A multi-device presence agent

Bram Bonné

Eindwerk voorgedragen tot het behalen van de graad van
bachelor in de informatica/ICT/kennistechnologie

Promotor: Prof. dr. Karin Coninx
Co-promotor: Prof. dr. Kris Luyten
Begeleider: Jo Vermeulen

Universiteit Hasselt / transnationale Universiteit Limburg
Academiejaar 2008-2009

Abstract

In dit bachelor eindwerk wordt een presence network geïmplementeerd, waarbij verschillende apparaten aan elkaar kunnen laten weten hoe actief ze op dat moment gebruikt worden. Ook worden hierop enkele diensten gebouwd. Zo is er een everywhere messaging service, waarmee verschillende gebruikers op het netwerk met elkaar kunnen blijven communiceren, ook wanneer ze van apparaat wisselen. Hierbij wordt dan ook een SMS service voorzien, die berichten kan doorsturen naar de mobiele telefoon van de gebruiker indien hij niet actief is op het presence network. Ook wordt automatisch de huidige locatie van de correspondent aangegeven.

Vervolgens wordt een vergelijking gemaakt met het XMPP protocol, dat ook een presence mechanisme aanbiedt bij het gebruik van instant messaging. Hierbij wordt voor een (basis) implementatie van een presence network dat gebruik maakt van XMPP bots gezorgd.

Tenslotte werden er gebruikerstesten uitgevoerd waarbij de presence client gecombineerd werd met een presence agent (die effectief de aanwezigheid van de gebruiker op één apparaat zal meten) uit een andere bachelorproef [Bamps 2009]. De resultaten hiervan worden tot slot ook kort besproken in dit eindwerk.

Woord vooraf

Bij het maken van dit bachelor eindwerk heb ik kunnen rekenen op veel hulp van Jo Vermeulen. Ik wil hem dan ook op de eerste plaats bedanken voor het helpen sturen en structureren van dit eindwerk, het aanbrengen van ideeën en het helpen verzamelen van informatie. Ook mijn co-promotor dr. Kris Luyten wil ik bedanken voor het geven van suggesties over de opbouw van het eindwerk. Verder bedank ik ook dr. Karin Coninx, promotor voor dit bachelor eindwerk.

Ook de deelnemers aan de gebruikerstests zijn erg bedankt voor het vrijmaken van wat tijd om de applicatie te testen. Verder bedank ik ook de rest van de studenten Informatica/ICT voor de fijne werkomgeving en de toffe sfeer in de klas.

Tenslotte bedank ik ook mijn ouders en mijn broer, voor de grote steun bij mijn studies.

Inhoudsopgave

Hoofdstuk 1

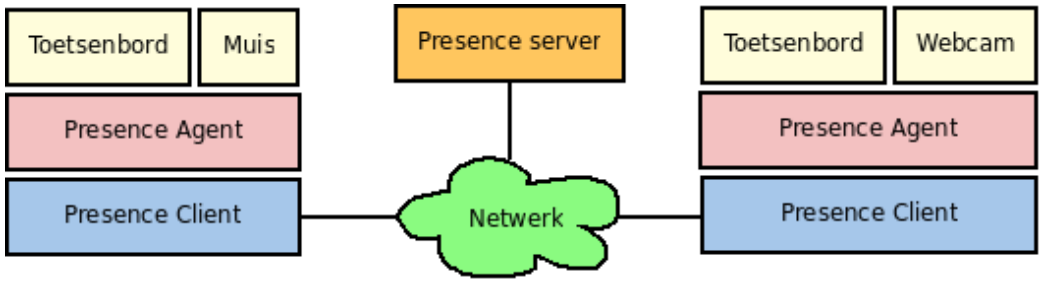
Inleiding

In dit eindwerk onderzoeken we op welke manier het client-gedeelte van een multi-device presence agent kan geïmplementeerd worden. Om te begrijpen wat een presence client is, moeten we eerst even uitleggen wat een presence *agent* is. Een presence *agent* is een stuk software dat de activiteit van de gebruiker meet op een apparaat door deze te schatten aan de hand van omgevings- en gebruiksfactoren. Het ontwikkelen van zulk een agent voor een bepaald apparaat vormde het onderwerp van een andere bachelorproef [Bamps 2009].

Het doel van dit eindwerk is dan om ervoor te zorgen dat verschillende apparaten die zo'n presence agent gebruiken voor het schatten van de aanwezigheid zich in een presence *network* bevinden. In dit netwerk wordt de aanwezigheid van de gebruiker op zijn verschillende apparaten gebruikt om te beslissen op welk van deze apparaten de gebruiker het meest actief is. Hiervoor implementeren we een presence *server*, die ervoor zal zorgen dat de verschillende apparaten op de hoogte zijn van elkaars veranderende aanwezigheid, evenals een presence *client* voor enkele platformen (meerbepaald GNU/Linux en Google Android [Google 2009]) die de laag vormt tussen de presence agent en de presence server en die de activiteit van het apparaat waarop hij geïnstalleerd is binnen het netwerk bijhoudt. Een schema hiervan is weergegeven in figuur 1.1 op de volgende pagina.

Voor het Android platform werd er, buiten de presence client, ook nog een presence agent ontwikkeld. Het was immers ook de bedoeling om te onderzoeken hoe een presence client moet omgaan met mobiele apparaten. Aangezien er voor dit platform echter nog geen implementatie van een presence agent bestond, moest ook deze ontwikkeld worden.

Verder werden er ook enkele uitbreidingen op het presence network geïmple-



Figuur 1.1: Een voorbeeld van een zeer eenvoudig presence network waarbij twee apparaten zijn aangesloten op een server.

menteerd, waarop we in het volgende hoofdstuk dieper zullen ingaan.

Hoofdstuk 2

Vereisten en doelstellingen

2.1 Algemeen

Het algemene doel is om een framework te maken dat ervoor zorgt dat verschillende apparaten van een bepaalde persoon op de hoogte zijn van elkaars beschikbaarheid. Met het woord 'beschikbaarheid' bedoelen we hier: de mate waarin het apparaat op dat moment door een persoon gebruikt wordt. Ook moet het geheel uitbreidbaar zijn, zodat er verschillende diensten bovenop gebouwd kunnen worden. Deze diensten omvatten:

- Een globaal notificatie-framework, waarmee verschillende apparaten (niet noodzakelijk desktop computers) notificaties kunnen weergeven op het apparaat van de gebruiker met de hoogste beschikbaarheid. Voorbeelden van notificaties die nut hebben op een andere machine dan degene waar de notificatie vandaan komt zijn:

  Voltooide taken: bestand gedownload, code gecompileerd

  Dringende berichten: een kritieke fout op een webserver kan onmiddellijk aan de beheerder gemeld worden.

  Berichten van andere soorten IP-aware apparaten: een wasmachine kan bijvoorbeeld laten weten wanneer de was klaar is [Williams 2000].

- Een everywhere messaging-framework, waarmee gebruikers korte (dringende) berichtjes naar elkaar kunnen sturen die naar het apparaat met de hoogste beschikbaarheid worden gestuurd. Deze dienst is vooral bedoeld voor gebruik binnen eenzelfde bedrijf, gezien het hier gaat om een systeem op basis van vertrouwen. Berichtjes kunnen immers als

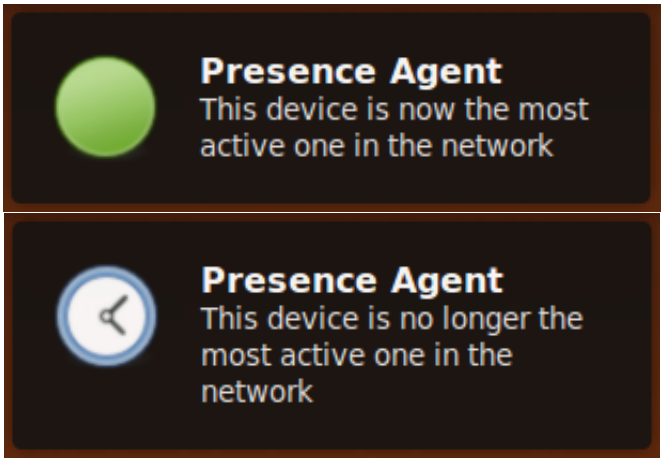
storend ervaren worden en mogen dus enkel verstuurd worden indien ze belangrijk zijn. Deze berichtjes dienen als vervanging voor een SMS: "Kan je even langskomen?" of "Waarom ben je nog niet op de vergadering? We wachten op je!". Ze zullen, indien de gebruiker met geen enkel apparaat bezig is dat een mogelijkheid tot communicatie voorziet, ook automatisch doorgestuurd kunnen worden als SMS naar de mobiele telefoon van de gebruiker.
Uiteraard is het bij het gebruik van deze dienst mogelijk dat de gebruiker zich verplaatst tussen verschillende apparaten. Hierbij moet er dan voor gezorgd worden dat het gesprek verder gaat via het (nieuwe) meest actieve apparaat.

De reden dat zowel de notificaties als de berichten enkel op de meest actieve machine mogen weergegeven worden is dat dit redundantie beperkt. Gebruikers hebben namelijk snel te kampen met het probleem van 'information overload' [Yang 2003]. Dit is nog vervelender indien de informatie die aangeboden wordt redundant blijkt te zijn. We verwijzen hierbij naar de bespreking van een *Active Messenger* in [Schmandt 2000], waarin gehamerd wordt op het niet storen van de gebruiker wanneer dit niet nodig is. Dit is zeker het geval bij berichten van andere gebruikers, gezien deze vaak een antwoord verwachten. Ook zou dit een groter probleem vormen op het Android platform, waar elke notificatie moet aangeklikt worden voor deze verdwijnt. Het zijn dus beide zeer relevante toepassingen voor een presence framework.

2.2 Ubiquitous Computing

De presence clients moeten werken binnen het concept van *Ubiquitous Computing* [Weiser 1991, Wiki 2009]. In dit model gebeurt het verwerken van informatie door alledaagse objecten tijdens alledaagse activiteiten. Bij het gebruik van Ubiquitous Computing is het mogelijk dat de gebruiker van een dienst gebruik maakt zonder dat hij zich er zelf van bewust is. In het geval van de presence client betekent dit dat deze moet werken als deel van de omgeving van de gebruiker. Deze 'omgeving' houdt dan in dat de gebruiker niet langer actief met verschillende apparaten werkt, maar eerder tussen apparaten wisselt zonder dat hij daarbij nadenkt. Bijgevolg zou het enige dat de gebruiker merkt van de aanwezigheid van de presence client moeten liggen in de toepassingen hierop. Een voorbeeld hiervan is de messaging service, waarbij de gebruiker enkel merkt dat berichten die voor hem bestemd zijn aankomen op de machine waar hij op dat moment mee werkt (een manier

waarop dit uiteindelijk zal weergegeven worden kan gevonden worden in 2.1, ter illustratie). Ook kiezen we ervoor om de gebruiker te verwittigen wanneer de machine waarop hij aan het werken is de meest actieve machine wordt op het netwerk en wanneer de machine deze status verliest. Dit doen we om ervoor te zorgen dat de gebruiker zelf kan zien wanneer het 'meest actieve apparaat' foutief bepaald wordt. Wanneer dit het geval zou zijn kan dan bijvoorbeeld in de presence *agent* aangegeven worden dat de gebruiker actiever is op de huidige machine, zodat deze hieruit kan leren [Bamps 2009].



Figuur 2.1: Notificatie van aanwezigheid bij de presence client

Ook kan de presence client als doel hebben *ambient intelligence* [Bieliková 2001] te voorzien. In dit model is een apparaat (of een elektronische omgeving) zich bewust van de aanwezigheid van een gebruiker, en kan het deze informatie gebruiken om bepaalde diensten aan te bieden en om zich aan te passen aan de situatie. Een presence agent kan hier de client gebruiken om zijn relatieve beschikbaarheid binnen het netwerk van apparaten te bepalen, in plaats van enkel zijn absolute beschikbaarheid te kunnen bepalen met gegevens over het apparaat zelf. Zo kan de presence agent een betere schatting maken van de beschikbaarheid van het apparaat. Ook kan in deze context een machine zijn eigen status beter bepalen, om bijvoorbeeld het beeldscherm te blokkeren indien de gebruiker zijn kantoor verlaten heeft.

Verder is het belangrijk dat de gebruiker zelf niet mag kunnen instellen wat zijn beschikbaarheid is (op het corrigeren van foutieve statussen in de presence agent na). Dit lijkt op het eerste zicht een vrij drastische maatregel, maar het biedt een grotere zekerheid dat het niveau van beschikbaarheid klopt aangezien de gebruiker niet kan vergeten zijn status te veranderen. Dit zorgt er ook voor dat andere gebruikers minder snel berichten zullen sturen

wanneer de gebruiker niet beschikbaar is omdat ze er zekerder van kunnen zijn dat deze status correct is [Fogarty 2004].

2.3 Privacy

Het is ook belangrijk dat de privacy van de gebruiker gerespecteerd wordt. Zo moet de aanwezigheid op het apparaat van de eindgebruiker berekend worden. Op die manier wordt enkel een indicatie van de aanwezigheid, en niet de bepalende factoren ervan, over het netwerk verstuurd. Veel gebruikers willen namelijk niet dat andere mensen exact weten waarmee ze op dat moment bezig zijn [Fogarty 2004]. Indien de presence server ervoor kiest de status van een gebruiker telkens mee te delen aan zijn correspondent kan dit dus een probleem vormen. We zullen om deze reden werken met een aanwezigheidsniveau, hetgeen in feite gewoon een getalwaarde is die de aanwezigheid bepaalt. Hierbij geldt dan dat een groter getal overeen komt met een hogere aanwezigheid, zonder dat daarbij moet opgegeven worden hoe we het getal bepaald hebben.

Sommige implementaties van een presence agent kunnen er echter voor opteren om toch extra informatie door te geven, bijvoorbeeld om de correspondent zelf te laten bepalen of de gebruiker gestoord kan worden [Schmandt 2000]. Dit is nog steeds mogelijk. De presence agent moet dan simpelweg met unieke aanwezigheidsniveaus werken. Een mogelijkheid om dit te verwezenlijken is het interpreteren van deze aanwezigheidsniveaus als een serie bits die bepalen op welke manier de gebruiker bezig is met het apparaat (zoals voorgesteld in figuur 2.2). Als we dan zorgen dat de apparaten die de aanwezigheid sterker bepalen de significantere bits zijn, hebben we nog steeds een representatief globaal aanwezigheidsniveau.

toetsenbord (MSB)	muis	webcam	bluetooth (LSB)	aanwezigheids-niveau
1	1	0	1	13
0	1	1	1	7

Figuur 2.2: Het gebruik van unieke aanwezigheidsniveaus voor het exacter bepalen van de activiteit van de gebruiker. MSB en LSB zijn respectievelijk de meest significante bit en de minst significante bit.

2.4 Modulariteit

Aangezien het de bedoeling is dat we een framework ontwikkelen moeten we ervoor zorgen dat dit modulair is. De presence client is namelijk geen losstaand geheel, maar een *service* die moet kunnen communiceren met andere processen zoals de presence agent. Evenzeer is de presence client geen doel op zich, maar eerder een middel voor andere programma's om de aanwezigheid van de gebruiker te kunnen schatten. Om deze reden zullen we ernaar streven om de verschillende onderdelen van de presence client niet te sterk te koppelen.

Ook willen we dat het geheel eenvoudig uitbreidbaar is, zodat we extra diensten zonder te veel moeite kunnen implementeren.

2.5 Schaal

De bedoeling is dat een volledig presence network, met daarin verschillende gebruikers die elk over één of meer apparaten beschikken, op schaal van een bedrijf kan werken. Hierbinnen kan het presence network dan gebruikt worden voor de onderlinge communicatie van verschillende gebruikers.

Belangrijk is dus dat het handig moet zijn om binnen het bedrijfsnetwerk verbinding te maken met de presence server. Gebruikers die thuis werken of op verplaatsing zijn moeten echter ook nog steeds de mogelijkheid hebben om gebruik te maken van dezelfde functionaliteiten als de gebruikers die zich binnen het bedrijfsnetwerk bevinden.

Verder moet er ook voor gezorgd worden dat de presence server gebruikt kan worden in grote bedrijven met meer dan 10.000 werknemers die elk over minstens één machine beschikken. Deze werknemers moeten allemaal onderling kunnen communiceren en deze communicatie moet betrouwbaar zijn. Ook moet de server robuust genoeg zijn en met fouten kunnen omgaan zonder hierbij onverwacht af te sluiten.

Hoofdstuk 3

Implementatie

In deze sectie gaan we enkel in op de implementatie van de presence client en server zelf en zullen we het niet hebben over het gebruik ervan. Meer informatie over het gebruik, alsook extra informatie over de structuur, is te vinden in bijlage A op pagina 42.

3.1 Algemeen

In dit eindwerk werd gebruik gemaakt van verschillende libraries, waaronder Avahi/Zeroconf [Avahi 2009], D-Bus [D-Bus 2009], gtk/libnotify [GTK+ 2009] en de Google Calendar API [Google 2009b]. Omwille van deze reden, en om het mogelijk te maken op een later ogenblik de presence client te integreren met de presence agent, werd geopteerd voor Python [Python 2009] als programmeertaal. Programma's die voor het Android platform bestemd zijn moesten in de programmeertaal Java geschreven worden.

3.2 Connecties

Voor de verbinding tussen server en client maken we gebruik van persistente TCP verbindingen. Deze zijn persistent omdat dit ervoor zorgt dat er niet aan polling gedaan moet worden door de client om aan nieuwe informatie te komen. De client is immers niet altijd eenvoudig bereikbaar aangezien deze zich vaak achter een firewall of achter NAT[1] zal bevinden. Het is dus meestal

[1] NAT staat hier voor *Network Address Translation*, een techniek die vaak door routers gebruikt wordt om ervoor te zorgen dat verschillende apparaten op één netwerk van het-

niet mogelijk voor de server om bij nieuwe informatie een nieuwe connectie met de client te maken. Omdat de updates in bijna realtime moeten gebeuren (berichten moeten immers altijd op het juiste apparaat aankomen) zou dit betekenen dat de client erg vaak zou moeten vragen aan de server of er nieuwe informatie is.

Ook indien de client zich niet achter een firewall of achter NAT zou bevinden (bijvoorbeeld indien er geen externe verbindingen worden toegestaan), zorgen persistente connecties nog steeds voor een voordeel: er moet niet telkens wanneer er nieuwe informatie is een nieuwe connectie aangemaakt worden, hetgeen zou zorgen voor significante overhead. De reden dat we met TCP verbindingen werken is dat we er dan zeker van kunnen zijn dat alle berichten betrouwbaar afgeleverd worden.

Alle berichten die verstuurd worden over deze verbindingen zijn gestructureerd als gewone tekst omdat een bericht telkens over slechts enkele argumenten beschikt die gemakkelijk van elkaar onderscheiden kunnen worden. Een andere optie zou zijn om met een XML representatie te werken, hetgeen echter de implementatie van een presence client voor verschillende platformen (zoals een willekeurige mobiele telefoon) iets moeilijker zou maken.

3.3 Beschikbaarheid

De beschikbaarheid wordt aangegeven door middel van een aanwezigheidsniveau. Deze aanwezigheidsniveaus (in de code *presence levels* genoemd) kunnen eender welke integer waarde aannemen, waarbij de client met het hoogste aanwezigheidsniveau wordt beschouwd als de meest actieve client. Wanneer er twee clients zijn met hetzelfde aanwezigheidsniveau, wordt de client die zijn aanwezigheidsniveau het laatst aanpaste beschouwd als de meest actieve client. De redenering hierachter is dat er een zekere activiteit moet zijn op het apparaat zodat het aanwezigheidsniveau aangepast wordt. Ook beschikt het apparaat dat zijn status het laatst aanpaste waarschijnlijk over nieuwere informatie.

zelfde externe IP adres gebruik maken. Meer informatie over NAT is te vinden in RFC 1631 op `http://tools.ietf.org/html/rfc1631`.

3.4 Presence server

3.4.1 Gebruikersgegevens

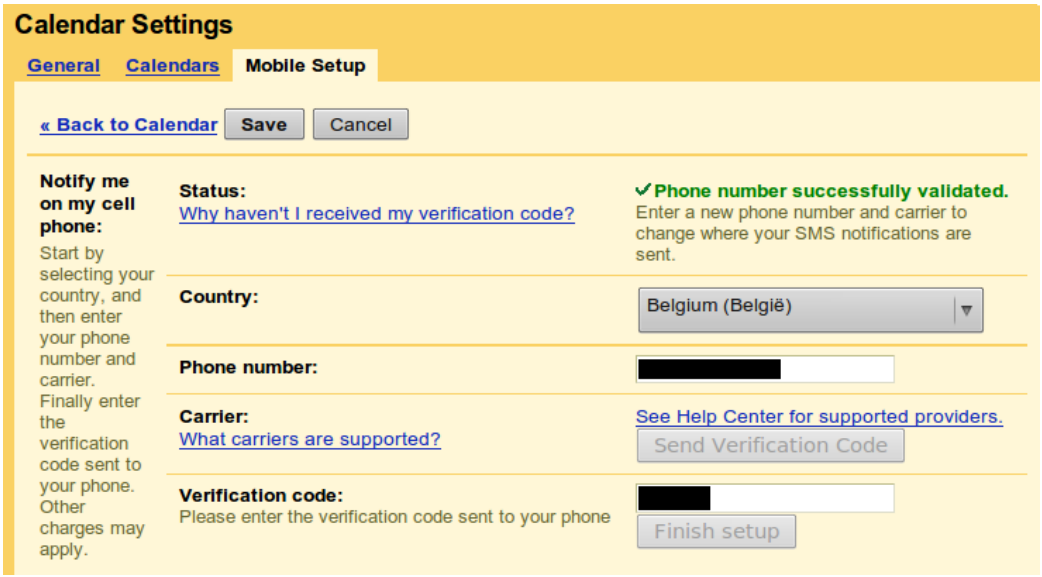
De presence server houdt alle gebruikersgegevens bij in een bestand met de naam `users.dat` in de map waarin hij zich bevindt. Dit bestand wordt automatisch door de server aangemaakt en onderhouden, omdat er met gehashte wachtwoorden[2] gewerkt wordt. Het formaat is als volgt:

```
gebruikersnaam:wachtwoordhash
gebruikersnaam2:wachtwoordhash
gebruikersnaam3:wachtwoordhash:googlegebruikersnaam:googlewachtwoord
...
```

De Google gebruikersnaam en wachtwoord zijn enkel nodig indien de gebruiker van de SMS service gebruik wil maken. Hij dient zich dan eerst op deze Google account aan te melden om vervolgens in zijn Google Calendar instellingen zijn mobiel nummer te registreren, zoals gedemonstreerd in figuur 3.1 op de volgende pagina. De reden dat deze Google wachtwoorden niet gehashed opgeslagen worden (in tegenstelling tot die van de presence server zelf) is dat Google's API niet met sleutels voor geautoriseerde toepassingen werkt (zoals de last.fm API dat bijvoorbeeld doet [Last.fm 2009]) en er dus bij elk gebruik ervan moet ingelogd worden met gebruikersnaam en (een ongecodeerde versie van het) wachtwoord.

Voor het registeren van een nieuwe gebruiker bij de server wordt een request van het formaat `REGISTER:gebruikersnaam:wachtwoord[:googlegebruikersnaam:googlewachtwoord]` gestuurd. De server antwoordt dan met `OK` of `NOK` zodat de gebruiker ziet of de naam al in gebruik was. Het aanmelden gebeurt op een gelijkaardige manier, waarbij een request van het formaat `LOGIN:gebruikersnaam:wachtwoord` wordt verstuurd. Uiteraard zou het registreren bij een server ook via een web service kunnen gebeuren, aangezien de server niet vereist dat de registrerende partij (in dat geval dan een webserver) ook effectief deel zal uitmaken van het presence network van de zojuist geregistreerde gebruiker.

[2] Er wordt gewerkt met een SHA1-gecodeerde versie van een combinatie van gebruikersnaam, wachtwoord en een salt-waarde.

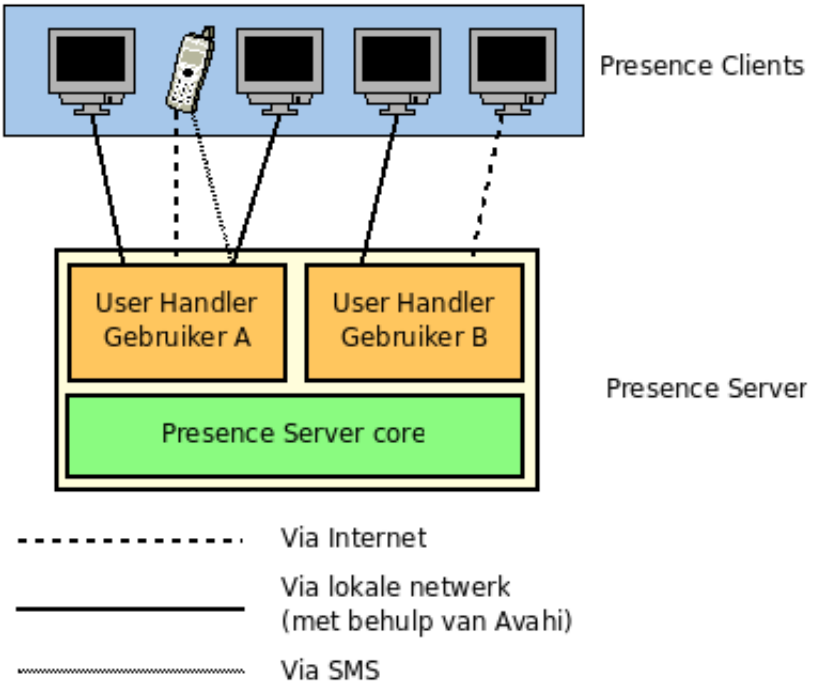


Figuur 3.1: Het registreren van een mobiel nummer bij Google Calendar

3.4.2 User handlers

Om ervoor te zorgen dat de presence server schaalbaar is (zodat hij ook gebruikt zou kunnen worden in grote bedrijven waarin elke werknemer gebruik maakt van één en dezelfde presence server), is ervoor geopteerd om voor elke gebruiker een aparte *user handler* op te starten. Zo'n user handler verzorgt alle transacties en updates van de presence clients van één gebruiker (dit wordt verduidelijkt in figuur 3.2 op de pagina hierna). In een later stadium zouden dan deze user handlers verdeeld kunnen worden over verschillende servers, om ervoor te zorgen dat één server kan gebruikt worden voor het afhandelen van logins en en het koppelen van de juiste client aan de juiste user handler. Ook zorgt deze ene server dan voor het doorgeven van berichten die tussen verschillende user handlers worden doorgestuurd, zoals het geval is wanneer een gebruiker een instant message stuurt naar een andere gebruiker. Wel is het dan zo dat de socket niet gewoon kan doorgegeven worden tussen de user handler en de server (zoals in de huidige implementatie wel gebeurt bij de verschillende user handlers en de server), gezien deze zich mogelijk op een aparte machine bevinden. Een mogelijke oplossing hiervoor is dat de presence server, wanneer deze de juiste user handler voor een bepaalde gebruiker gevonden heeft, het IP adres en de poort van de user handler op de juiste server doorgeeft aan de client, die dan op zijn beurt een nieuwe verbinding zal aanmaken met deze user handler.

We maken in de rest van deze bespreking een duidelijk onderscheid tussen de *server*, die de verschillende user handlers beheert en berichten doorstuurt, en de *user handler*, die alle apparaten van één gebruiker beheert.



Figuur 3.2: Schema van de werking van de presence server. De drie meest linkse apparaten (twee computers en een mobiele telefoon) zijn apparaten van gebruiker A, terwijl de twee rechtse computers apparaten van gebruiker B zijn. De computers worden allemaal verbonden met de presence server via hun presence client. De mobiele telefoon kan ook op deze manier verbonden worden met de presence server. Indien de telefoon geen verbinding met het internet heeft, of zelfs indien deze hier geen ondersteuning voor biedt, kunnen berichten en notificaties ook als SMS ontvangen worden (waarvoor geen presence client op de telefoon nodig is).

De user handler houdt van alle aangesloten apparaten bij wat hun aanwezigheidsniveau is en berekent ook welk apparaat het meest actieve apparaat is. Hij zorgt ervoor dat alle apparaten ook op de hoogte zijn van de status van alle andere apparaten, en dit door telkens wanneer een apparaat zijn activiteitsniveau wijzigt of wanneer er een apparaat het netwerk verlaat deze nieuwe informatie te broadcasten naar alle andere apparaten van deze gebruiker.

Wanneer een client zijn aanwezigheidsniveau wijzigt zal hij meteen een bericht

sturen over de (reeds actieve) verbinding naar de user handler. Deze zal dan herberekenen welke client het hoogste aanwezigheidsniveau heeft om zo te bepalen naar waar instant messages en notificaties moeten doorgestuurd worden, en of er eventueel gebruik gemaakt moet worden van de SMS service (besproken in sectie 3.8 op pagina 22). Vervolgens broadcast de user handler ook de nieuwe lijst naar alle andere clients van deze gebruiker, zodat ook deze weet hebben van de nieuwe aanwezigheidsniveaus.

3.5 Presence client

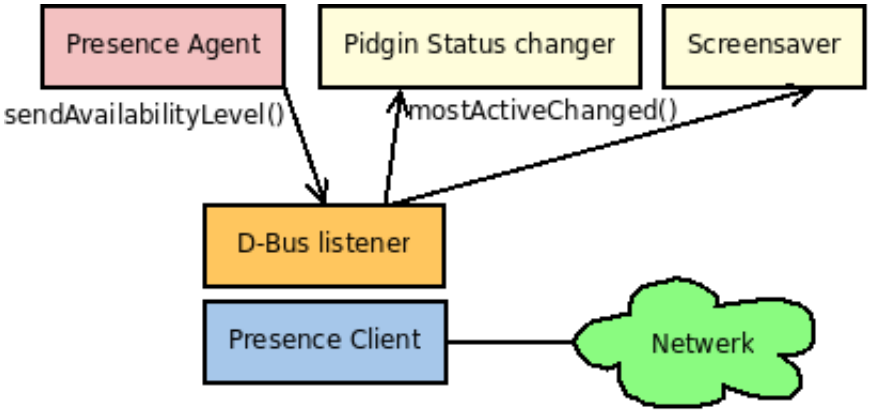
3.5.1 Wat de presence client weet en doet

De presence client krijgt van zijn user handler een volledige lijst van de aanwezigheidsniveaus van alle andere clients van deze gebruiker. Dit werd gedaan omdat een presence agent eventueel deze extra informatie kan gebruiken voor het maken van een betere schatting van de aanwezigheid van de gebruiker, en omdat het geen extra overhead met zich meebrengt (er moet nog steeds maar erg weinig data verstuurd worden). Een andere toepassing van het kennen van de aanwezigheidsniveaus van *alle* apparaten die aan deze gebruiker toebehoren zou het opmerken van ongeautoriseerd gebruik kunnen zijn. Zo kan de presence client op de huidige meest actieve machine van de gebruiker waarschuwen indien er plots activiteit is op één van zijn andere machines. De client houdt ook bij welke van zijn peers het meest actief is, en of hij zelf de meest actieve client in zijn presence network is.

3.5.2 Onder GNU/Linux

Op dit platform is ervoor geopteerd om de bediening van de presence client te laten verlopen met behulp van D-Bus [D-Bus 2009]. D-Bus is een framework dat voor interprocescommunicatie zorgt. Dit is gedaan omdat de presence client kan gebruikt en bestuurd worden door verschillende programma's, die bijvoorbeeld allemaal willen opvragen of de huidige machine de meest actieve client is op het netwerk (om bijvoorbeeld een CPU-intensieve taak die niet dringend is enkel uit te voeren wanneer de gebruiker op een ander apparaat aan het werken is). Een schema van de opbouw wordt gegeven in figuur 3.3 op de pagina hierna. In sectie A.2.2 op pagina 43 wordt uitgelegd welke methods er via D-Bus beschikbaar zijn en hoe deze gebruikt kunnen worden.

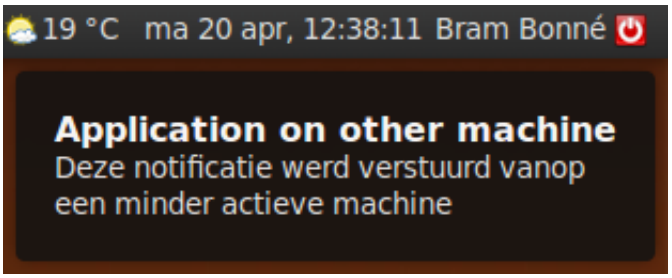
Een programma dat via D-Bus gebruik wil maken van de presence client moet dit echter niet noodzakelijk volledig in D-Bus implementeren. Het bestand



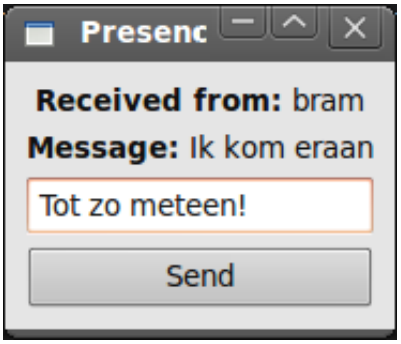
Figuur 3.3: Schema van de presence client onder GNU/Linux.

`dbus-send.py`, dat wordt meegeleverd, kan immers ook door elk programma aangeroepen worden met de juiste argumenten. De manier waarop dit gedaan kan worden is beschreven in sectie A.2.2 op pagina 43.

Voor de notificaties wordt gebruik gemaakt van `libnotify`, hetgeen een library is die ervoor zorgt dat notificaties op een specifieke manier per desktopomgeving worden weergegeven (voor een voorbeeld onder GNOME in Ubuntu 9.04, zie figuur 3.4). Voor het weergeven van berichten willen we ook de mogelijkheid voorzien om meteen een antwoord te versturen naar de correspondent, en opteerden we voor een GTK kadertje (zie figuur 3.5 op de pagina hierna). De bedoeling is hier niet om een chatvenster te maken, gezien deze berichten moeten beschouwd worden als een vervanging van SMS'jes. Dat betekent dat ze enkel verstuurd dienen te worden indien ze belangrijk zijn en dat ze kort moeten zijn. Om dit aan te moedigen en om ervoor te zorgen dat de berichten niet te storend zijn werd er voor een zo minimalistisch mogelijke vormgeving geopteerd.

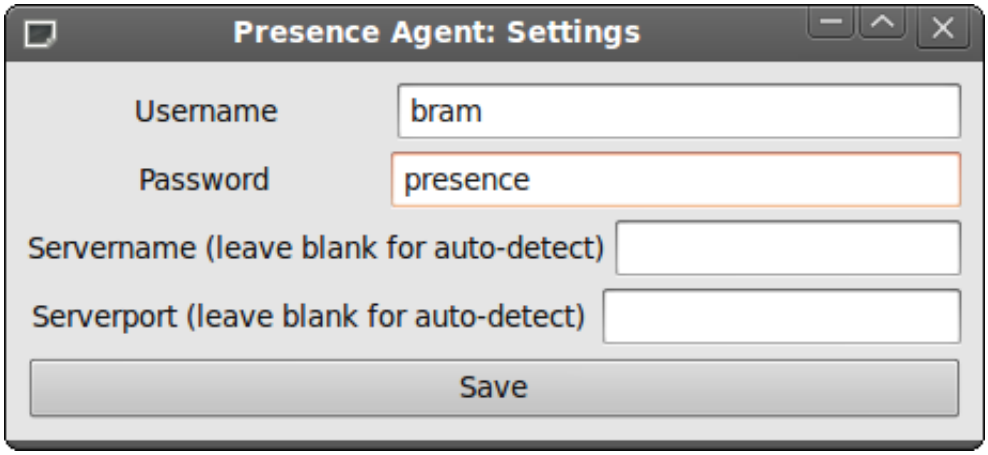


Figuur 3.4: Een ontvangen notificatie onder Ubuntu 9.04. Deze ziet er anders uit afhankelijk van het platform.



Figuur 3.5: Een ontvangen bericht onder GNU/Linux (met behulp van GTK)

De gebruikersconfiguratie wordt opgeslagen in een bestand. Indien dit bestand niet wordt gevonden (of corrupt is geraakt) wordt de aandacht van de gebruiker gevestigd op het instellingen-venster (zie figuur 3.6).



Figuur 3.6: Het instellingen-venster onder GNU/Linux

Het is erg eenvoudig om de gebruikersinterface te vervangen. De interface is immers volledig gescheiden van de rest van de code. Zo is er momenteel een interface beschikbaar die geschreven werd in GTK, maar deze kan makkelijk vervangen worden door bijvoorbeeld een Qt interface. Er is echter niet voor geopteerd om ook de gebruikersinterface via D-Bus te laten werken, aangezien we zeker willen zijn dat berichten afgeleverd worden bij de gebruiker (denk eraan dat we berichten en notificaties met een hoge prioriteit beschouwen) en we niet zonder meer een signaal willen uitsturen waarvan we niet zeker kunnen zijn dat het opgevangen wordt door een gebruikersinterface.

Het is ook vrij eenvoudig de GNU/Linux client uit te breiden met extra

functionaliteit. De eenvoudigste manier om dit te doen is door een D-Bus service te maken die signals van de presence client opvangt (een voorbeeld is te zien in figuur 3.7). Meer uitleg over de beschikbare D-Bus methods kan gevonden worden in de appendix op pagina 43.

```
1 import dbus
2
3 sessBus = dbus.SessionBus()
4 presClient = sessBus.get_object("org.gooz.presenceAgent", "/Client")
5 presClient.sendNotification("Server at location D down")
6 sessBus.add_signal_receiver(mostactive_callback,
7                     signal_name="mostactiveChanged",
8                     dbus_interface="org.gooz.presenceAgent.interface")
```

Figuur 3.7: Voorbeeld van het gebruik van D-Bus voor het sturen van een notificatie en het koppelen van een callback method aan de gebeurtenis waarbij de huidige client verandert van of naar de meest actieve client op het netwerk.

Er zijn momenteel twee extra diensten op deze manier geïmplementeerd:

Pidgin status change Wanneer de client merkt dat hijzelf niet langer de meest actieve client op het netwerk is zal hij de status van de instant messaging-client *Pidgin*[3] op 'afwezig' zetten. Ook deze aanroep van Pidgin methods gebeurt met behulp van D-Bus. De handigheid hiervan is dat protocollen die verschillende machines ondersteunen (zoals XMPP/Jabber) kunnen zien op welke machine de gebruiker het meest actief is, en op die manier een andere vorm van *Everywhere Messaging* aanbieden (hierover meer in sectie 4.4 op pagina 33). Uiteraard zal, wanneer de machine terug als meest actief beschouwd wordt, de status terug naar 'aanwezig' veranderen. Dit draagt bij tot het geautomatiseerd veranderen van status, wat ervoor zorgt dat andere gebruikers minder geneigd zijn te storen als de correspondent gemarkeerd staat als 'afwezig' (zie ook sectie 2.2 op pagina 4 en [Fogarty 2004]).
Het voordeel dat deze methode biedt ten opzichte van een presence agent die zelf de status verandert, is dat de presence *client* ook weet heeft van zijn aanwezigheid binnen het netwerk, in plaats van enkel te weten wat zijn lokale aanwezigheid is. Op die manier kunnen we voor een beter aangepaste status zorgen.

[3]Meer informatie over Pidgin kan gevonden worden op `http://pidgin.im/`

Gnome-screensaver Ook de schermbeveiliging kan automatisch geactiveerd worden wanneer de client inactief wordt. Dit is handig om ervoor te zorgen dat het scherm geblokkeerd wordt wanneer de gebruiker zich niet achter zijn computer bevindt. Jammer genoeg is het activeren van een schermbeveiliging erg systeemafhankelijk, hetgeen de reden is dat we hier enkel gnome-screensaver integratie als proof-of-concept aanbieden. De schermbeveiliging wordt niet automatisch gesloten indien er wel terug aanwezigheid wordt gedetecteerd, om de simpele reden dat we niet kunnen weten of het om de activiteit van een geautoriseerde gebruiker gaat.

Met een D-Bus client voor het veranderen van de Pidgin status en een D-Bus client voor het inschakelen van de screensaver ziet de algemene structuur van onze applicatie er uit zoals in schema 3.3 op pagina 14.

3.5.3 Onder Android

Op het Android platform was de implementatie van een presence client iets ingewikkelder, gezien hierop een applicatie die altijd op de achtergrond moet werken beschouwd wordt als een *service*. Om echter een gebruikersinterface te voorzien moet ook een zogenaamde *activity* aangemaakt worden, die dan verbonden wordt met de service. Ook is de implementatie van een presence client op Android iets minder uitgebreid dan die onder GNU/Linux, omdat hier de nadruk niet op lag voor dit eindwerk. Desalniettemin kunnen er met de Android presence client ook berichtjes verstuurd en ontvangen en notificaties ontvangen worden. Concreet bevat de presence client volgende klassen:

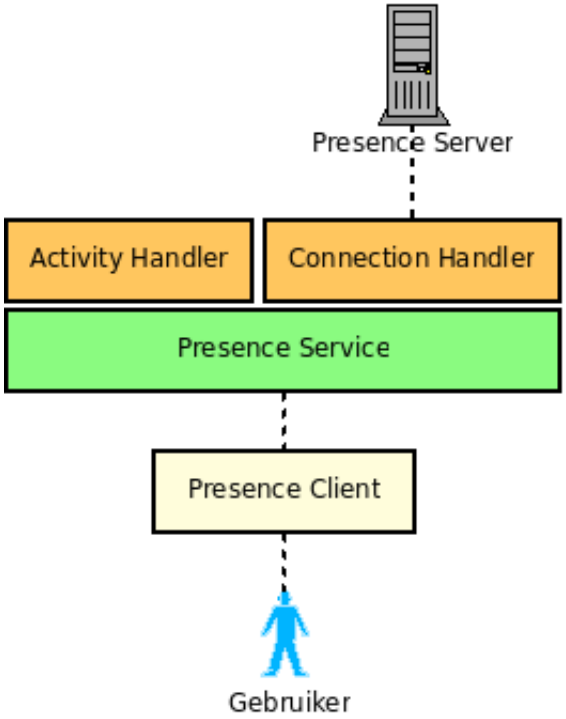
PresenceService Dit is de service die altijd op de achtergrond blijft werken. Hij zorgt ervoor dat de verschillende klassen worden aangemaakt. Ook zorgt hij ervoor dat de ConnectionHandler indien nodig de gebruiker kan verwittigen.

PresenceClient Deze klasse verzorgt de gebruikersinterface en zorgt ervoor dat de Presence Service gestart kan worden.

ConnectionHandler Deze klasse zorgt voor de communicatie met de server.

ActivityHandler Deze klasse houdt bij of het apparaat gebruikt wordt, door te kijken of het scherm geblokkeerd is. Hiervoor wordt gebruik gemaakt van de (in de documentatie goed verborgen) KeyguardManager [Google 2009e].

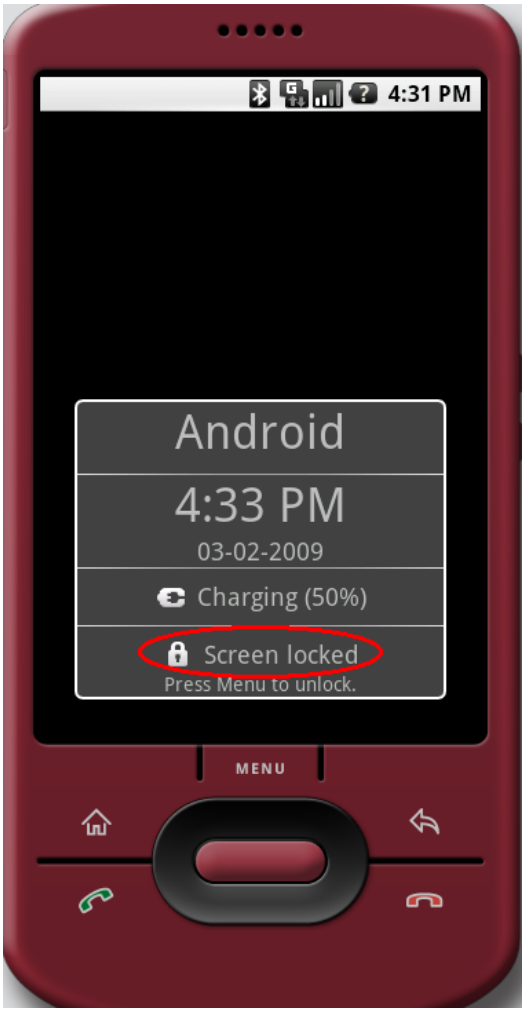
Een schema hiervan wordt getoond in figuur 3.8.



Figuur 3.8: Schema van de werking van de presence client op het Android platform.

De Android client heeft slechts twee aanwezigheidsniveaus. Wanneer de gebruiker bezig is op het apparaat zal de client een aanwezigheidsniveau 10 naar de server sturen. Wanneer dit niet zo is wordt aanwezigheidsniveau 0 doorgestuurd. Deze aanwezigheid wordt geschat door te controleren of de gebruiker het scherm geblokkeerd heeft, zodat er geen interactie mogelijk is (zie figuur 3.9 op de pagina hierna). Dit is een goede indicatie van afwezigheid, gezien het scherm ook automatisch geblokkeerd zal worden na een periode van inactiviteit (zoals een schermbeveiliging). De aanwezigheidsniveaus 0 en 10 werden zo gekozen om een representatieve waarde te hebben in combinatie met de presence agent uit [Bamps 2009].

Wanneer een bericht of notificatie ontvangen wordt, zal deze eerst weergegeven worden in het mededelingengebied (zie figuur 3.10a op pagina 20). Wanneer de gebruiker op dit gebied klikt, ziet hij een drop-down menu (zie figuur 3.10b) en worden de notificaties/berichten automatisch gewist uit het mededelingengebied. Voor berichten is er dan ook nog de mogelijkheid te klikken op de tekst, zodat een antwoord gestuurd kan worden naar de correspondent.



Figuur 3.9: Geblokkeerd scherm op Android

(a) Het ontvangen van een nieuw bericht of notificatie wordt bij de meldingen weergegeven.

(b) Wanneer de melding aangeklikt wordt ziet de gebruiker een lijst van zijn ontvangen berichten en notificaties. Van hieruit kan er ook geantwoord worden.

Figuur 3.10: Bericht of notificatie ontvangen op Android

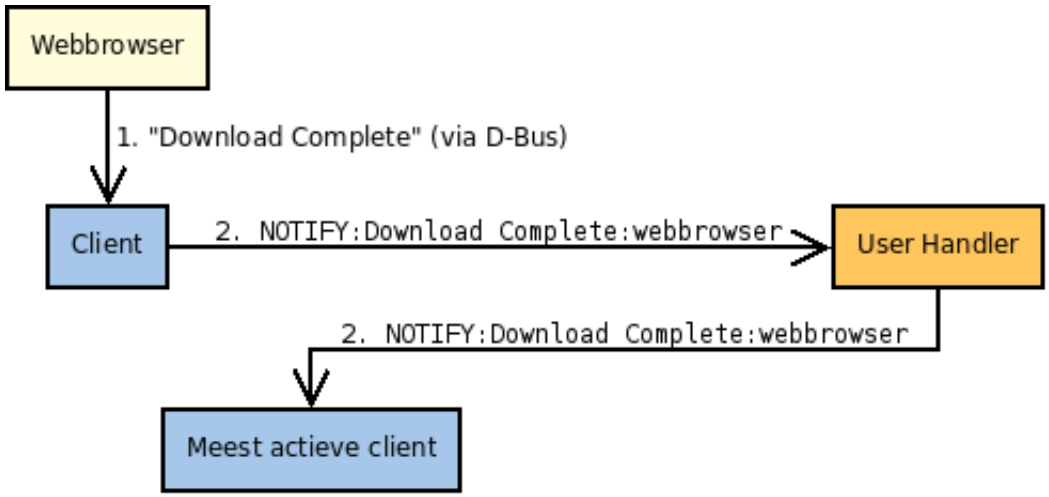
3.6 Notificaties

Bij het versturen van een notificatie worden volgende stappen uitgevoerd (schematisch weergegeven in figuur 3.11):

1. De verzendende client verstuurt een bericht van de vorm `NOTIFY:bericht[:applicatie]` naar zijn user handler. Het meegeven van de versturende `applicatie` is optioneel. Wanneer deze niet wordt meegegeven zal als applicatienaam 'Multi-device presence agent' gebruikt worden.

2. De user handler zoekt op welke client het meest actief is op zijn netwerk (deze waarde wordt bijgehouden in een variabele en herberekend bij elke update) en verstuurt de notificatie naar deze client.

3. De meest actieve client geeft (afhankelijk van het platform, zie sectie 3.5 op pagina 13) de notificatie weer. Dit 'platform' kan evenzeer een SMS zijn indien de user handler geen clients vond met een aanwezigheidsniveau dat boven een bepaalde waarde ligt.



Figuur 3.11: Afhandeling van notificaties

3.7 Berichten

Bij het versturen van een bericht worden volgende stappen uitgevoerd (schematisch weergegeven in figuur 3.12 op pagina 23):

1. De verzendende client verstuurt een bericht van de vorm `MSG:gebruiker:bericht` naar zijn user handler.

2. De user handler vraagt aan de presence server de user handler van de ontvanger. Erg belangrijk hierbij is dat de server er ook voor kan opteren om een object van het type `SmsService` (zie sectie 3.8) terug te geven, in plaats van een echte user handler. Dit kan bijvoorbeeld gebeuren indien de gebruiker op dat moment met geen enkel apparaat op het netwerk aanwezig is, als een laatste optie om het bericht toch af te kunnen leveren.

3. De user handler vraagt nu aan de ontvangende user handler (of de SMS service) om het bericht af te leveren bij de meest actieve gebruiker.

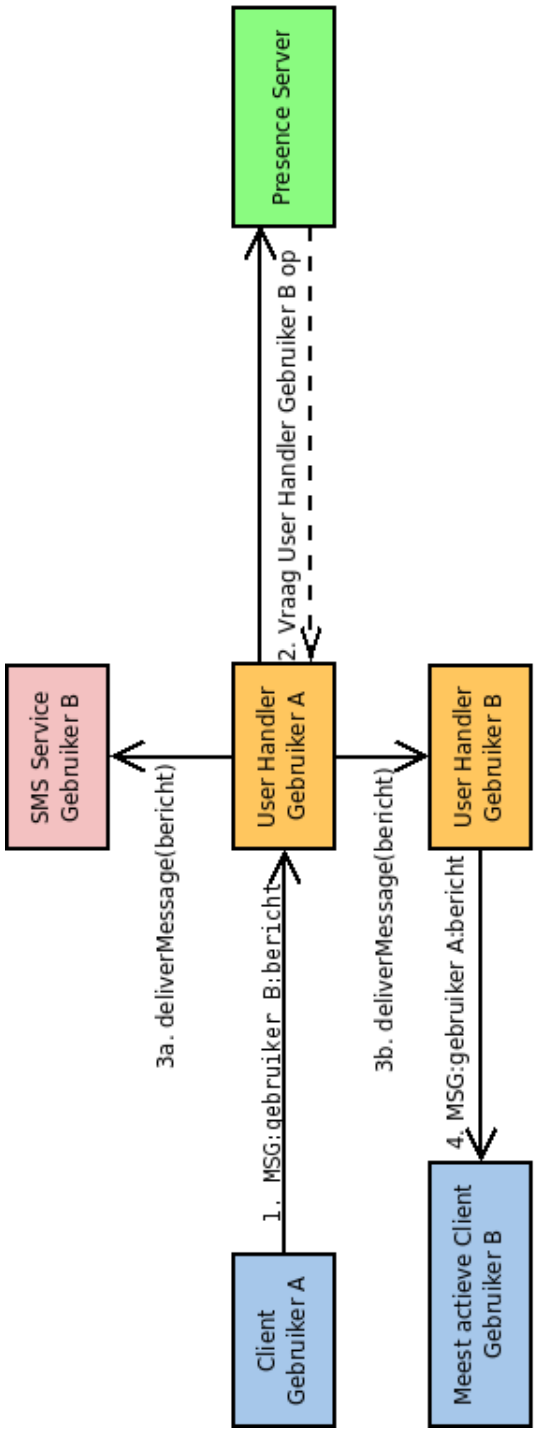
4. Vervolgens verloopt het proces analoog aan stappen 2 en 3 bij het afleveren van een notificatie.

3.8 De SMS service

Om ervoor te zorgen dat zowel berichten als notificaties ook als SMS naar de gebruiker gestuurd kunnen worden, wordt gebruik gemaakt van de gratis SMS-dienst die Google aanbiedt voor zijn Calendar-dienst en van de API hiervoor [Google 2009b]. Op deze manier kunnen we ervoor zorgen dat het gebruik van SMS'jes voor notificaties volledig gratis is voor de eindgebruiker, met als enige vereiste dat deze beschikt over een Google account (die uiteraard ook eenvoudig gratis aan te maken is [Google 2009f]).

Wat Google aanbiedt is de mogelijkheid om een SMS'je te sturen om de gebruiker te herinneren aan een afspraak die in de nabije toekomst op zijn kalender gepland staat. In onze client maken we hier gebruik van door een afspraak aan te maken binnen twee minuten, waarbij een herinnering wordt ingesteld één minuut voor aanvang van de afspraak. Op deze manier zorgen we ervoor dat de notificatie/het bericht gegarandeerd wordt afgeleverd met een maximale vertraging van één minuut. De tijd wordt, indien er een internetverbinding beschikbaar is, opgevraagd bij een lijst betrouwbare NTP[4] servers. Indien er geen internetverbinding beschikbaar is wordt de lokale tijd van de server zelf genomen, die echter minder betrouwbaar is. De reden dat

[4]NTP staat voor *Network Time Protocol*. Dit is een protocol dat gebruikt wordt voor het synchroniseren van de interne klok van computers over een netwerkverbinding. Meer informatie hierover kan gevonden worden in [IETF 1992].



Figuur 3.12: Schema voor het afhandelen van berichten. De SMS Service kan hierbij ook gezien worden als een soort user handler voor de mobiele telefoon, met echter maar één specifieke functie: het afleveren van berichten en notificaties in de vorm van een SMS bericht.

we kiezen voor een herinnering binnen één minuut en niet voor een herinnering op dit exacte moment ligt in het feit dat het zou kunnen dat de herinnering niet wordt afgeleverd indien de klokken van de Google server en de server waar we de tijd vandaan halen niet exact gelijk lopen. We beschouwen immers het gegarandeerd afleveren van een bericht als belangrijker dan het meteen afleveren van het bericht. De reden dat we de tijd opvragen bij NTP servers is dat serverklokken niet altijd even betrouwbaar zijn. De kans is dus groot dat een (betrouwbare) NTP server ons een exactere tijd biedt dan degene die we van de lokale server zouden krijgen.

Omdat de Google Calendar SMS dienst normaal gezien enkel bedoeld is voor herinneringen aan afspraken, is er een strikte manier waarop de SMS'jes opgesteld zijn. Een bericht dat als SMS is aangekomen wordt weergegeven in figuur 3.13.

Figuur 3.13: Een bericht dat aankomt als SMS. De gebruiker die het bericht verstuurde heeft hier als gebruikersnaam 'bram'.

3.9 Opnieuw verzenden

Bij het versturen van een bericht of notificatie naar de meest actieve machine zou het kunnen dat de gebruiker zich net naar een andere machine verplaatst, waarbij de presence *agent* op deze machine de verandering in aanwezigheid nog niet heeft opgemerkt. Om deze reden houdt de presence server ook een lijst bij van berichten en notificaties die recent verstuurd werden, samen met de tijd waarop dit gebeurde. Deze tijd wordt op dezelfde manier bepaald als bij het versturen van een SMS, namelijk door het aanvragen van de huidige tijd bij een NTP server met als fallback de lokale server. Wanneer dan binnen een bepaalde tijd (standaard 20 seconden, maar deze waarde is instelbaar) een ander apparaat van de gebruiker als meest actief wordt aangegeven worden deze berichten en notificaties opnieuw verstuurd, ditmaal naar de nu meest actieve machine. De timestamps worden voor deze berichten dan ook opnieuw op de huidige tijd gezet voor het geval dat dit scenario zich opnieuw zou voordoen. Voor het gebruik van NTP bij het opvragen van de huidige tijd gelden dezelfde opmerkingen als voor het gebruik van NTP bij de SMS service (zie sectie 3.8 op pagina 22).

Op deze manier kunnen we er zeker van zijn dat alle berichten en notificaties zeker worden afgeleverd bij de gebruiker. Dit is nodig omdat de berichten en notificaties die door de presence client verstuurd worden geacht worden een hoge prioriteit te hebben. Het met zekerheid afleveren van de berichten en notificaties weegt dus soms op tegen het vervelende effect van het dubbel ontvangen van een bericht.

3.10 De locatie service

Het is ook mogelijk om te bepalen op welke locatie de meest actieve machine van een gebruiker zich bevindt. Hiervoor maakt de presence server gebruik van de gratis geolocation API van IP Info DB [IPInfoDB 2009]. Hieraan geeft de user handler het IP adres van de gebruiker waar de locatie van opgevraagd wordt. Wanneer de gebruiker zich binnen het lokale netwerk bevindt, zal de server dit ook melden aan de vragende partij.

Concreet zijn dit de stappen die ondernomen worden voor het achterhalen van het IP adres van een gebruiker:

1. De client die vraagt naar de locatie van een gebruiker verstuurt een bericht van de vorm `LOCATE:gebruiker` naar zijn user handler.

2. De user handler vraagt aan de presence server de user handler van de andere gebruiker.

   (a) Indien deze niet bestaat wordt het bericht "User not logged in" teruggegeven. De gebruiker die om de locatie vroeg weet dan ook dat, wanneer hij een bericht naar de andere gebruiker zal sturen, dit bericht zal aankomen als SMS.

   (b) In het andere geval vraagt de server aan de ontvangende user handler wat de locatie is van zijn meest actieve client.

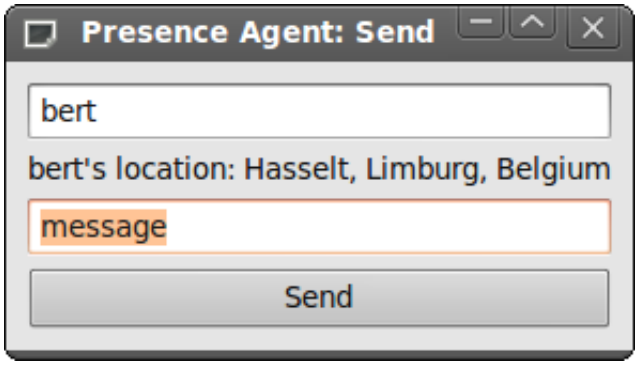
      i. Deze user handler doet een request bij de geolocation service, met daarin het IP adres van zijn meest actieve client.

      ii. Indien het IP adres een lokaal adres blijkt te zijn zal de server zijn eigen extern IP adres opvragen bij een web service [ShowMyIP 2009]. Dit is dan hetzelfde IP adres als dat van de client. Vervolgens wordt voor dit IP adres gekeken wat de locatie is.

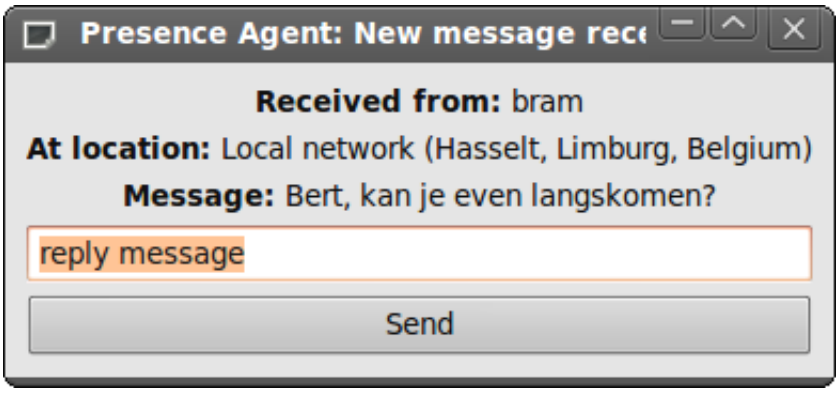
3. Alles wordt in omgekeerde volgorde weer doorgegeven, tot bij de verzoekende client (die de locatie binnen krijgt als een bericht van de vorm `LOCATION:gebruiker:locatie`).

Op deze manier kan bijvoorbeeld in een bedrijf gekeken worden of een bepaalde persoon zich binnen het bedrijf zelf bevindt of, indien dit niet het geval is, op welke locatie hij zich bevindt. Ook dit kan handig zijn in combinatie met het versturen van berichten: op voorhand kan immers gekeken worden of de gebruiker in de buurt is. Deze laatste mogelijkheid werd geïmplementeerd in deze bachelorproef. Een voorbeeld van hoe het versturen en ontvangen van berichten eruit ziet wanneer de locatie service gebruikt wordt is weergegeven in figuren 3.14 op de volgende pagina en 3.15 op de pagina hierna. Een voorbeeld van hoe het versturen van en het antwoorden op berichten eruit ziet op het Android platform kan terug gevonden worden in figuur 3.16 op pagina 28.

Deze dienst is vooral handig bij het gebruik van mobiele apparaten zoals een Android-telefoon of een notebook, maar kan evenzeer gebruikt worden wanneer een bedrijf over meerdere kantoren zou beschikken.



Figuur 3.14: Een nieuw bericht versturen met gebruik van de locatie service. De locatie wordt pas weergegeven wanneer de naam van de correspondent werd ingevoerd. Indien de correspondent op dat moment met geen enkel apparaat op het netwerk aanwezig is, zal er expliciet vermeld worden dat het bericht als SMS zal aankomen.



Figuur 3.15: Het ontvangen van een bericht met gebruik van de locatie service. Het venster wordt meteen weergegeven bij het ontvangen van een bericht. Indien de geolocation server last heeft van vertraging zal dit de snelle ontvangst van het bericht niet beïnvloeden: de locatie wordt dan toegevoegd wanneer de server een antwoord geeft.

(a) Versturen van een nieuw bericht (b) Antwoorden op een ontvangen bericht

Figuur 3.16: Versturen van berichten op het Android platform

3.11 Automatische discovery met behulp van Avahi

Aangezien het de bedoeling is dat een presence server vooral binnen een bedrijf gebruikt wordt, moet het mogelijk zijn om de presence clients binnen het netwerk automatisch de presence server te laten vinden. Dit gebeurt met behulp van Avahi [Avahi 2009]. Avahi is een implementatie van Zeroconf, waarmee het mogelijk is om bepaalde services te broadcasten en te ontdekken op een netwerk. Voor meer informatie over Zeroconf, zie [Cheshire 2009].

Het blijft echter nog steeds mogelijk om rechtstreeks met een server naar keuze te verbinden, bijvoorbeeld voor gebruikers die via Android op de server worden aangesloten of gebruikers die vanuit verschillende netwerken met dezelfde presence server willen verbinden.

3.12 Robuustheid

Voor een server is het extra belangrijk dat hij niet crasht en dat hij zolang dat het nodig is zijn dienst kan blijven aanbieden zonder daarbij uit te vallen. Dankzij de abstractie die Python biedt (waarin elke mogelijke fout een exception is) is het relatief eenvoudig om onverwachte fouten op te vangen en de server terug te herstellen naar een toestand waarin hij verder kan werken. Dit is bijvoorbeeld niet altijd mogelijk in C(++), waarin de kans bestaat dat er foutief geheugen wordt aangesproken en waarin buffer overflows kunnen voorkomen, waardoor het programma onherroepelijk crasht. Exceptions worden dan ook uitgebreid opgevangen en onderzocht in de presence server en bijhorende user handlers. Bij het gebruik van de location service wordt er ook op gelet dat de servers waarmee verbonden moet worden een reactie sturen en dat er een internetverbinding beschikbaar is, zodat de user handlers nooit te lang blijven wachten op een antwoord.

Ook is er bij het begin van de implementatie intensief met Unit Tests gewerkt. Dit verliep vrij moeizaam gezien er getest moet worden over een (onvoorspelbaar) netwerk, maar liet toch toe om de kern van de presence client en de presence server uitvoerig te testen. Op deze manier werden nog enkele bugs ontdekt die zich voordeden in uitzonderlijke situaties. Deze bugs zijn nu uiteraard verholpen.

Hoofdstuk 4

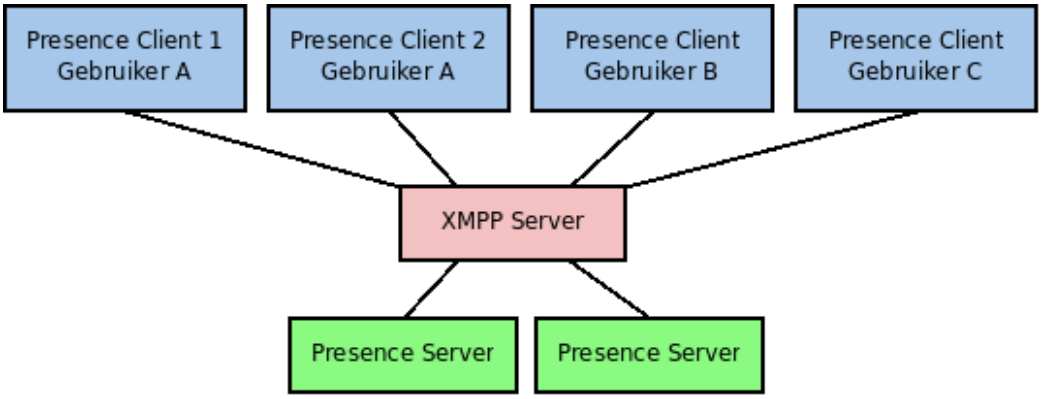
Vergelijking met XMPP

Bij het maken van deze bachelorproef werden er ook (basis-)versies van de presence server en client ontwikkeld die gebruik maken van het XMPP protocol. Dit is gedaan omdat het XMPP protocol duidelijke raakvlakken heeft met wat een presence client/server combinatie zou moeten doen [Moffitt 2008]. In dit hoofdstuk bespreken we overeenkomsten en verschillen tussen onze implementatie en de manier waarop XMPP werkt.

4.1 Werking van de XMPP implementatie

Eerst en vooral moet vermeld worden dat ervoor gekozen is om ook de presence server zelf als een client te laten werken op het XMPP netwerk. De presence server is dus een XMPP bot, gebouwd met behulp van de `xmpppy` library [xmpppy 2009]. Met andere woorden: er wordt een bestaande XMPP server gebruikt (een lijst van zulke servers kan gevonden worden op [XMPP 2000]) en de presence server werkt hierop alsof hij een normale client is. Dit wordt verduidelijkt in figuur 4.1 op de pagina hierna.

Achteraf gezien was het misschien een beter idee geweest de presence server te laten bestaan uit een set uitbreidingen op een XMPP server. Er zal dan ook in de tekst vaak vermeld worden op welke manier een XMPP server de diensten van een presence server zou kunnen leveren. Tenzij anders vermeld wordt er in dit hoofdstuk echter altijd verwezen naar de implementatie waarin de presence server een XMPP bot op het netwerk is.



Figuur 4.1: Schema van de XMPP opstelling waarbij de presence server zich gedraagt als een client op het XMPP netwerk. Alle getekende lijnen zijn netwerkverbindingen (mogelijk over het internet).

4.2 Connecties

Er is een vrij grote overeenkomst in de manier van verbinding maken tussen de clients en de server bij de beide implementaties. In beide gevallen zijn deze connecties persistent (in het geval van XMPP via de XMPP server), zodat er niet onnodig aan polling gedaan moet worden. Dit zorgt namelijk voor een aanzienlijke overhead, vooral aangezien er telkens een nieuwe TCP connectie zou moeten opgezet worden wanneer er een bericht verstuurd wordt (we willen immers betrouwbare gegevensoverdracht).

Het systeem van persistente connecties in onze implementatie kan dus vergeleken worden met het XMPP PubSub mechanisme [Henshaw-Plath 2008]. Hierbij zal een client subscriben op een server[1] waarbij de server ervoor zal zorgen dat hij, telkens wanneer hij nieuwe data heeft, de nieuwe gegevens publiceert naar de client. In onze implementatie fungeren dan zowel de presence client als de presence server tegelijk als server en client op XMPP niveau. De client publiceert namelijk zijn aanwezigheidsniveau naar de server, terwijl de server het aanwezigheidsniveau van alle andere clients op het netwerk naar de client publiceert. Dit is wat er in feite ook gebeurt in een standaard XMPP scenario, waarbij de verschillende clients op een XMPP server hun status publiceren naar de server, die op zijn beurt deze status weer publiceert naar alle andere personen die deze client in hun lijst van contacten hebben staan.

[1] We gebruiken hier de term 'server' om aan te geven dat dit een XMPP client is die als server fungeert. Indien we het over een XMPP server hebben zullen we dit er altijd bij vermelden.

Bedenk echter dat we hier nog steeds werken met een XMPP client, en niet met een XMPP server als presence server.

De aanwezigheidsniveaus worden ook verstuurd via het in XMPP ingebouwde presence mechanisme, waarbij statussen het XML formaat hebben. Dit is vooral handig omdat de meeste instant messaging clients deze statussen ook ondersteunen. Het versturen van een gewijzigde lijst van activiteitsniveaus naar alle clients gebeurt door het versturen van een standaard XMPP presence bericht, waarbij we ervoor zorgen dat de standaard tags van het bericht worden verwijderd zodat instant messaging clients het bericht niet zullen proberen te interpreteren als een normaal presence bericht. In plaats hiervan wordt dan een `<presencelist>` tag ingevoegd, met als kinderen de verschillende presence clients die als attributen een naam en een aanwezigheidsniveau hebben. Dit is gelijkaardig aan de manier waarop SOAP berichten over XMPP kunnen verstuurd worden [Forno 2005], met het verschil dat we hier het geheel inkapselen in een `<presence>` tag in plaats van een `<message>` tag (waarom dit zo is bespreken we in sectie 4.4 op de volgende pagina).

Een verschil is wel dat er bij XMPP geen problemen kunnen optreden wanneer de server zich achter NAT bevindt, omdat het verbinden van de presence client met de presence server gebeurt via een XMPP server. We moeten echter opmerken dat dit eigenlijk een verschuiving van het probleem is, gezien nu de XMPP server zich niet achter NAT kan bevinden. Doordat er gewerkt wordt met XML in plaats van gewone tekst zorgt XMPP ook voor een iets grotere overhead, doch deze is verwaarloosbaar.

4.3 Schaalbaarheid

Een probleem bij het gebruik van XMPP zou kunnen liggen in de schaalbaarheid van het programma. Bij de standaard implementatie van de presence client wordt gebruik gemaakt van user handlers, die elk de apparaten van één enkele gebruiker voor hun rekening nemen. Omdat bij XMPP alle berichten van alle gebruikers binnen een netwerk via één XMPP client moeten (de server is als bot immers ook maar gewoon een 'client' op het XMPP netwerk) kan deze XMPP client een bottleneck zijn. Dit is echter voornamelijk speculatie gezien de XMPP presence server niet op grote schaal werd getest.

Een oplossing voor dit eventuele probleem zou zijn om voor elke gebruiker een eigen presence server te voorzien die dienst doet als user handler. Er moet dan niet alleen voor gezorgd worden dat de clients geauthenticeerd

zijn bij de server (zodat ze onderling berichten kunnen uitwisselen), maar ook dat de servers onderling geauthenticeerd zijn. Het zou immers anders niet mogelijk zijn voor verschillende gebruikers om berichten naar elkaar te sturen via de server. Merk wel op dat XMPP reeds een ingebouwd instant messaging systeem heeft, zodat het ook niet noodzakelijk is dat er via de presence server gecommuniceerd wordt. We zijn dan echter weer te zeer op server niveau aan het kijken, waarbij de XMPP server zelf dienst zou doen als presence server om er zo voor te zorgen dat berichten enkel bij de meest actieve client afgeleverd worden. Om deze reden zou de XMPP server ook meer geschikt zijn voor het versturen van berichten dan een XMPP bot.

4.4 Berichten en notificaties

XMPP is in eerste instantie een instant messaging protocol. Het voordeel hiervan is dat XMPP specifieke diensten biedt voor het versturen van berichten naar andere gebruikers. Zo is er bijvoorbeeld al een dienst die ervoor zorgt dat, wanneer de gebruiker op meerdere instant messaging clients met dezelfde account aangemeld is, chatberichten automatisch zullen aankomen op de instant messaging client waarop deze gebruiker de hoogste prioriteit heeft. Dit is echter geen verplichte functionaliteit. Het hangt dus van de XMPP server af of hier gebruik van gemaakt wordt. [IETF 2004] zegt hierover:

> For message stanzas, the server SHOULD deliver the stanza to the highest-priority available resource (if the resource did not provide a value for the `<priority/>` element, the server SHOULD consider it to have provided a value of zero). If two or more available resources have the same priority, the server MAY use some other rule (e.g., most recent connect time, most recent activity time, or highest availability as determined by some hierarchy of `<show>` values) to choose between them or MAY deliver the message to all such resources.

Belangrijk om hierbij op te merken is dat de statussen die door de presence client worden verstuurd voor dit doel gebruikt kunnen worden. Dit is ook de reden dat we de presence lists die van de server naar de clients gaan inkapselen in een `<presence>` tag in plaats van een `<message>` tag. Het zou anders kunnen gebeuren dat de server de presence lists enkel naar de client met de hoogste `<priority>` stuurt, gezien de meeste servers berichten met

`<message>` als root-tag zullen beschouwen als een bericht dat enkel naar de meest actieve client verstuurd moet worden.

Een oplossing die we zouden kunnen gebruiken bij onze implementatie van de presence server zou erin bestaan om berichten nog steeds via de *presence* server te laten gaan, die ze dan doorstuurt naar de juiste gebruiker als een gewoon XMPP bericht. Ook notificaties kunnen op deze manier behandeld worden, met het verschil dat ook hiervoor (net zoals voor de presence lists die verstuurd worden) een aparte XML representatie voorzien moet worden. Dit bericht zou dan uiteraard wel ingekapseld worden in een `<message>` root-tag in plaats van een `<presence>` tag.

Een andere mogelijkheid is om de XMPP account gewoon in te stellen in een instant messaging-applicatie zoals Pidgin en de status hiervan te updaten met de presence client (zoals besproken in sectie 3.5.2 op pagina 13). Op die manier zal de gebruiker nog steeds gebruik kunnen maken van de routing diensten die de XMPP server aanbiedt voor instant messages, terwijl de presence server ervoor zorgt dat alle clients van een bepaalde gebruiker op de hoogte zijn van elkaars presence en er notificaties verstuurd kunnen worden.

4.5 Conclusie

Uit deze vergelijking tussen onze presence server en de XMPP server leren we voornamelijk dat het XMPP protocol al veel functionaliteit voorziet waarvan in dit eindwerk de implementatie werd onderzocht. Zo zorgt het XMPP protocol standaard al voor een service waarbij berichten enkel verstuurd worden naar de meest actieve client.

We kunnen, om een XMPP implementatie van de presence server te voorzien, wel beter werken op het niveau van de XMPP server dan op het niveau van een XMPP client. Hierbij is elk apparaat dan simpelweg een XMPP client op het netwerk, waarvan het aanwezigheidsniveau (in de vorm van een `<priority>`) wordt ingesteld door de presence agent. Twee extra's die de presence server biedt (de locatie service en het automatisch herverzenden) zouden dan als uitbreidingen op de XMPP server geschreven kunnen worden.

Voor het voorzien van een SMS service zijn er twee mogelijkheden:

- Er kan per gebruiker een bot aangemaakt worden met hetzelfde XMPP ID als deze gebruiker, maar met een erg lage `<priority>`. Wanneer een bericht aankomt bij de bot stuurt deze het bericht door als een SMS'je naar de mobiele telefoon van de gebruiker (eventueel door gebruik te

maken van de Google Calendar API). Een bericht dat aankomt bij de bot impliceert immers dat alle andere clients van deze gebruiker een erg lage `<priority>` hebben en dus niet beschikbaar zijn.

- Op de XMPP server kan een SMS transport (ook wel gateway genoemd) voorzien worden. Een transport zorgt ervoor dat gebruikers van het XMPP protocol ook kunnen communiceren met mensen die op andere instant messaging services aangemeld zijn. Dit gebeurt volledig op server niveau, waarbij het voor de gebruiker lijkt alsof hij met een andere XMPP gebruiker communiceert. Er bestaan reeds enkele basis implementaties van een SMS transport [Palij 2007, Sun 2008]. Meer informatie over XMPP transports kan gevonden worden op [Wiki 2009b].

Het voordeel van het gebruik van XMPP zou zijn dat we werken met een gestandaardiseerd protocol [IETF 2004], hetgeen voor verhoogde interoperabiliteit zorgt. De specificatie van XMPP werd ook onderzocht en aangevuld door verschillende personen, wat een hogere garantie biedt op volledigheid. Verder is ook Google Talk, een veelgebruikt protocol voor instant messaging, gebaseerd op XMPP [Google 2009g].

Een eventueel nadeel van het gebruik van XMPP zou zijn dat extra diensten zoals SMS, automatisch opnieuw verzenden van berichten en de locatie service niet in de specificatie zijn opgenomen. Er is echter al een draft specificatie voor het verkrijgen van de geografische of fysieke locatie van een gebruiker over XMPP [Hildebrand 2008].

Hoofdstuk 5

Gebruikerstesten

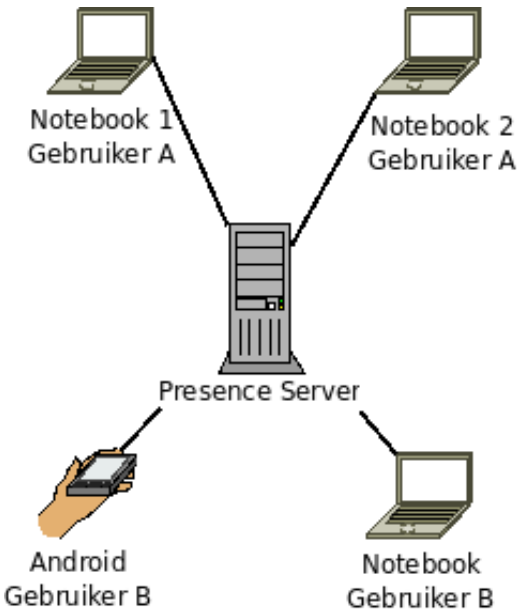
Bij het afronden van deze bachelorproef zijn er ook informele testen uitgevoerd met echte gebruikers. Hiervoor werd de presence client gecombineerd met de presence agent uit [Bamps 2009].

5.1 Opstelling

De opstelling die gebruikt werd voor de gebruikerstesten is weergegeven in figuur 5.1 op de volgende pagina. De afgebeelde Android client was echter geen fysiek Android toestel, maar een notebook met daarop een Android emulator. Ook bevond de presence server zich niet op een aparte server, maar op een notebook waarop ook één van de clients geïnstalleerd was. De presence server was echter volledig onzichtbaar voor de testers, zodat dit geen substantieel verschil opleverde.

Er zijn wegens tijdsgebrek slechts vier (twee keer twee) personen geweest die de applicatie getest hebben. Deze personen zaten allen in het derde bachelorjaar Informatica. De applicatie is immers vooral bedoeld voor mensen die vertrouwd zijn met computers: ze moeten beschikken over meerdere op een netwerk aangesloten apparaten en maken gebruik van een dienst die pas nuttig is wanneer ze veel tijd achter hun computer doorbrengen. Toch werd er expliciet aan de gebruikers gevraagd enkel op het end-user aspect te letten en zich dus niet bezig te houden met hoe de applicatie intern zou kunnen werken.

Er werd aan de gebruikers gevraagd hun apparaten te gebruiken zoals ze dat normaal zouden doen en daarbij af en toe te wisselen tussen de twee



Figuur 5.1: De opstelling die gebruikt werd voor het uitvoeren van de gebruikerstesten.

apparaten. Ook werd gevraagd om enkele berichten te versturen naar de andere gebruiker, en om te antwoorden op berichten van de deze gebruiker.

5.2 Vragenlijst

Na het testen werd aan de gebruikers gevraagd een korte vragenlijst in te vullen over het gebruik van de applicatie. Deze vragenlijst bestond uit volgende vragen:

- Gaf de presence client steeds voor het juiste apparaat de status 'meest actief'? Indien niet: wat ging er mis? Onder GNU/Linux wordt steeds een notificatie weergegeven indien de huidige machine van/naar meest actief verandert.
- Verliep het versturen van berichten zoals verwacht? Kwamen de berichten steeds op de machine aan waar u op dat moment mee aan het werken was?
- Werd (in de mate dat u dat zelf wist) de locatie van de correspondent steeds correct weergegeven? Indien niet: wat ging er mis?

- Werkte de gebruikersinterface intuïtief? Was het steeds duidelijk wat er van u verwacht werd en wat de mogelijke opties waren? Vermeld ook welk platform gebruikt werd (GNU/Linux of Google Android).
- Was de presence client storend bij het gewone computergebruik? Welke verbeteringen zijn volgens u mogelijk? Houd hierbij in acht dat het de bedoeling is de presence client enkel in vertrouwde omgevingen te gebruiken, waarbij de ander nooit bewust gestoord zal worden.
- Werd, indien u van apparaat wisselde terwijl er net een bericht aankwam, dit bericht opnieuw op de nieuwe meest actieve machine (waarnaar u wisselde) weergegeven? Indien ja: ervaart u dit als storend of als handig?
- Verdere opmerkingen?

5.3 Bevindingen

Een eerste opvallend resultaat is dat alle gebruikers de chatvensters als vrij vervelend ervoeren. Deze zouden storend zijn bij het uitvoeren van gewoon werk. Hierbij moet wel worden opgemerkt dat bij het testen van de applicatie meer berichten werden verstuurd dan er zouden verstuurd worden bij het normale gebruik ervan. Een mogelijke oplossing voor dit probleem zou zijn om de chatvensters niet meteen te laten verschijnen op het scherm, maar in plaats hiervan een notificatie te tonen waarop geklikt kan worden om het chatvenster te voorschijn te laten komen.

Ook vonden de gebruikers het vervelend dat verschillende berichten van eenzelfde gebruiker in verschillende pop-up vensters getoond werden. Dit zou eenvoudig opgelost kunnen worden door het chatvenster van een bepaalde persoon te updaten wanneer er nieuwe berichten van deze persoon aankomen in plaats van hiervoor telkens een nieuw venster te openen.

Verder begrepen enkele gebruikers ook niet waarom berichten opnieuw aankwamen wanneer er kort na het ontvangen van een bericht gewisseld werd naar een ander apparaat. Ook hier moet wel in het achterhoofd gehouden worden dat er bij de testen veel vaker (en sneller) van apparaat gewisseld werd dan bij het normale gebruik van deze applicatie het geval zou zijn. Een eventuele oplossing zou kunnen zijn om bij het opnieuw versturen van berichten aan te geven dat het gaat om een bericht dat opnieuw verstuurd werd. De kans is echter vrij groot dat het probleem zich hier enkel voordoet omdat alle apparaten zo dicht bij elkaar stonden en er een grote hoeveelheid berichten

verstuurd werd. Één persoon gaf expliciet aan het automatisch opnieuw versturen van berichten handig te vinden.

Buiten voorgaande punten waren de antwoorden op de in sectie 5.2 vermelde vragen positief. De presence client bleek naar behoren te werken in vrij realistische use cases, waarbij berichten altijd op de juiste machine aankwamen en de locatie telkens juist werd weergegeven. De problemen lijken zich dus vooral te bevinden in de user interface en kunnen gelukkig vrij gemakkelijk verholpen worden door het maken van kleine aanpassingen aan de buitenste schil van de presence client.

Hoofdstuk 6

Conclusie

Het is in dit eindwerk vrij goed gelukt om een presence network te voorzien op een eenvoudige manier. Essentieel hebben we namelijk gewoon een server die voor elke gebruiker een lijst van apparaten en hun activiteitsniveau moet bijhouden, en die ervoor moet zorgen dat alle apparaten ook up-to-date aanwezigheidsinformatie hebben van alle andere apparaten. Om deze reden hebben we de implementatie van de kern van het presence network simpel kunnen houden, zodat het uitbreiden ervan met extra functionaliteit makkelijk ging. Dit is ook de reden dat instant messaging, notificaties (beiden met opnieuw verzenden bij het veranderen van de status), de locatie service en het versturen van SMS'jes met succes geïmplementeerd konden worden.

Mede dankzij het gebruik van D-Bus hebben we er ook voor kunnen zorgen dat het geheel zeer modulair en uitbreidbaar is. Extra diensten die gebruik maken van de informatie van de presence client, of die informatie voorzien voor de presence client, kunnen met behulp van D-Bus op een handige manier communiceren met deze client. Het bewijs hiervan wordt geleverd door het kleine aantal (amper 70) regels code dat nodig is voor een module waarin zowel Pidgin als gnome-screensaver integratie voorzien wordt. Ook was het door het gebruik van D-Bus helemaal niet moeilijk om de presence client te integreren in de presence agent uit [Bamps 2009]. Er moesten slechts vier regels toegevoegd worden aan de bestaande code van de presence agent om ervoor te zorgen dat het presence level van de client altijd up-to-date is en dat bijgevolg het hele presence network weet wat het aanwezigheidsniveau van het apparaat is.

Tenslotte merkten we op dat een XMPP server op een zeer gelijkaardige manier kan werken als de presence server, doordat ook deze de mogelijkheid voorziet om berichten enkel naar de meest actieve instant messaging ap-

plicatie op het netwerk te versturen. Toch kan de presence server enkele voordelen bieden ten opzichte van een standaard XMPP server. Zo is er de SMS service, die weet heeft van de Google account van een gebruiker en die een bericht of notificatie dus kan forwarden naar zijn mobiele telefoon. Ook is er de locatie service, die aan de verzender kan laten weten op welke locatie de ontvanger zich bevindt. Deze beide vormen van extra functionaliteit zouden echter ook als uitbreiding op een XMPP server (of in het geval van de SMS service ook als een bot op het XMPP netwerk) voorzien kunnen worden, hetgeen helaas buiten het bereik van dit eindwerk ligt.

Bijlage A

Handleiding voor ontwikkelaars

A.1 Presence server

A.1.1 Vereisten

De volgende libraries dienen geïnstalleerd te worden voordat de presence server gebruikt kan worden:

- Python: de presence server werd getest onder Python versies 2.5 en 2.6.
- Google Data API voor Python: deze library is nodig om SMS'jes te kunnen versturen. Hij kan gevonden worden op [Google 2009c].
- `python-avahi` (of `avahi-tools`): dit is enkel nodig als we de client automatisch de server willen laten detecteren. Deze library kan via het package management systeem geïnstalleerd worden in de meeste courante GNU/Linux distributies.

A.1.2 Gebruik

De presence server kan simpelweg gestart worden met het commando:

```
./presenceServer.py
```

Het inloggen en alle andere interactie gebeurt dan volledig vanuit de presence clients. Het registreren op de server kan gedaan worden met behulp van het volgende script:

```
./register.py [serverIP poort]
```

Indien er geen IP van de server wordt opgegeven wordt met Avahi op het lokale netwerk gezocht naar een server.

A.2 Presence client onder GNU/Linux

A.2.1 Vereisten

- Python: de presence client werd getest onder Python versies 2.5 en 2.6.
- `python-notify`: deze library is nodig zodat Python het standaard notificatie-systeem van de desktop kan gebruiken, en is meestal standaard geïnstalleerd. Indien dit niet het geval is, kan hij in de meeste courante GNU/Linux distributies via het package management systeem geïnstalleerd worden. `python-notify` is geen vereiste als we de presence client enkel willen gebruiken om de aanwezigheid van de gebruiker te bepalen, of als we het voldoende vinden dat de notificaties en berichten enkel in de console weergegeven worden.
- `python-gtk2`: hiervoor gelden dezelfde opmerkingen als voor `python-notify`.
- `python-dbus`: deze library is enkel nodig indien we de presence client standalone willen gebruiken en we hem via interprocescommunicatie willen bedienen (dit wordt verder besproken in sectie A.2.2).
- `python-avahi` (of `avahi-tools`): dit is enkel nodig als we de client automatisch de server willen laten detecteren. Deze library kan via het package management systeem geïnstalleerd worden in de meeste courante GNU/Linux distributies.

A.2.2 Gebruik

De presence client kan op twee manieren gebruikt worden: als standalone client die bediend wordt via interprocescommunicatie, of als library binnen een presence agent. We bespreken hieronder beide methoden.

Om de presence client als standalone client te gebruiken dient deze gestart te worden met het commando:

```
./presenceClientDBusListener.py
```

Er zal dan een D-Bus service [D-Bus 2009] gestart worden die enkele methods publiceert voor gebruik door andere programma's. Deze methods worden opgesomd in tabel A.1. Een voorbeeld van het gebruik ervan (in Python) kan gevonden worden in figuur A.1 op de pagina hierna. Het is echter niet noodzakelijk dat de presence client via een Python programma bediend wordt. Een implementatie van D-Bus is namelijk beschikbaar voor een groot aantal programmeertalen, waaronder C(++) en Java.

Method	Uitleg
`sendAvailabilityLevel(level)`	Stuur een nieuw aanwezigheidsniveau naar de server.
`sendNotification(message, application)`	Stuur een notificatie naar de meest actieve machine van de gebruiker. De `application` parameter is optioneel.
`sendMessageToUser(username, message)`	Stuur een bericht naar de meest actieve machine van de gebruiker met als gebruikersnaam `username`.
`showNewMessageScreen()`	Geef een venster weer waarmee het mogelijk is een nieuw bericht naar een willekeurige gebruiker te versturen. Dit is een end-user interface voor `sendMessageToUser()`, zodat deze D-Bus method ook gewoon in een executable kan opgeroepen worden.
`mostactiveChanged(self, isMostActive)`	Een signaal dat wordt uitgestuurd wanneer de huidige client verandert naar of van meest actieve client in het netwerk.

Tabel A.1: D-Bus methods aangeboden door de presence client

Voor het gemak is ook het bestand `dbus-send.py` meegeleverd, dat kan gebruikt worden om via D-Bus berichten te sturen naar de presence client zonder dat hiervoor speciaal een service moet opgestart worden. Het gebruik ervan is zeer eenvoudig: een bericht kan verstuurd worden met het commando:

```
./dbus-send.py
```

Met daarachter één van de volgende argumenten:

```
1 import dbus
2
3 sessBus = dbus.SessionBus()
4 presClient = sessBus.get_object("org.gooz.presenceAgent", "/Client")
5 presClient.sendAvailabilityLevel(5)
```

Figuur A.1: Voorbeeld van het gebruik van D-Bus voor het instellen van aanwezigheidsniveau 5

level:<integer> Verstuurt een nieuw aanwezigheidsniveau.

notify:<notificatie-tekst>[:<versturende applicatie>] Indien <versturende applicatie> niet wordt meegegeven, wordt standaard 'Multi-device presence agent' gebruikt.

message[:<ontvanger>:<bericht>] Indien ontvanger en bericht niet worden opgegeven, wordt er een venster geopend waarmee een bericht verstuurd kan worden. Voor de eindgebruiker zou dus ook gewoon een snelkoppeling naar '`<map van de presence client>/dbus-send.py message`' in het menu of op het bureaublad gezet kunnen worden, zodat eenvoudig berichtjes verstuurd kunnen worden. Specifiek voor dit doel is echter ook het bestand `newMessageSender.py` meegeleverd.

De presence client kan echter ook als Python object door een presence agent of binnen een bestaand framework gebruikt worden. Hierbij wordt aangeraden nog steeds gebruik te maken van het DBusListener object (hetgeen in feite gewoon een wrapper is rond een PresenceClient object) zodat ook andere programma's gebruik kunnen maken van deze instantie van de presence client.

Indien het echter gewenst is de presence client af te schermen voor andere programma's, of indien geen D-Bus beschikbaar is op de machine, kan de presence client zelf als aparte library gebruikt worden. De uitleg per method is dan te vinden in de Python docstrings[1] (die gebruikt wordt door de meeste Python editors). Het is dankzij deze docstrings ook mogelijk om Doxygen-achtige documentatie te genereren. Dit kan gedaan worden met behulp van het programma epydoc[2].

[1] Docstrings zijn strings van de vorm """documentatie""" die zich net onder de klassenaam bevinden in de definitie ervan. Voor meer informatie over Python docstrings, zie `http://www.python.org/dev/peps/pep-0257/`.

[2] epydoc kan gevonden worden op `http://epydoc.sourceforge.net/`

A.3 Presence client onder Google Android

Om de presence client op het Android platform te gebruiken, kan met behulp van de Android SDK (die werkt onder de Eclipse ontwikkelomgeving[3]) [Google 2009d] een Android package gegenereerd worden. Dit werd echter niet getest bij het maken van de bachelorproef, gezien er geen Android telefoon ter beschikking was. De presence agent werd wel getest op de Android emulator die wordt meegeleverd met de SDK.

[3]Eclipse is beschikbaar op `http://www.eclipse.org/`

Bijlage B

Handleiding voor eindgebruikers

B.1 Presence server

De presence server kan simpelweg gestart worden met het commando:

```
./presenceServer.py
```

Het inloggen en alle andere interactie gebeurt dan volledig vanuit de presence clients. Het registreren op de server kan gedaan worden met behulp van het volgende script:

```
./register.py [serverIP poort]
```

Indien er geen IP van de server wordt opgegeven wordt er op het lokale netwerk gezocht naar een server.

B.2 Presence client onder GNU/Linux

B.2.1 Starten

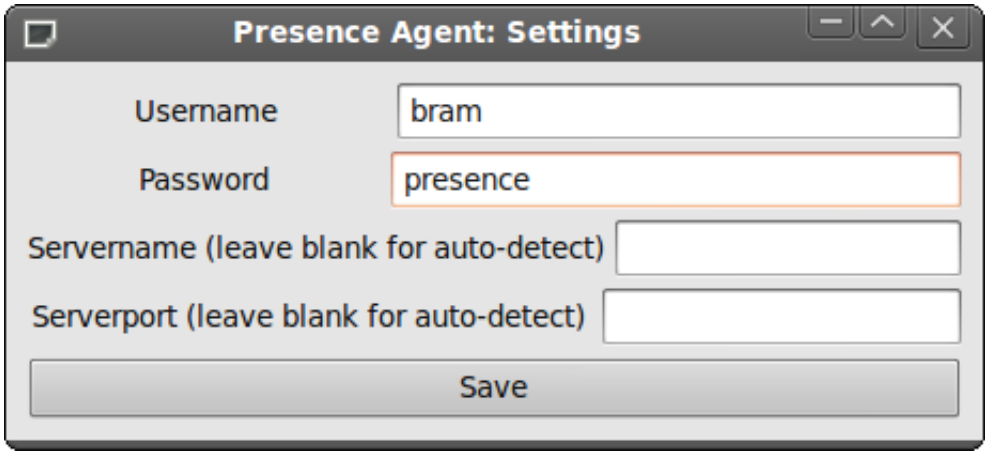
De presence client kan best gestart worden met het commando:

```
./presenceClientDBusListener.py
```

Merk op dat er ook een presence agent gestart dient te worden zodat de status correct aangepast wordt. Meestal is het niet nodig om de presence client expliciet te starten, gezien een standaard implementatie van een presence agent deze al automatisch zal starten.

B.2.2 Aanmelden

Bij het opstarten van de presence client / presence agent wordt er een venster getoond dat eruit ziet zoals in figuur B.1. Geef in dit venster je gebruikersnaam en wachtwoord in. Vul daaronder het adres van de server in. Indien deze server zich op het lokale netwerk bevindt mag je de laatste twee velden gewoon leeg laten.



Figuur B.1: Het instellingen-venster onder GNU/Linux

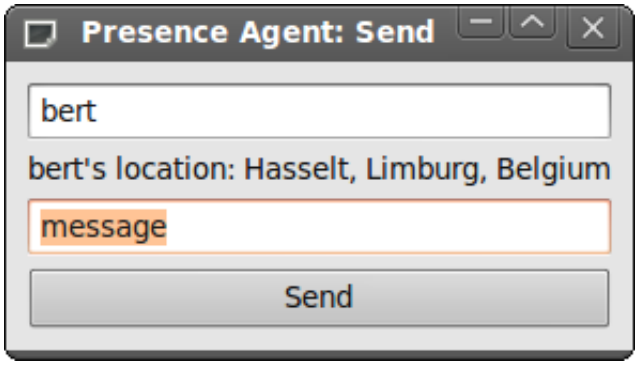
Als alles gelukt is verdwijnt het venster en ben je aangemeld bij de server. Er is voor de rest geen interactie nodig, de presence client kan nu gewoon zijn werk doen.

B.2.3 Berichten versturen

Een bericht kan verstuurd worden met het commando:

```
./newMessageSender.py
```

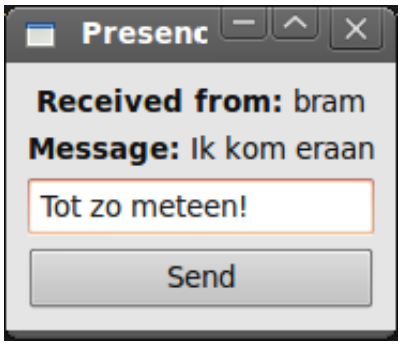
Er wordt dan een venster geopend (zoals weergegeven in figuur B.2 op de pagina hierna) waarin je de naam van de andere gebruiker en het bericht kan ingeven. Wanneer de presence client nog geen verbinding heeft met de server zal dit gemeld worden in een notificatie. Merk ook op dat het mogelijk is dat je bericht aankomt op de mobiele telefoon van de andere gebruiker. Dit wordt dan weergegeven bij de locatie. Let erop dat je de andere persoon niet stoort!



Figuur B.2: Een nieuw bericht versturen met gebruik van de locatie service. De locatie wordt pas weergegeven als de naam van de correspondent werd ingevoerd. Indien de correspondent op dat moment met geen enkel apparaat op het netwerk aanwezig is, zal er expliciet vermeld worden dat het bericht als SMS zal aankomen.

B.2.4 Berichten ontvangen

Om een bericht te ontvangen moet je niets speciaals doen. Wanneer een andere gebruiker je een bericht stuurt zie je een venster zoals in figuur 3.5. Je kan dan in dit venster de andere persoon antwoorden door een bericht in te voeren in het tekstveld. Als je niets wil antwoorden moet je enkel het venster sluiten.



Figuur B.3: Een ontvangen bericht onder GNU/Linux

B.3 Presence client/agent onder Google Android

B.3.1 Starten en aanmelden

Om de presence client/agent te starten klik je op 'Multi-device presence agent' in het applicatiemenu. Er wordt een scherm getoond zoals in figuur B.4. Vul hierin je gebruikersnaam en wachtwoord in. Vul ook het webadres samen met de poort van de server in. Indien je niet zeker bent welke poort je moet gebruiken, vul dan 9742 in als poort.

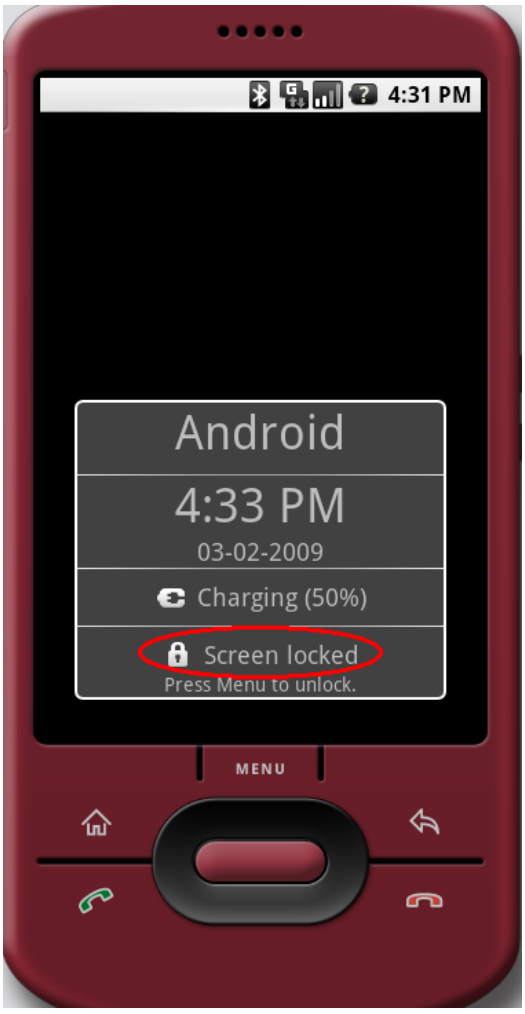
Figuur B.4: Aanmelden op Android

Als alles goed verlopen is zie je nu een scherm zoals in figuur B.6a op pagina 52 (links), waarop je berichten kan versturen (zie sectie B.3.3). Dit venster mag

je sluiten, de presence client/agent blijft dan op de achtergrond werken.

B.3.2 Aanwezigheid

De aanwezigheid op het Android toestel wordt gedetecteerd door te kijken of je scherm geblokkeerd is. Dit gebeurt automatisch na enkele seconden van inactiviteit. Indien je handmatig het scherm wil blokkeren, druk dan op de knop met de rode telefoon. Om het scherm terug te deblokkeren druk je op de knop met de groene telefoon. Een voorbeeld van een geblokkeerd scherm vind je in figuur B.5.



Figuur B.5: Geblokkeerd scherm op Android

B.3.3 Berichten versturen

Om een bericht te versturen start je de presence client/agent opnieuw vanuit het applicatiemenu terwijl je aangemeld bent. Er wordt dan een scherm getoond zoals in figuur B.6a. Klik in dit scherm op de knop 'Send message' en je krijgt een scherm zoals in figuur B.6b. Hierop kan je de naam van de correspondent en het bericht ingeven. Klik tenslotte op 'Send message' om het bericht te versturen.

(a) Het 'running' scherm dat getoond wordt wanneer je aangemeld bent

(b) Het scherm waarin een bericht verstuurd kan worden

Figuur B.6: Versturen van een nieuw bericht op Android

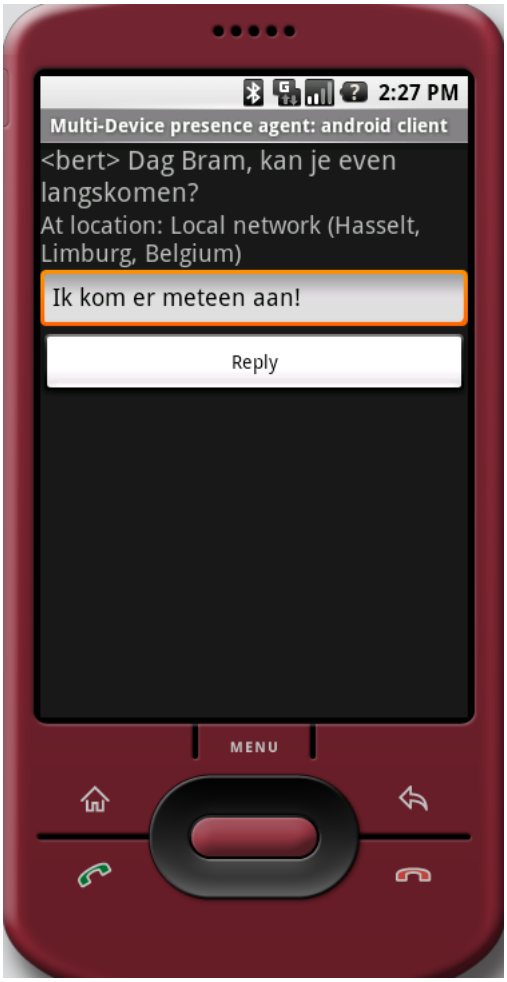
B.3.4 Berichten ontvangen

Bij het ontvangen van een nieuw bericht krijg je een notificatie te zien zoals in figuur B.7a. Wanneer je deze notificatie naar de onderkant van je scherm sleept zie je alle ongelezen berichten en notificaties (zoals weergegeven in figuur B.7b). Klik op één van deze berichten om ze als gelezen te markeren. Je komt dan in het antwoordscherm (figuur B.8 op de pagina hierna) terecht, hetgeen je gewoon kan sluiten als je niet wil antwoorden op het laatst ontvangen bericht.

(a) Wanneer een nieuw bericht wordt ontvangen is dit zichtbaar als notificatie

(b) Uitgeklapte berichten en notificaties

Figuur B.7: Bericht of notificatie ontvangen op Android



Figuur B.8: Antwoorden op een bericht op Android

Bibliografie

[Avahi 2009] 'About Avahi', http://avahi.org/wiki/AboutAvahi, 2009 [bezocht op 16 februari 2009] 3.1, 3.11

[Bamps 2009] K. Bamps, 'Presence Agent', 2009 (document), 1, 2.2, 3.5.3, 5, 6

[Bieliková 2001] M. Bieliková & T. Krajcovic, 'Ambient Intelligence within a Home Environment', ERCIM News vol 47, 2001 2.2

[Cheshire 2009] S. Cheshire, 'Zero Configuration Networking (Zeroconf)', http://www.zeroconf.org/, 2009 [bezocht op 16 februari 2009] 3.11

[D-Bus 2009] 'What is D-Bus?', http://www.freedesktop.org/wiki/Software/dbus, 2009 [bezocht op 9 maart 2009] 3.1, 3.5.2, A.2.2

[Fogarty 2004] J. Fogarty, J. Lai, J. Christensen, 'Presence versus Availability: The Design and Evaluation of a Context-Aware Communication Client', 2004 2.2, 2.3, 3.5.2

[Forno 2005] F. Forno, P. Saint-Andre, 'XEP-0072: SOAP Over XMPP', http://xmpp.org/extensions/xep-0072.html, 2005 [bezocht op 9 april 2009] 4.2

[Google 2009] 'Android - An Open Handset Alliance Project', http://code.google.com/intl/nl/android/, 2009 [bezocht op 19 februari 2009] 1

[Google 2009b] 'Google Calendar API', http://code.google.com/intl/nl/apis/calendar/docs/1.0/developers_guide_python.html, 2009 [bezocht op 24 maart 2009] 3.1, 3.8

[Google 2009c] 'Google Data Python API', http://code.google.com/p/gdata-python-client/, 2009 [bezocht op 24 maart 2009] A.1.1

[Google 2009d] 'Google Android SDK', http://developer.android.com/sdk/1.1_r1/index.html, 2009 [bezocht op 19 februari 2009] A.3

[Google 2009e] 'Android KeyguardManager', http://developer.android.com/reference/android/app/KeyguardManager.html, 2009 [bezocht op 25 februari 2009] 3.5.3

[Google 2009f] 'Google Account signup', https://www.google.com/accounts/NewAccount, 2009 [bezocht op 24 maart 2009] 3.8

[Google 2009g] 'Google Talk for developers', http://code.google.com/apis/talk/open_communications.html, 2009 [bezocht op 22 juni 2009] 4.5

[GTK+ 2009] 'What is GTK+?', http://www.gtk.org/, 2009 [bezocht op 26 februari 2009] 3.1

[Henshaw-Plath 2008] E. Henshaw-Plath, K. Elliott-McCrea, 'Beyond REST? Building data services with XMPP PubSub', http://www.slideshare.net/rabble/beyond-rest-building-data-services-with-xmpp-pubsub, 2008 [bezocht op 9 april 2009] 4.2

[Hildebrand 2008] J. Hildebrand, P. Saint-Andre, 'XEP-0080: User Location', http://xmpp.org/extensions/xep-0080.html, 2008 [bezocht op 22 juni 2009] 4.5

[IETF 1992] D. L. Mills, 'Network Time Protocol (Version 3): Specification, Implementation and Analysis', http://tools.ietf.org/html/rfc1305, 1992 [bezocht op 20 mei 2009] 4

[IETF 2004] P. Saint-Andre et. al., 'Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence', http://www.ietf.org/rfc/rfc3921.txt, 2004 [bezocht op 9 april 2009] 4.4, 4.5

[IPInfoDB 2009] 'IP Address Location XML API', http://www.ipinfodb.com/ip_location_api.php, 2009 [bezocht op 13 mei 2009] 3.10

[Last.fm 2009] 'Last.fm API user authentication', http://www.last.fm/api/authentication, 2009 [bezocht op 26 mei 2009] 3.4.1

[Moffitt 2008] J. Moffitt, 'Exploring XMPP', http://www.slideshare.net/codebits/exploring-xmpp-presentation, 2008 [bezocht op 9 april 2009] 4

[Python 2009] 'About Python', http://www.python.org/about/, 2009 [bezocht op 10 februari 2009] 3.1

[Palij 2007] O. Palij, 'mod_sms - SMS (Short Message Service) transport', http://www.dp.uz.gov.ua/o.palij/mod_sms/, 2007 [bezocht op 22 juni 2009] 4.5

[Schmandt 2000] C. Schmandt et al., 'Everywhere messaging', IBM Systems Journal vol 39, 2000 2.1, 2.3

[ShowMyIP 2009] 'External IP address API', http://www.showmyip.com/simple/, 2009 [bezocht op 13 mei 2009] 2(b)ii

[Sun 2008] 'SMS Gateway for Instant Messaging', http://wikis.sun.com/display/CommSuite/SMS+Gateway+for+Instant+Messaging, 2008 [bezocht op 22 juni 2009] 4.5

[Weiser 1991] Mark Weiser, 'The Computer for the Twenty-First Century', Scientific American September, 1991 2.2

[Wiki 2009] Wikipedia medewerkers, 'Ubiquitous Computing', http://en.wikipedia.org/wiki/Ubiquitous_computing, 6 april 2009 [bezocht op 13 april 2009] 2.2

[Wiki 2009b] Wikipedia medewerkers, 'Extensible Messaging and Presence Protocol: Connecting to other protocols', http://en.wikipedia.org/wiki/XMPP#Connecting_to_other_protocols, 21 juni 2009 [bezocht op 23 juni 2009] 4.5

[Williams 2000] M. Williams, 'Surf Among Suds With Web-Enabled Washing Machine', http://www.pcworld.com/article/32128/surf_among_suds_with_webenabled_washing_machine.html, 2000 [bezocht op 13 april 2009] 2.1

[XMPP 2000] 'XMPP Software: Servers', http://xmpp.org/software/servers.shtml, 2009 [bezocht op 9 april 2009] 4.1

[xmpppy 2009] 'The xmpppy project', http://xmpppy.sourceforge.net/, 2009 [bezocht op 9 april 2009] 4.1

[Yang 2003] C.C. Yang et al., 'Visualization of Large Category Map for Internet Browsing', Decision Support Systems vol 35, Issue 1, 2003

2.1