueerde knowledge bases en de ontologieën waarop deze gebaseerd kunnen zijn.

6.3.1.2. Resource information document

In deze sectie wordt beschreven waaruit en waarvoor het resource information document dient. Dit document wordt opgeslagen onder de naam ‘resourceInformation.xml’ en wordt ingelezen door de mediator.

Dit document bevat informatie betreffende de verschillende gedistribueerde resources.
Op Figuur 46 wordt een voorbeeld gegeven.

```
<resource>
<id>KBRoomA</id>
        <ontologyId>localOntology</ontologyId>
        <ns>http://thesis/local.owl#</ns>
        <ip>localhost</ip>
        <port>5004</port>
</resource>
```

Figuur 46: Resource information voorbeeld

Dit voorbeeld geeft informatie weer voor resource KBRoomA. Je kunt met deze resource connecteren via het ip adres 127.0.0.1:5004. Naast deze informatie wordt er ook een id gekoppeld aan de resource en een ontologie id. Deze id’s worden door de mediator gebruikt om snel informatie terug te vinden en om te zien aan elke ontologie een resource gekoppeld is. In het mapping document worden deze id’s ook gebruikt zoals te zien is op Figuur 42 onder de tags sourceOntology en targetOntology.

6.3.2. Resource service

De resource service wordt gebruikt om de verschillende knowledge bases en remote ontologieën op een gedistribueerde manier aan te kunnen spreken. Deze service zorgt ervoor dat een query aangenomen kan worden, opgelost wordt en dat er een antwoord terug gestuurd wordt. Deze services worden dus normaal gezien enkel aangesproken door de mediator.

6.4. Query afhandeling

Deze sectie bespreekt de verschillende stappen die ondernomen worden om een antwoord te verkrijgen op een query. We gaan dit proces uitleggen aan de hand van een voorbeeld query (zie Figuur 47).

```
select ?alias ?rfidTag where {
?x <http://thesis/mergedOntology.owl#hasName> ?name . filter regex(?name,'niels', 'i')
?x <http://thesis/mergedOntology.owl#hasAlias> ?alias .
?x <http://thesis/mergedOntology.owl#hasRfidTag> ?rfidTag .
?x http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://thesis/mergedOntology.owl#Person    }
```

Figuur 47: Voorbeeld query

Deze query moet de alias en de rfid tag van ‘Niels Clauwers’ teruggeven. De meeste verwerking vindt plaats in de mediator en bespreken we in volgende sectie.

6.4.1. Mediator

De eerste stap die ondernomen moeten worden om een query op te lossen is deze te versturen naar de mediator. De mediator heeft een service draaien die luistert naar binnenkomende connecties. Wanneer een applicatie verbinding maakt met de mediator kan deze een 'NetworkObject' versturen. Het NetworkObject bevat informatie die nodig is om een de query op te kunnen lossen.

De mediator kan 2 types van NetworkObjecten ontvangen, namelijk 'QueryObject' of 'UpdateObject'. Wanneer men een 'QueryObject' verstuurt, wordt deze opgelost zoals in volgende sectie beschreven wordt. NetworkObjecten van het type 'UpdateObject' worden anders verwerkt, dit wordt beschreven in sectie 7.4.1.2.

6.4.1.1. Verwerking query

Bij het ontvangen van een query wordt er altijd vanuit gegaan dat deze geschreven is tegen de 'mergedOntologie' (zie Figuur 45). Wanneer dit niet het geval is, gaat er natuurlijk geen correct resultaat gevonden worden. Er wordt ook geen controle uitgevoerd of dit het geval is. Er zou bijvoorbeeld gecontroleerd kunnen worden of de juiste sleutelwoorden gebruikt worden in de query.

Stap 1:
De eerste verwerking die plaatsvindt, is het opsplitsen van de query in triple patterns (ref. sectie 3.2). Bij ons voorbeeld query krijgen we vier triple patterns, namelijk :

- ?x <http://thesis/mergedOntology.owl#hasName> ?name . filter regex(?name,'niels', 'i') (pattern 1)
- ?x <http://thesis/mergedOntology.owl#hasAlias> ?alias (pattern 2)
- ?x <http://thesis/mergedOntology.owl#hasRfidTag> ?rfidTag (pattern 3)
- ?x http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://thesis/mergedOntology.owl#Person (pattern 4)

Stap 2:
In de volgende stap wordt er per triple pattern gekeken naar welke ontologie deze pattern vertaald kan worden. Bij deze stap wordt de informatie uit het mapping bestand gebruikt (zie sectie 7.3.1.1). We krijgen dan per triple pattern een lijst van ontologieën naar waar de triple pattern vertaald kan worden. Bij ons voorbeeld query krijgen we dan volgend resultaat:

- pattern 1: localOntology, generalDataOntology
- pattern 2: generalDataOntology
- pattern 3: generalDataOntology
- pattern 4: localOntology, generalDataOntology

Per triple pattern wordt er gekeken of zowel het subject, predikaat als object vertaald kan worden naar de ontologie. Indien dit het geval is wordt de ontologie toegevoegd aan de lijst. Of een predikaat vertaald kan worden naar een bepaalde ontologie kan men in het mapping document zien. Als we het predikaat uit pattern 1 nemen:

```
<http://thesis/mergedOntology.owl#hasName>
```

Figuur 48: Predikaat triple pattern 1

Dan kunnen we uit het mapping document afleiden dat deze vertaald kan worden naar de ontologieën localOntology en generalDataOntology door volgende alignments uit het mapping document:

```
<alignment>
        <sourceEntity>http://thesis/local.owl#hasName</sourceEntity>
        <sourceOntology>localOntology</sourceOntology>
        <targetEntity>http://thesis/mergedOntology.owl#hasName</targetEntity>
        <targetOntology>mergedOntology</targetOntology>
</alignment>

<alignment>
        <sourceEntity>http://thesis/generalData.owl#hasName</sourceEntity>
        <sourceOntology>generalDataOntology</sourceOntology>
        <targetEntity>http://thesis/mergedOntology.owl#hasName</targetEntity>
        <targetOntology>mergedOntology</targetOntology>
</alignment>
```

Figuur 49: Alignments uit mapping document

Bij pattern 1 is er maar één entiteit die gecontroleerd moet worden en dat is het predikaat. Het subject en object zijn beiden variabelen en deze kunnen natuurlijk niet vertaald worden.

Stap 3:
In de derde stap wordt iedere triple pattern vertaald tegen iedere ontologie uit de lijst met compatibele ontologieën. We zetten ons voorbeeld verder en we krijgen dan volgende vertalingen voor pattern 1:

- localOntology:

```
CONSTRUCT { ?x <http://thesis/local.owl#hasName> ?name .
?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?type } where {
?x <http://thesis/local.owl#hasName> ?name  . filter regex(?name,'niels clauwers', 'i')
?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?type }
```

Figuur 50: Subquery compatibel aan localOntology

- generalDataOntology:

```
CONSTRUCT { ?x <http://thesis/generalData.owl#hasName> ?name .
?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?type } where {
?x <http://thesis/generalData.owl#hasName> ?name  . filter regex(?name,'niels clauwers', 'i')
?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?type }
```

Figuur 51:Subquery compatibel aan generalDataOntology

Via het sparql commando CONSTRUCT wordt er een nieuwe RDF graaf geconstrueerd. Dit wordt gedaan omdat er per vertaling of triple pattern ook het type van het subject moet opgevraagd worden. Dit type wordt later in het verwerkingsproces gebruikt, we komen hier later op terug.

Stap 4:
In stap 3 hebben we gezien dat pattern 1 vertaald kan worden naar twee ontologieën met als gevolg dat we twee queries hebben die verstuurd moeten worden. In stap 4 worden de vertaalde queries verstuurd naar alle resources die compatibel zijn aan de ontologie naar waar de query vertaald is. De resources die compatible zijn aan een bepaalde ontologie kunnen we

afleiden uit het resource information document. Voor localOntology vinden we volgende resources terug:

- KBRoomA
- KBRoomB
- KBRoomC

Voor generalDataOntology vinden we volgende resource terug:

- KBGeneralData

Deze data vinden we terug in het resource information document omdat iedere resource gekoppeld wordt aan een ontologyId zoals onderaan te zien is:

```
<resource>
        <id>KBRoomA</id>
        <ontologyId>localOntology</ontologyId>
        <ns>http://thesis/local.owl#</ns>
        <ip>localhost</ip>
        <port>5004</port>
</resource>
```

Figuur 52: Resource information voorbeeld

We kunnen hieruit afleiden dat resource 'KBRoomA' compatibel is aan ontologie 'localOntology'.
De nodige informatie om te connecteren met een remote resource kan eveneens terug gevonden worden in het resource information document.
De mediator ontvangt dan een antwoord dat geëncapsuleerd is in een NetworkObject van het type 'ResultObject'. De antwoorden die de mediator ontvangt worden gemerged in een algemeen rdf model. In dit rdf model zitten dus alle antwoorden op de triple patterns. Voor onze query ziet dit model er als volgt uit:

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:j.0="http://thesis/generalData.owl#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >

<rdf:Description rdf:about="http://thesis/generalData.owl#Person2">
   <j.0:hasRfidTag>ABC124</j.0:hasRfidTag>
   <j.0:hasAlias>P1</j.0:hasAlias>
   <rdf:type rdf:resource="http://thesis/generalData.owl#Person"/>
</rdf:Description>

<rdf:Description rdf:about="http://thesis/generalData.owl#Person1">
   <j.0:hasRfidTag>E004010008B1B4B40000</j.0:hasRfidTag>
   <j.0:hasName>Niels Clauwers</j.0:hasName>
   <rdf:type rdf:resource="http://thesis/generalData.owl#Person"/>
   <j.0:hasAlias>Skyke</j.0:hasAlias>
</rdf:Description>
```

Figuur 53: Algemeen rdf model

De meest relevante delen zijn uit dit model gehaald en worden getoond op Figuur 53. Zoals je kunt zien is deze data nog niet compatibel met de mergedOntology.

Stap 5:
In de 5de stap worden de instanties uit het algemene rdf model (zie Figuur 53) vertaald tegen de mergedOntology. Omdat alles gedistribueerd werkt, kunnen ook de ontologieën gedistribueerd benaderd worden. Het eerste wat gedaan wordt bij deze vertaling is het ophalen van de mergedOntology. Hiervoor wordt volgende query verstuurd naar de mergedOntology:

```
CONSTRUCT{ ?x ?y ?z } where { ?x ?y ?z }
```

Figuur 54: Query voor ontologie op te halen

Dan wordt er voor ieder subject een nieuwe vertaalde instantie gemaakt. Voor ons voorbeeld hebben we volgende subjects die vertaald moeten worden:

- http://thesis/generalData.owl#Person1
- http://thesis/generalData.owl#Person2

Hier zijn enkel de meest relevante subjects getoond. In stap 3 hebben we gezien dat voor iedere triple pattern er ook het type aanvast gehangen werd door het construct commando. Dit type wordt nu gebruikt om te bepalen naar welk element de instantie vertaald moet worden zodat deze compatibel is aan de mergedOntology ontologie. Op Figuur 53 zien we dat Person2 van het type http://thesis/generalData.owl#Person is. Vervolgens wordt er gekeken wat het overeenkomende type in de mergedOntology ontologie is. We krijgen dan als resultaat http://thesis/mergedOntology.owl#Person. Als de nieuwe instantie aangemaakt is worden de properties ook vertaald en aan de instantie gekoppeld.
Deze vertaling gebeurt voor ieder subject en we krijgen als resultaat het volgende (zie Figuur 55):

```
<rdf:Description rdf:about="http://thesis/mergedOntology.owl#Person2">
   <j.0:hasAlias>P1</j.0:hasAlias>
   <j.0:hasRfidTag>ABC124</j.0:hasRfidTag>
   <rdf:type rdf:resource="http://thesis/mergedOntology.owl#Person"/>
</rdf:Description>
<rdf:Description rdf:about="http://thesis/mergedOntology.owl#Person1">
   <j.0:hasAlias>Skyke</j.0:hasAlias>
   <j.0:hasName>Niels Clauwers</j.0:hasName>
   <j.0:hasRfidTag>E004010008B1B4B40000</j.0:hasRfidTag>
   <rdf:type rdf:resource="http://thesis/mergedOntology.owl#Person"/>
</rdf:Description>
```

Figuur 55: Resultaat vertaling

Stap 6:
In de 6de en laatste stap wordt de initiële query losgelaten op het vertaalde rdf model. Het verkregen antwoord wordt dan verstuurd naar de vragende partij in een NetworkObject van het type ‘ResultsObject’. Het resultaat ziet er dan als volgt uit:

```
RESULT
------------
?alias: Skyke
?rfidTag: E004010008B1B4B40000
```

Figuur 56: Resultaat

6.4.1.2. Verwerking update query

Een update query verschilt van een normale query doordat een update query niet opgedeeld kan worden. Een update query moet verstuurd worden naar een specifieke resource. Indien er bijvoorbeeld een bericht moet verwijderd worden van een resource dan moet er een delete query verstuurd worden naar deze specifieke resource. Update queries worden ook eerst naar de mediator verstuurd die geëncapsuleerd zijn in een NetworkObject van het type 'UpdateObject'. Dit object bevat dan de nodige informatie zodat de mediator weet naar welke resource de query verstuurd moet worden.

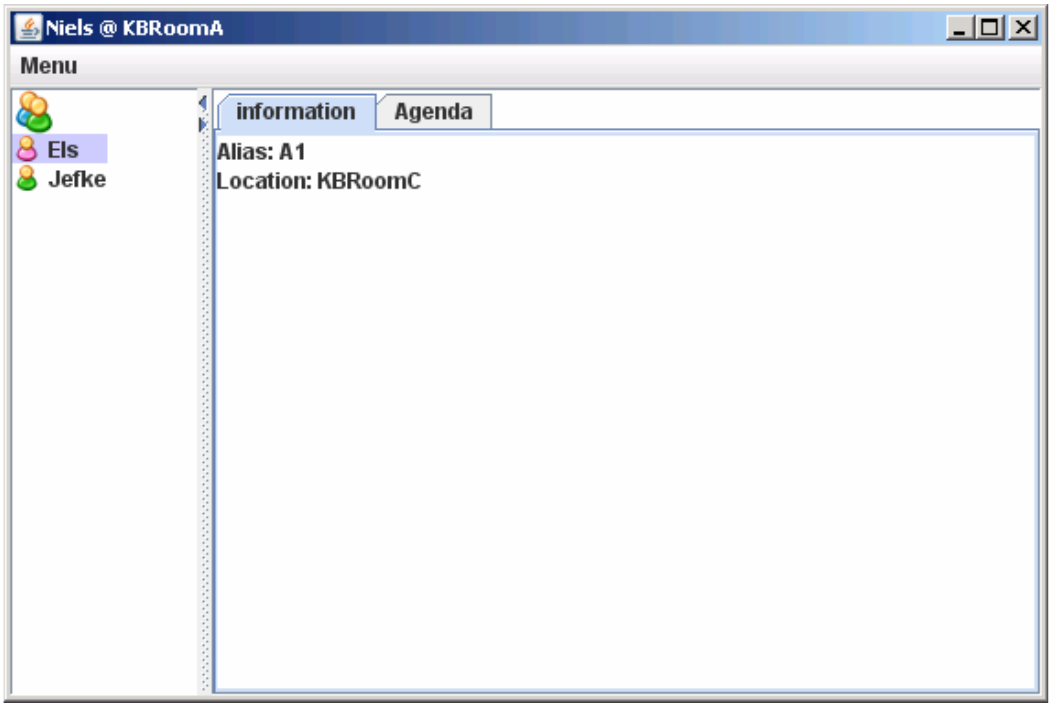
6.5. Interface

Het applicatieve gedeelte bestaat uit een interface voor de client en een aantal interfaces voor de mediator, de locaties en voor de rfid reader. In volgende secties worden deze interfaces kort toegelicht.

6.5.1. Client

De client interface wordt gebruikt om informatie betreffende bepaalde personen, die een rfid tag dragen, op te vragen en om berichten voor deze personen achter te laten.

Op Figuur 57 zien we de interface van de applicatie waarbij gebruiker 'Niels' is aangemeld.



Figuur 57: Userinterface client

In de titelbalk kan je zien met welke gebruiker je bent aangemeld en in welke locatie je geregistreerd staat. Aan de linker kant heb je een lijst met gebruikers waarvan je de locatie kan opvragen en andere informatie zoals de gebruikte alias of agenda. Deze informatie kan gemakkelijk uitgebreid worden door de overeenkomstige ontologieën aan te passen. In sectie

7.6 wordt een overzicht gegeven van de ontologieën. Het zal dan duidelijk worden hoe deze data uitgebreid kan worden.

Via het menu (zie Figuur 58) kan je van gebruiker veranderen, een bericht achterlaten op een locatie voor een bepaalde gebruiker of opnieuw connecteren met de mediator.

Figuur 58: Menu

De opties 'Switch User' en 'Leave Message' worden verduidelijk in volgende sectie. De optie 'Connect' wordt gebruikt om te connecteren met de mediator. De applicatie probeert standaard verbinding te maken met ip adres 127.0.0.1, maar indien de mediator via een ander ip adres te bereiken is, kan je deze optie gebruiken (zie Figuur 59).

Figuur 59: Set ip optie

6.5.1.1. Berichten

Deze sectie geeft een kort scenario over hoe je een bericht kan achter laten voor een andere gebruiker.

De eerste stap is het aanmelden met de juiste gebruikersnaam. Dit gebeurt via het menu 'Switch User' (zie Figuur 60)

Figuur 60: Switch User optie

Als de gebruiker een bericht wil achterlaten op een bepaalde locatie moet hij zichzelf eerst registreren op die locatie. Dit kan de hij doen door naar de locatie te lopen en zijn rfid tag te laten lezen door de rfid reader van deze locatie. Je kunt zien dat je geregistreerd bent op een locatie doordat deze locatie vermeld staat in de titelbalk van de applicatie (zie Figuur 61).

Figuur 61: Titelbalk met informatie

Nu kan de gebruiker een bericht achter laten via het menu 'Leave message'.

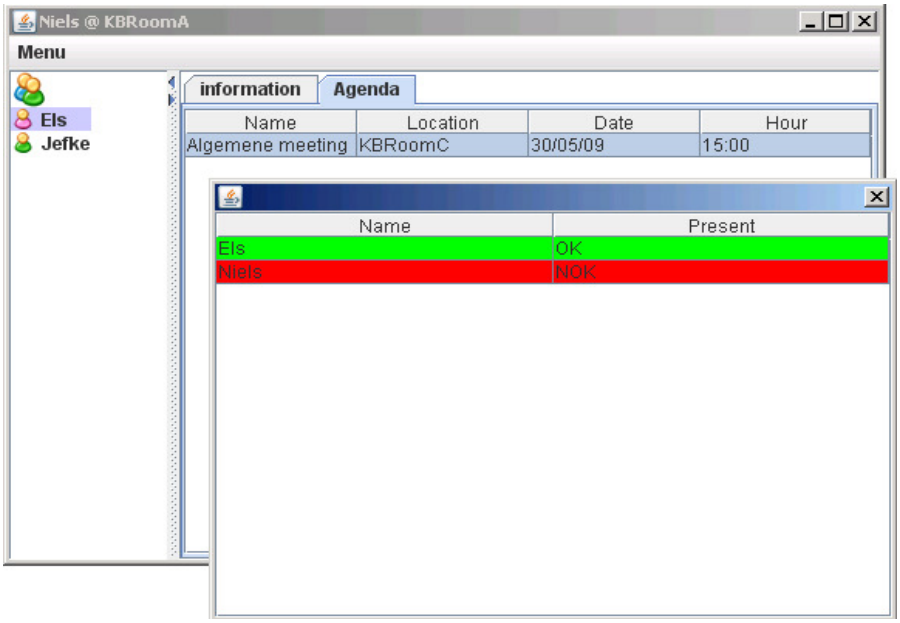
Figuur 62: Leave message interface

In het dropdown-menu selecteer je de juiste gebruiker, je typt het bericht in en je verzendt het bericht. Als je nu met gebruiker 'Jefke' aanmeldt en je jezelf registreert op locatie 'KBRoomA' zal er een bericht verschijnen dat achtergelaten is door gebruiker 'Niels' (zie Figuur 63).

Figuur 63: Bericht ontvangen

6.5.1.2. Agenda

Deze sectie geeft kort weer hoe je de agenda functie kan gebruiken. Stel dat je wilt weten of gebruiker 'Els' afspraken heeft. Dan kan je aan de linkerkant klikken op de naam en het tabblad 'Agenda' openen. Door met de rechter muisknop te klikken op een agendapunt kun je controleren of er nog andere mensen op deze zelfde afspraak aanwezig moeten zijn en of de mensen effectief aanwezig zijn op de locatie.



Figuur 64: Agenda functie

Op Figuur 64 kunnen we zien dat gebruiker 'Niels' ook aanwezig moet zijn op deze afspraak maar nog niet aanwezig is. Deze functie kan bijvoorbeeld nuttig zijn om snel een overzicht te krijgen of alle mensen die aanwezig zouden moeten zijn ook effectief aanwezig zijn. Indien er bepaalde mensen niet aanwezig zouden zijn, kan er gecontroleerd worden waar ze zich bevinden.

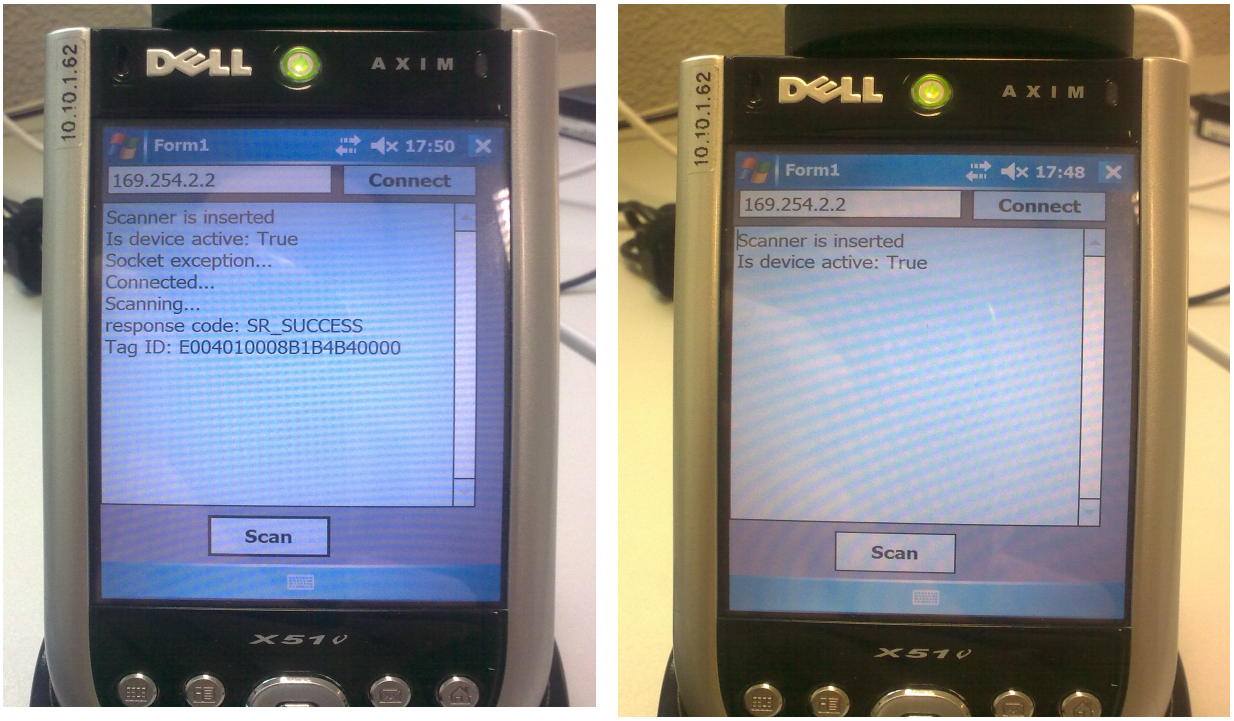
6.5.2. Locaties

Alle locaties worden van een simpele text based interface voorzien (zie Figuur 65) waar we een log kunnen opvolgen van de queries die aankomen, waar er via bepaalde commando's settings kunnen aangepast worden en gebruikers geregistreerd kunnen worden. Dit is voorzien zodat het testen van de applicatie vlotter verloopt. De mediator heeft ook een dergelijk log venster.

Figuur 65: Interface locaties

6.5.3. Rfid reader

Op onderstaande afbeeldingen (Figuur 66) zie je enkele schermafdrukken van de RFID-reader interface.



Figuur 66: Rfid reader interface

Via deze interface kun je connecteren met een service die voorzien is om rfid tags te ontvangen. Wanneer deze service een rfid tag ontvangt, wordt de user, die gekoppeld is aan deze tag, geregistreerd binnen de locatie waar deze service aan verbonden is. Softwarematig kan deze service verbonden worden aan verschillende locaties.

Door de rfid tag dicht bij de reader te houden en op scan te drukken, wordt de tag gelezen. Indien er dan een verbinding bestaat naar de rfid service wordt de tag ook verzonden.

6.6. Ontologieën

In deze sectie worden de ontologieën, die binnen deze applicatie gebruikt worden, toegelicht. Zo zal duidelijk worden hoe data gemakkelijk uitgebreid kan worden en hoe deze ontologieën binnen de applicatie gebruikt worden.

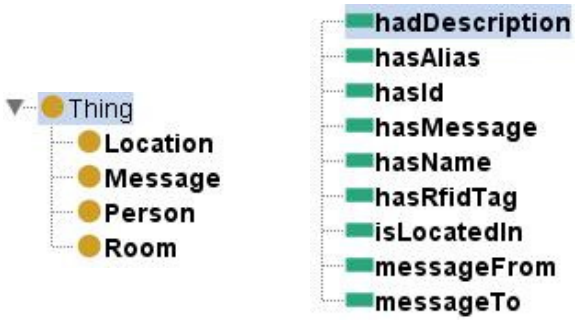
6.6.1. Merged ontologie

De merged ontologie is het resultaat van de ontologie merging stap. Deze ontologie wordt dan ook gebruikt in de verschillende ontologie mediatie scenario's zoals het herschrijven van queries of het vertalen van instanties (zie sectie 4.2). Deze merged ontologie is gebaseerd op een aantal input ontologieën die we later nog bespreken. Deze input ontologieën zijn:

- local ontologie
- general ontologie
- agenda ontologie
-

Een query die bij de mediator aankomt, is geschreven tegen de merged ontologie. Dit zorgt ervoor dat alle termen die in de input ontologieën voorkomen ook moeten voorkomen in de merged ontologie. Wat als gevolg heeft dat als er een term wordt toegevoegd in één van de input ontologieën, de merged ontologie ook aangepast moet worden.

Op onderstaande afbeelding zie je de klassen en de data properties van deze ontologie.



Figuur 67: Overzicht merged ontologie

6.6.2. Local ontologie

De local ontologie wordt gebruikt voor de rdf databases op de verschillende locaties. In deze databases worden de gebruikers geregistreerd die aanwezig zijn op de desbetreffende locatie. Queries die afgehandeld moeten worden op deze databases moeten geschreven zijn tegen de local ontologie. Op onderstaande afbeelding zie je de klassen en data properties van deze ontologie.

Figuur 68: Overzicht local ontologie

6.6.3. Agenda ontologie

De agenda punten van de gebruikers worden opgeslagen in hun agenda database. Deze database moet conform zijn aan de agenda ontologie. Op Figuur 69 zie je een overzicht van de klassen en properties van deze ontologie.

Figuur 69: Overzicht agenda ontologie

6.6.4. General ontologie

Er is één database voorzien waarin algemene informatie wordt opgeslagen zoals de naam van een persoon en de alias. Deze database is conform aan de general ontologie. Queries die geschreven worden voor deze database moeten dus conform zijn tegen deze ontologie.
Op onderstaande afbeelding zie je de klassen en properties van deze ontologie.

Figuur 70: Overzicht general ontologie

6.7. Mogelijke uitbreidingen

In dit hoofdstuk hebben we de mediator besproken. Wat we zien is dat de mediator twee bestanden moet inlezen (zie 7.3.1.1 en 7.3.1.2) voordat hij queries kan beantwoorden. Bij een aanpassing aan de pervasive omgeving moeten deze bestanden ook aangepast worden en opnieuw ingeladen worden door de mediator. Dit zorgt ervoor dat de mediator niet volledig autonoom kan werken. Met behulp van optimalisatie 7.7.2 en 7.7.3 kan de mediator wel autonoom (tot op een bepaald niveau) werken.

6.7.1. Optimalisatie

In sectie 6.3.1.3 hebben we gesproken over een optimalisatie waarbij we door toevoeging van inter-store overlap informatie, kunnen vermijden dat source queries verstuurd worden naar onnodige knowledge bases. De implementatie die we nu hebben gaat controleren naar welke knowledge bases een source query verstuurd kan worden. Indien er drie knowledge bases bestaan waar naar de source query verstuur kan worden, dan wordt deze source query ook

naar deze drie databases verstuurd. Er wordt geen rekening gehouden of er duplicaat informatie in deze drie databanken bevindt. Waardoor er onnodige queries verstuurd worden. Dit is een cruciale optimalisatie bij een grote hoeveelheid aan gedistribueerde databases. Er is echter ook een negatieve kant aan deze optimalisatie. Deze inter-store overlap informatie moet beschikbaar zijn voor de mediator wat wil zeggen dat deze informatie altijd onderhouden moet worden als ontologieën aangepast worden of wanneer er knowledge bases bijkomen.

6.7.2. Automatisatie ontologie alignment

De implementatie maakt gebruik van een mapping document waarin de semantische overeenkomsten tussen de verschillende ontologieën opgenomen wordt. Dit document wordt gebruikt voor de verschillende mediatie scenario's. Voor onze implementatie volstaat het om dit document handmatig op te stellen, maar in een grote pervasive omgeving waar er continue ontologieën kunnen bijkomen of wegvallen zou het handig zijn als er op dit vlak een automatisatie zou toegevoegd worden. Hoewel dit handig zou zijn, moet er rekening mee gehouden worden dat er altijd een domein expert aan te pas moet komen om het mapping document volledig te maken. Een dergelijke automatisering kan nooit volledig zijn.

6.7.3. P2P

In sectie 7.3.1.2 hebben we het nut van het resource information document besproken. Dit is een gecentraliseerd document waardoor bij toevoeging of verwijdering van gedistribueerde resources dit document aangepast moet worden. Bij een dynamische pervasive omgeving zou dit document veel aangepast moeten worden. Het zou handig zijn een manier te voorzien waardoor gedistribueerde resources automatisch ontdekt kunnen worden door de mediator. Via Peer 2 Peer kunnen we dit mogelijk maken. We zouden hiervoor bijvoorbeeld het JXTA framework kunnen gebruiken [30].

6.8. Conclusie

In dit hoofdstuk hebben we de eigen implementatie van deze thesis besproken. Er wordt een client server architectuur gebruikt waarbij gebruik wordt gemaakt van Query shipping (zie 6.2). De mediator gaat de gedistribueerde resources contacteren om een antwoord te vormen op de query van de gebruiker. Deze gedistribueerde resources zijn heterogeen, Met andere woorden ze kunnen gebaseerd zijn op verschillende ontologieën. Een mogelijke oplossing zou kunnen zijn om wrappers (Zie 6.3) te introduceren die dit kan opvangen. Voor iedere ontologie waarop een database kan gebaseerd zijn, zou er dan een wrapper voor voorzien moeten worden. Het nadeel hierbij is dat als er een nieuwe ontologie geïntroduceerd wordt er een nieuwe wrapper geschreven moet worden wat niet echt een voordeel is bij een dynamische pervasive omgeving. We hebben hier dus gekozen voor ontologie mediatie technieken (zie sectie 4) zodat de uitbreiding van de pervasive omgeving enkel resulteert in de aanpassing van het mapping document (zie 7.3.1.1) en het resource information document (zie 7.3.1.2). We moeten hier wel rekening houden met het feit dat de omgeving in deze implementatie overal gebruik maakt van het framework JENA [31] en verschillende technieken die gebruikt worden in het semantische web (zie sectie 3). Stel dat er een gedistribueerde resource geïntroduceerd wordt die met een relationele database werkt, die dus een heel andere aanpak vereist, dan zouden er wel wrappers geïntroduceerd moeten worden omdat de syntax van deze query taal anders is.

7. Conclusies

Bij het bestuderen van de literatuur zijn we tot een aantal conclusies gekomen.

We zijn begonnen met het beschrijven van een scenario waarin duidelijk gemaakt wordt, waaruit een pervasive omgeving kan bestaan en waarom er nood is aan pervasive querying. Vervolgens is kort besproken waaruit een pervasive omgeving bestaat en hebben we het semantische web toegelicht. Hieruit kunnen we besluiten dat de technieken die in het semantische web worden gebruikt, ook gebruikt kunnen worden bij pervasive queryen. Het semantische web is een web van dingen die gerelateerd worden met elkaar. Een pervasive omgeving kan op een gelijkaardige manier geïnterpreteerd worden.

Bij het queryen van een pervasive omgeving is het mogelijk dat er verschillende databases ondervraagd moeten worden, die niet altijd homogeen zijn. Toch moet er een manier bestaan om deze transparant te kunnen aanspreken. Deze transparantie kan voorzien worden met behulp van wrappers. Wrappers zorgen ervoor dat alle databases uniform aangesproken kunnen worden. Voor iedere heterogene database moet er een dergelijke wrapper worden opgesteld. Hierdoor moeten er bij uitbreidingen nieuwe wrappers gemaakt worden voor de nieuwe databases. Ontologie mediatie biedt hiervoor een betere oplossing. Bij ontologie mediatie gaan we iedere heterogene database baseren op een ontologie. Voor de communicatie tussen deze ontologieën worden mediatietechnieken gebruikt. Kort samengevat leggen deze technieken de overeenkomsten vast tussen de ontologieën in een document. Dit document kan dan voor verschillende scenario's gebruikt worden zoals het herschrijven van een query, bestemd voor database A, naar een query die bestemd is voor database B.

Bij het gedistribueerd queryen hebben we gezien dat het netwerk kan zorgen voor de grootste bottleneck. Daardoor moet het netwerk zo efficiënt mogelijk worden gebruikt. Het is dus niet de bedoeling bepaalde queries onnodig te versturen. Om dit te voorkomen moeten enkel de nodige databases geselecteerd en gecontacteerd worden. Dit probleem kan worden aangepakt door resource selection technieken. Deze technieken zorgen ervoor dat enkel de belangrijkste databases geselecteerd worden door de queries te analyseren. Ze controleren welke delen van een query beantwoord kunnen worden door een database.

We kunnen besluiten dat door samenwerking van ontologie mediatie en de verschillende resource selection technieken, we een transparante methode kunnen aanbieden die het queryen van gedistribueerde, heterogene databases een stuk gemakkelijker en efficiënter maken.

Bibliografie

[1] Nigel Shadbolt, Wendy Hall, Tim Berners-Lee | The Semantic Web *Revisited* | http://eprints.ecs.soton.ac.uk/12614/1/Semantic_Web_Revisted.pdf

[2] An Introduction to the semantic web | http://www.youtube.com/watch?v=OGg8A2zfWKg

[3] Sean B. Palmer | *The Semantic Web: An Introduction* | http://infomesh.net/2001/swintro/

[4] Aaron Swartz | *The Semantic Web In Breadth* | http://logicerror.com/semanticWeb-long

[5] Sandro Hawke | *How We Identify Things (on the Semantic Web)?* | http://www.w3.org/2001/03/identification-problem/

[6] Frank Manola, Eric Miller | *RDF-primer* | W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-primer/

[7] Aimilia Magkanaraki, Grigoris Karvounarakis, Ta Tuan Anh, Vassilis Christophides, Dimitris Plexousakis | *Ontology Storage and Querying* | Foundation for Research and Technology Hellas Institute of Computer Science Information Systems Laboratory. http://139.91.183.30:9090/RDF/publications/tr308.pdf

[8] Eric Prud'hommeaux, Andy Seaborne | SPARQL Query Language for RDF | W3C Recommendation 15 January 2008. http://www.w3.org/TR/rdf-sparql-query/

[9] World Wide Web Consortium | http://www.w3.org/

[10] Natalya F. Noy, Deborah L. McGuinness | *Ontology Development 101: A Guide to Creating Your First Ontology* | http://protege.stanford.edu/publications/ontology_development/ontology101.pdf

[11] Jérôme Euzenat, François Scharffe, Antoine Zimmermann | *Expressive alignment language and implementation* | http://knowledgeweb.semanticweb.org/semanticportal/deliverables/D2.2.10.pdf

[12] Giorgos Stoilos, Giorgos Stamou, and Stefanos Kollias | *A String Metric for Ontology Alignment* | http://www.image.ece.ntua.gr/php/savepaper.php?id=378

[13] Jérôme Euzenat | *An API for ontology alignment* | http://gforge.inria.fr/docman/view.php/117/251/align.pdf

[14] Jos de Bruijn, Marc Ehrig, Cristina Feier, Francisco Martin-Recuerda, Fran¸cois cois Scharffe, Moritz Weiten | *Ontology mediation, merging and aligning* | http://www.dit.unitn.it/~p2p/RelatedWork/Matching/mediation-chapter.pdf

[15] Jérôme Euzenat, Petko Valtchev | *Similarity-based ontology alignment in OWL-Lite* | http://www.dit.unitn.it/~accord/RelatedWork/Matching/align-ECAI04-FSub.pdf

[16] Franccois Scharffe, Jérôme Euzenat, Dieter Fensel | *Towards Design Patterns for Ontology Alignment* | ftp://ftp.inrialpes.fr/pub/exmo/publications/scharffe2008a.pdf

[17] Donald Kossmann | *The State of the Art in Distributed Query Processing*

[18] Olaf Hartig and Ralf Heese | The *SPARQL Query Graph Model for Query Optimization* | http://www.eswc2007.org/pdf/eswc07-hartig.pdf

[19] W.J.A. Verheijen | *Efficient Query Processing in Distributed RDF Databases* | http://wwwis.win.tue.nl/~ksluijs/material/Verheijen-Master-Thesis-2008.pdf

[20] Vassilis Papadimos, David Maier | Distributed *Queries without Distributed State* | http://db.ucsd.edu/webdb2002/papers/48.pdf

[21] James C. French, Allison L. Powell, Walter R. Creighton | *Efficient Searching in Distributed Digital Libraries* | http://www.cs.virginia.edu/papers/p283-french.pdf

[22] Yongluan Zhou | *Adaptive Distributed Query Processing* | http://www.imada.sdu.dk/~zhou/papers/phdposter.pdf

[23] E. Mena, V. Kashyap, A. Sheth, A. Illarramendi | *OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies*

[24] Adrian K. Clear, Stephen Knox, Juan Ye, Lorcan Coyle, Simon Dobson, Paddy Nixon | *Integrating Multiple Contexts and Ontologies in a Pervasive Computing Framework* | http://www.csi.ucd.ie/UserFiles/publications/1148314115874.pdf

[25] AnHai Doan, Jayant Madhavan, Robin Dhamankar, Pedro Domingos, Alon Halevy | *Learning to Match Ontologies on the Semantic Web* | http://pages.cs.wisc.edu/~anhai/papers/glue-vldbj.pdf

[26] M. Elena Renda, Umberto Straccia | *On the Effectiveness of Automatic Schema Matching Over Heterogeneous Digital Libraries* | http://dienst.isti.cnr.it/Dienst/Repository/2.0/Body/ercim.cnr.isti/2005-TR-12/pdf?tiposearch=cnr&langver=

[27] *Pervasive (ubiquitous) computing today* | http://www.stylusinc.com/Common/whitepapers/WhitePapers/Pervasive%20Computing%20Today.pdf

[28] *Pervasive computing in general* | http://www.imada.sdu.dk/~joan/security/reports06/poder.pdf

[29] Parliamentary Office of Science and Technology | *Pervasive computing* | http://www.parliament.uk/documents/upload/postpn263.pdf

[30] JXTA | https://jxta.dev.java.net

[31] JENA | http://jena.sourceforge.net