Auteursrechterlijke overeenkomst

Opdat de Universiteit Hasselt uw eindverhandeling wereldwijd kan reproduceren, vertalen en distribueren is uw akkoord voor deze overeenkomst noodzakelijk. Gelieve de tijd te nemen om deze overeenkomst door te nemen, de gevraagde informatie in te vullen (en de overeenkomst te ondertekenen en af te geven).

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling met

Titel: Composable and Splittable User Interfaces for Distributed Device Federations
Richting: master in de informatica - Human Computer Interaction Jaar: 2008

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Ik ga akkoord,

ROBBEN, Neal

Datum: 20.11.2008

Composable and Splittable User Interfaces for Distributed Device Federations

Neal Robben

promotor :
Prof. dr. Kris LUYTEN,
Prof. dr. Karin CONINX

Universiteit Maastricht

Eindverhandeling voorgedragen tot het bekomen van de graad
Master in de informatica human-computer interaction

universiteit hasselt

Composable and Splittable User Interfaces for Distributed Device Federations

Neal Robben

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

Begeleiders: Jo Vermeulen & Geert Vanderhulst

Academiejaar: 2007-2008

Eindverhandeling voorgedragen tot het behalen van de graad van

Master in de Informatica
Afstudeervariant: Human-Computer Interaction

Universiteit Hasselt / Transnationale Universiteit Limburg

Abstract

Er is tegenwoordig al een aantal jaren een geweldige opmars van elektronische apparaten aan de gang. Er komen elke dag nieuwe types van apparaten bij in alle denkbare afmetingen en gewichten en de apparaten worden ook steeds krachtiger en veelzijdiger. Vooral de opkomst van mobiele apparaten met netwerkmogelijkheden is sterk tegenwoordig: PDA's, Smartphones, tablet Pc's, Ultra mobile Pc's' (UMPC),…

Dit alles maakt dat mensen vandaag meer dan ooit een ongelofelijke diversiteit aan apparaten met de verwerkingsmogelijkheden van een computer rondom zich hebben in het dagelijkse leven. Hiermee gaan we de richting uit van ubiquitous computing, waarbij het gebruik van de computer meer en meer in de dagelijkse activiteiten geïntegreerd wordt, tot op het punt dat de computer in de omgeving opgaat en mensen zich bij momenten zelfs niet meer bewust zijn van het feit dat ze een computer gebruiken.

Wanneer we zoveel apparaten in onze omgeving hebben kunnen we hierin ook de mogelijkheid zien om deze niet meer te beschouwen als een verzameling individuele apparaten om mee te werken, maar in plaats daarvan ze te laten samenwerken als één geheel om ons op een intelligente en gebruiksvriendelijke manier te ondersteunen in onze dagelijkse taken. Deze visie noemen we ambient intelligence.

Een dergelijke aanpak heeft ook gevolgen voor de user interface van applicaties waar we op deze manier mee werken. Traditionele user interfaces zijn er op voorzien om gebruikt te worden op één computer, die verbonden is met één set van IO-apparaten. Wanneer we meerdere computers in onze omgeving laten samenwerken als één geheel, biedt ons dit ook de mogelijkheid om te gaan werken met gedistribueerde gebruikersinterfaces. Door de user interface te verdelen over de beschikbare apparaten kunnen de unieke eigenschappen van elk apparaat gebruikt worden om het werken met de user interface zo optimaal mogelijk te laten verlopen voor de gebruiker.

Dat is precies waar deze thesis rond draait. We willen een systeem ontwikkelen dat het mogelijk maakt om op een gebruiksvriendelijke manier de user interface van een programma te verdelen over een heterogene verzameling apparaten. Hierbij moeten we een mechanisme zoeken om op een gebruiksvriendelijke manier te laten aangeven welk deel van de interface waar naartoe gedistribueerd moet worden, en de effectieve verdeling op een transparante manier te laten uitvoeren. Ook is het belangrijk dat we hierbij de state van de user interface consistent kunnen houden. Om de verdeling te kunnen realiseren is het van belang dat we beschikken over een meer abstracte voorstelling van de user interface, aangezien we delen van eenzelfde interface zullen moeten weergeven op apparaten die mogelijk zeer verschillend zijn. In deze thesis maken we daarvoor gebruik van de beschrijvingstaal UIML.

Voorwoord

Er zijn een aantal mensen die een rol gespeeld hebben bij het tot stand komen van deze thesis die ik bij deze zou willen bedanken.

In de eerste plaats mijn promotor Prof. dr. Kris Luyten en mijn co-promotor Prof. dr. Karin Coninx. Vooral de contactmomenten met Prof. Luyten hebben een aanzienlijke invloed gehad in een aantal van de designbeslissingen voor de implementatie die ik in het kader van deze thesis gerealiseerd heb.
Daarnaast wil ik ook mijn twee begeleiders bedanken: Jo Vermeulen en Geert Vanderhulst. Ze hebben mijn werk doorheen deze hele thesis opgevolgd en stonden altijd klaar om vragen te beantwoorden of wanneer er problemen waren.

Vervolgens is er nog mijn familie, die mij de kans heeft gegeven om mijn opleiding aan de universiteit af te maken.

Als laatste mag ik ook mijn medestudenten niet vergeten. Veel van de inspanning van de afgelopen twee jaar zijn opgegaan aan groepsprojecten waarin het resultaat volledig afhangt van de mensen waarmee samengewerkt wordt. Ik heb altijd fijne groepen gehad waarmee goed overlegd en samengewerkt kon worden.

Inhoud

Lijst van Figuren

Lijst van Listings

1 Inleiding

1.1 Probleembeschrijving

De grote opkomst van nieuwe types van apparaten, vooral in het mobiele segment, zorgt ervoor dat de gemiddelde persoon nog nooit eerder zoveel verschillende types apparaten rondom zich had doorheen een gemiddelde dag.

Computers in alle afmetingen en gewichten dringen stilaan door in elk aspect van ons dagelijks leven. Sommigen zijn nog altijd vrij opvallend, zoals bijvoorbeeld een traditionele desktop PC of een laptop. Anderen zijn dat al iets minder, zoals PDA's of smartphones. Sommigen zijn zelfs zo geïntegreerd in onze omgeving dat we ons er nauwelijks van bewust zijn dat we gebruik maken van een computer. Hoeveel mensen zouden zich tijdens het autorijden bijvoorbeeld bewust zijn van het feit dat er een boordcomputer in die auto zit die allerhande zaken regelt. Toch zijn dergelijke embedded systemen aanwezig in veel van de apparaten die we tegenwoordig rondom ons hebben.

Een volgende stap in de computer-evolutie is dan dat we deze apparaten die we rondom ons hebben uit hun isolement gaan halen, en laten samenwerken op een manier die het leven voor ons gemakkelijker of aangenamer maakt. Dit concept noemen we "ambient intelligence".

Veel van de apparaten die we rondom ons hebben zijn tegenwoordig al uitgerust met netwerkmogelijkheden. Vanuit een technisch standpunt is het dus in de meeste gevallen niet zo moeilijk om apparaten met elkaar te verbinden om ze te laten samenwerken. De gebruiksvriendelijkheid van een dergelijke set-up is echter minder vanzelfsprekend.

Op dit moment zijn we over het algemeen nog gewoon dat als we met een applicatie werken, dat de user interface hiervan draait op één apparaat, dat verbonden is met één set van input- en outputmechanismen. Wanneer we meerdere apparaten tegelijk gaan gebruiken om onze taken te vervullen krijgen we ook de mogelijkheid om de user interface van onze applicaties te gaan verdelen over meerdere apparaten. Zo kunnen we in principe elk apparaat dat we tot onze beschikking hebben gebruiken om het deel van de user interface te laten weergeven waar dit apparaat het beste voor geschikt is. Zo is bijvoorbeeld een PDA handig om via een beperkt aantal knoppen input te geven aan een programma, maar is het zeer omslachtig om hier grote hoeveelheden tekst op in te geven, en moeilijk om grote afbeeldingen of videobestanden te bekijken.

Wat we met deze thesis willen bereiken is onderzoek doen via literatuurstudie naar de achterliggende concepten die van belang zijn bij het creeëren van gedistribueerde gebruikersinterfaces en het opzetten van een systeem dat de gebruiker toelaat om op een gebruiksvriendelijke en transparante manier de user interface van een programma te verdelen over een aantal apparaten.

1.2 Voorbeeldscenario

De probleemstelling waar deze thesis rond draait is dus het verdelen van een user interface van een applicatie over meerdere apparaten op een simpele en gebruiksvriendelijke manier. Het gebruik van het systeem dat we in deze thesis hebben ontwikkeld om dit probleem op te lossen wordt hieronder geïllustreerd met een voorbeeldscenario.

Centraal in ons scenario staat natuurlijk onze gebruiker.

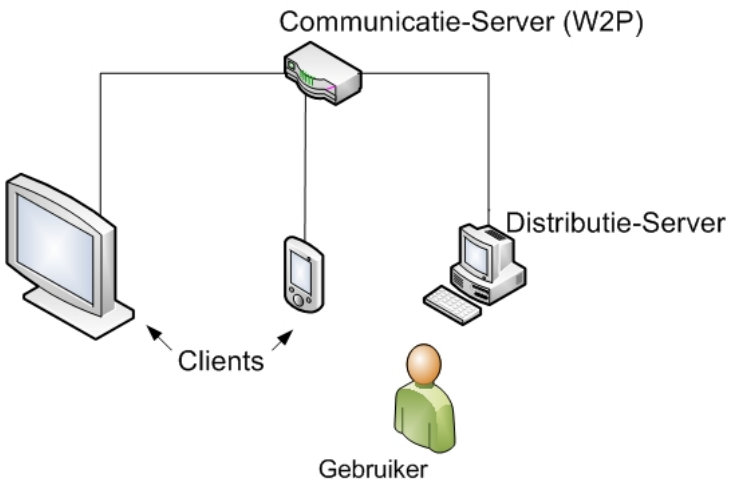
Figuur 1: De gebruiker uit ons voorbeeldscenario

Onze gebruiker heeft voor de gelegenheid enkele vrienden op bezoek. Tijdens het gesprek komt het onderwerp van zijn recente vakantie aan bod. Graag zou hij zijn vakantiefoto's tonen aan zijn vrienden. De foto's bevinden zich op zijn vaste computer.

Rondom de vaste computer is echter niet veel plaats en het zou handiger zijn als de foto's vanuit de zetel bekeken konden worden op een groot scherm.

Gelukkig beschikt hij over een PDA, een tv met set-top box, en de software die in deze thesis ontwikkeld werd voor het distribueren van een user interface.

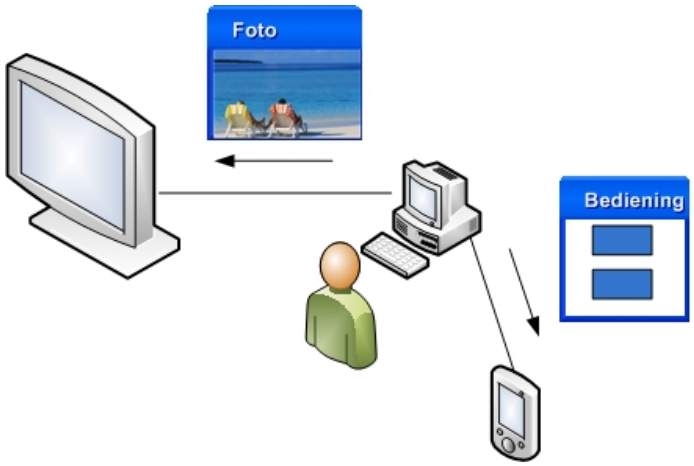
Hij begint met op zijn vaste pc de distributie-server op te starten en verbinding te maken met de communicatie-server. Vervolgens zet hij zijn PDA aan. Hij start de client-software en ook dit apparaat meldt hij aan bij de communicatie-server. Bij de set-top box van zijn televisie doet hij hetzelfde. Het resultaat is de hieronder afgebeelde setup.



Figuur 2: De apparaten binnen het systeem

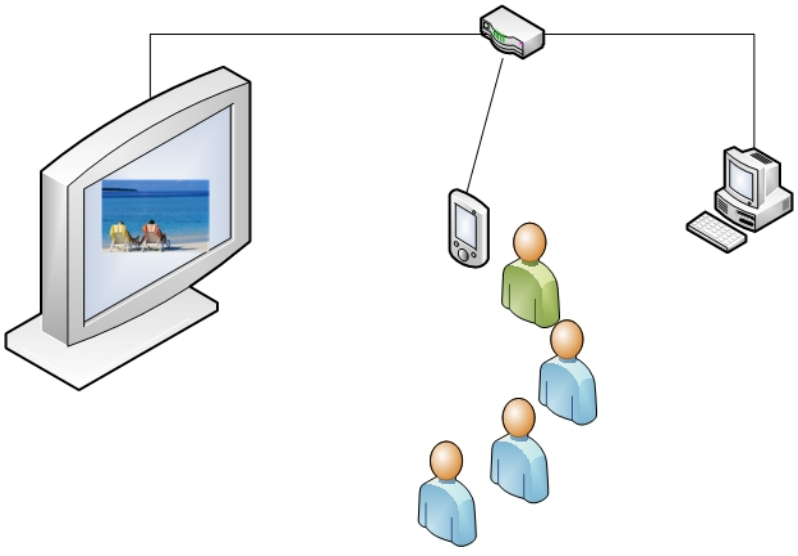
De distributie-server registreert de aanmelding van de PDA en televisie bij de communicatie-server. Vanaf nu zijn beide apparaten beschikbaar om een gedeelte van de interface te ontvangen van de distributie-server.

Onze gebruiker opent vervolgens in het server-venster van de distributie-server het programma waarvan hij de interface wil gaan verdelen. De distributie-server geeft hem nu de mogelijkheid om delen van de interface te selecteren en toe te wijzen. Hij selecteert de knoppenbalk waarmee hij het programma kan bedienen, en geeft aan dat deze naar zijn PDA gestuurd moet worden. Daarna selecteert hij het gedeelte van de interface waarin de foto getoond wordt en stuurt dit naar zijn tv.



Figuur 3: Het verdelen van de user interface

Nu de delen van de interface zijn toegewezen aan de gewenste apparaten, kan onze gebruiker op zijn gemak in de zetel gaan zitten bij zijn vrienden. Hij kan nu de applicatie om zijn foto's te bekijken bedienen vanop zijn PDA en de foto's worden in groot formaat getoond op zijn televisie.



Figuur 4: Het eindresultaat

1.3 Structuur van deze thesis

Deze thesis is als volgt opgevat: In de volgende twee hoofdstukken gaan we enkele van de belangrijkste concepten voorstellen die we nodig hebben om onze probleemstelling aan te pakken. Eerst komen migratie en distributie van user interfaces aan bod, daarna volgt een hoofdstuk omtrent het gebruik van beschrijvingstalen om een user interface voor te stellen. Dit hoofdstuk omvat ook een bespreking van de beschrijvingstaal die wij in deze thesis zullen gebruiken, nl. UIML.

Daarna volgt een hoofdstuk waarin we wat uitleg zullen geven over enkele reeds bestaande softwareprojecten waarvan we gebruik zullen maken bij de implementatie voor deze thesis.

Wat dan volgt is een hoofdstuk waarin een beschrijving wordt gegeven van de software die gerealiseerd werd in het kader van deze thesis. Als laatste volgt er een hoofdstuk met de conclusies omtrent hetgeen we in deze thesis bereikt hebben.

2 User Interface Migratie en Distributie

2.1 Inleiding

Het eerste wat we in deze thesis gaan doen is wat aandacht geven aan de achterliggende concepten die van belang zijn bij het ontwikkelen van een systeem voor distributie van een user interface, zoals we dat in ons voorbeeldscenario voor ogen hebben.
Distributie is hetgeen waar we vooral aan geïnteresseerd zijn, maar een dynamische distributie steunt op het concept “migratie”, zoals we verderop zullen zien. Daarom gaan we eerst kijken wat migratie juist inhoudt, welke vormen van migratie er bestaan en enkele voorbeelden aanhalen. Vervolgens zullen we zien wat distributie eigenlijk inhoudt, welke manieren er bestaan om dit aan te pakken en enkele voorbeelden van dergelijke systemen. Als laatste behandelen we kort de gebruiksvriendelijkheid van dergelijke systemen en zullen we iets zeggen over de mogelijkheden tot evaluatie.

2.2 User Interface Migratie

Wanneer een gebruiker bezig is met het uitvoeren van zijn taken kan het voorvallen dat voor verschillende taken binnen één programma, er ook verschillende platformen zijn waarop deze taken het best ondersteund kunnen worden. We willen echter niet dat de gebruiker het programma moet afsluiten, van apparaat wisselen, en daar het programma opstarten om dan opnieuw te beginnen. We willen dat de gebruiker tijdens zijn het uitvoeren van zijn taken kan wisselen van apparaat. Hiervoor moeten we migratie van de applicatie gaan ondersteunen. In principe hoeft niet de volledige applicatie gemigreerd te worden, maar enkel het interactieve deel [1], de user interface dus. De verwerking kan eventueel nog op het originele apparaat gebeuren, al zal duidelijk zijn dat dit enkel mogelijk is indien de apparaten met elkaar verbonden blijven. Wanneer we de mogelijkheid bieden om met mobiele apparaten zoals een PDA of een smartphone te werken, zal de volledige applicatie moeten migreren.

We kunnen migratie van een user interface definiëren als het overbrengen van de gehele user interface of een deel ervan, naar een andere set interaction resources [2]. Een *Interaction Resource* (IR) is een fysieke entiteit die de gebruiker in staat stelt de status van de applicatie te observeren en/of aan te passen. Bij het hanteren van deze definitie bestaat een apparaat dus uit een verzameling verschillende interaction resources, zoals knoppen en een scherm bij een PDA. Het touch-screen van diezelfde PDA is dan weer een voorbeeld van een interaction resource dat zowel dient voor observatie als input.
Het concept “interaction resource” komt ook nog terug later in dit hoofdstuk, in het deel over distributie.

Er zijn enkele dingen waar we rekening mee moeten houden bij het migreren van een user interface [3,4].

De verschillen tussen de platformen

Zoals eerder aangehaald zijn de karakteristieken van de platformen die zich in de omgeving van een gebruiker bevinden mogelijk zeer uiteenlopend. Deze diversiteit biedt op zich een

uitdaging omdat bijvoorbeeld een applicatie die ontworpen is voor een gewone pc meestal niet zomaar overgezet kan worden naar pakweg een PDA. Een PDA heeft een kleiner scherm, doorgaans minder verwerkingskracht, andere IO-mogelijkheden en een beperkte stroomvoorziening.
Dit alles wil zeggen dat wanneer we een applicatie willen migreren, we deze waarschijnlijk ook zullen moeten aanpassen aan het doelplatform.

De continuïteit van de interactie

Zoals gezegd willen we niet dat een gebruiker die van apparaat wisselt opnieuw moet beginnen. We willen dat hij gewoon zijn interactie kan hernemen op het punt waar hij gebleven was voor de migratie, en dan gewoon verder kan werken. Hiervoor moeten we dus de status van de applicatie gaan bijhouden.

Het voorgaande maakt dat we twee soorten data nodig hebben wanneer we aan migratie van user interfaces willen doen [3,4]:

- **Statische informatie**: over de platformen die deelnemen aan het migratieproces

- **Dynamische informatie**: over de run-time status van de applicatie die we willen migreren. De run-time status wordt gedefinieerd als de volledige geschiedenis van de interacties van de gebruiker met de applicatie en de resultaten hiervan [3].

Bij het omvormen van een user interface naar een vorm die geschikter is voor het doelplatform zullen we dan ook moeten opletten dat we rekening houden met de principes van de gebruiksvriendelijkheid. Bestaande transformaties voor applicaties nemen vaak hun toevlucht tot het verkleinen van visuele weergaven of het samenvatten van de weer te geven data [5]. Dit kan mogelijk de gebruiksvriendelijkheid van de user interface negatief beïnvloeden.

2.2.1 Vormen van migratie

Total migration

Bij deze vorm van migratie wordt de volledige user interface overgezet van één platform naar een ander. De gebruiker kan dus het apparaat veranderen dat hij gebruikt om te interageren met de applicatie.

Partial migration

Deze vorm van migratie is een stuk complexer. Hier wordt de interface opgesplitst in twee delen. Eén deel dient dan voor de interactie met de gebruiker en het andere deel dient dan voor het visualiseren van de outputinformatie. Het is de bedoeling dat één gedeelte op het bronapparaat blijft, terwijl het andere deel migreert naar een geschikter apparaat. Zo wordt de user interface verspreid over twee apparaten waarbij één het meest geschikt is voor interactie en het andere voor visualisatie.

Dit zou bijvoorbeeld gebruikt kunnen worden om een video te laten afspelen op een groot scherm, terwijl de controls hiervoor op een PDA gehouden worden. Dit kan een goede

oplossing zijn aangezien het soms moeilijk kan zijn om op een groot scherm input te geven. Wanneer een applicatie op een groot touch-screen gevisualiseerd wordt bijvoorbeeld is het misschien niet eens mogelijk voor de gebruiker om aan de menubalk te geraken.

Een moeilijkheid bij deze aanpak is natuurlijk uitmaken tot welke van de twee categorieën een bepaald UI-element behoort. Een eerste mogelijk stap hiervoor zou zijn te kijken naar het type van een bepaald widget. We kunnen dan widgets onderscheiden die input kunnen aanvaarden zoals een tekstvak en widgets die enkel output geven zoals een picturebox die een afbeelding weergeeft. Dit is uiteraard niet genoeg omdat het bijvoorbeeld mogelijk is dat een input-widget en een visualisatie-widget bij elkaar horen. Een simpel voorbeeld hiervan is een label dat voor een tekstvak staat.

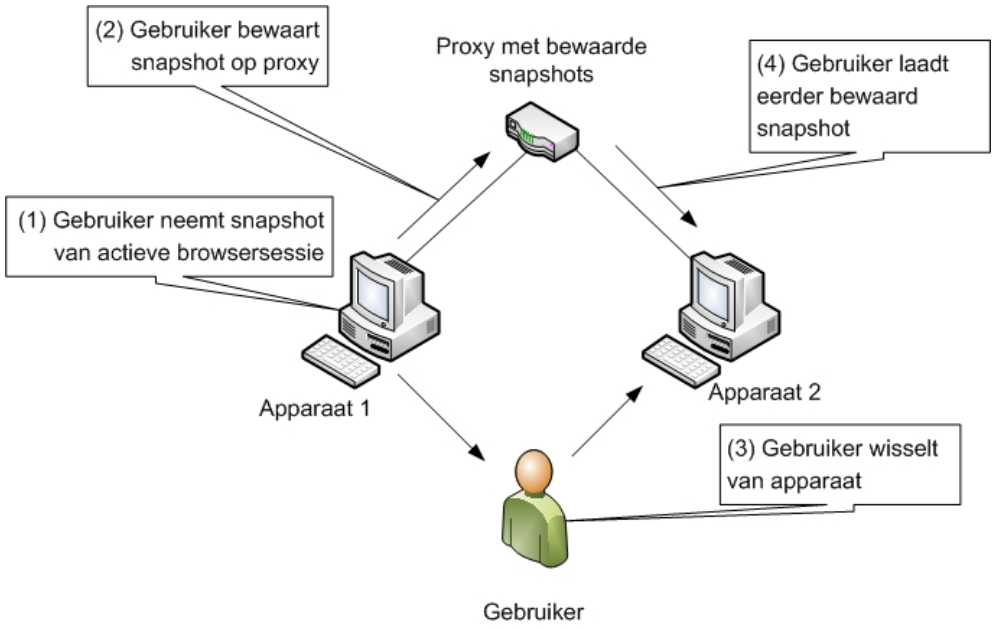
Om een verdergaande analyse te doen van de interface hebben we dan een model nodig dat de interface beschrijft op een hoger niveau. Dit model zou dan de relaties tussen de verschillende interface-elementen moeten bevatten via een aantal operators die relaties uitdrukken zoals hiërarchie, temporele relaties of groepering [3].

Mixed migration

Dit is een aanpak die nog verder gaat dan de gedeeltelijk migratie van daarnet. Bij deze vorm gaan we de interface opsplitsen in meerdere delen, die niet noodzakelijk een specifieke functie moeten hebben zoals interactie of visualisatie. De afzonderlijke delen kunnen een mengeling zijn van deze twee, en worden dan over twee of meer apparaten verspreid.
Hier kunnen we eigenlijk spreken van distributie van de user interface. Dit concept zal algemeen behandeld worden in het volgende deel.

2.2.2 Voorbeelden van migratiesystemen

Song, Chu & Kurakake ontwikkelden een systeem waarbij een sessie van een webbrowser gemigreerd kan worden naar een ander apparaat [6]. Dit wordt mogelijk gemaakt door gebruik van een proxy-server waar de gebruiker een snapshot van zijn actieve sessie kan opslaan.
Nadat hij van apparaat gewisseld heeft kan de gebruiker vervolgens gemakkelijk zijn sessie terug opnemen en verder werken.



Figuur 5: Werking van het systeem (Figuur geïnspireerd door [6])

Het systeem bestaat uit een plug-in en een proxy-server. Via de plugin kan de gebruiker zich aanmelden bij de proxy. Eens ingelogd kan de gebruiker met één druk op de knop een browsersessie opslaan, of een eerder bewaarde sessie verderzetten.
Het systeem kan uiteraard ook gebruikt worden zonder dat er van apparaat verwisseld moet worden. De gebruiker kan dan zijn actieve sessie bewaren om later verder te doen, of een aantal actieve websessies bijhouden.

Het nadeel van dit vrij simpele systeem is dat het geen rekening houdt met het eerste aandachtspunt dat we daarstraks hebben aangehaald, namelijk de diversiteit aan platformen waarmee gewerkt kan worden. Bij dit systeem wordt de sessie gewoon hernomen op het doelsysteem, zonder rekening te houden met de verschillen. Dit systeem is dus enkel bruikbaar wanneer bron- en doelplatform niet te fel van elkaar verschillen.

Een tweede voorbeeld, waarin wel expliciet rekening wordt gehouden met de diversiteit van de gebruikte platformen, wordt beschreven door Bandelloni & Paternò [4].
Het verschil met het vorige systeem is wel dat dit systeem niet zomaar met elke webapplicatie kan werken, maar extra informatie nodig heeft over de applicatie in de vorm van een aantal modellen van de applicatie op een hoger logisch niveau. Het systeem richt zich dan ook vooral op webapplicaties die gecreëerd werden op een model-gebaseerde wijze met behulp van TERESA[1] [7].

Centraal in dit systeem vinden we een migratie-server die de applicatie bijhoudt en de platformspecifieke interface hiervoor. Deze server handelt de binnenkomende migratie-requests af en stuurt de nodige informatie naar het doelplatform.

Wanneer een migratie-request binnenkomt gaat de server kijken welke pagina er actief is op het bronplatform. Dan gaat hij kijken welke pagina hiermee overeenkomt op het doelplatform. Omdat de eigenschappen van de gebruikte platformen mogelijk vrij ver uiteenlopen is er niet

[1] http://giove.isti.cnr.it/teresa.html

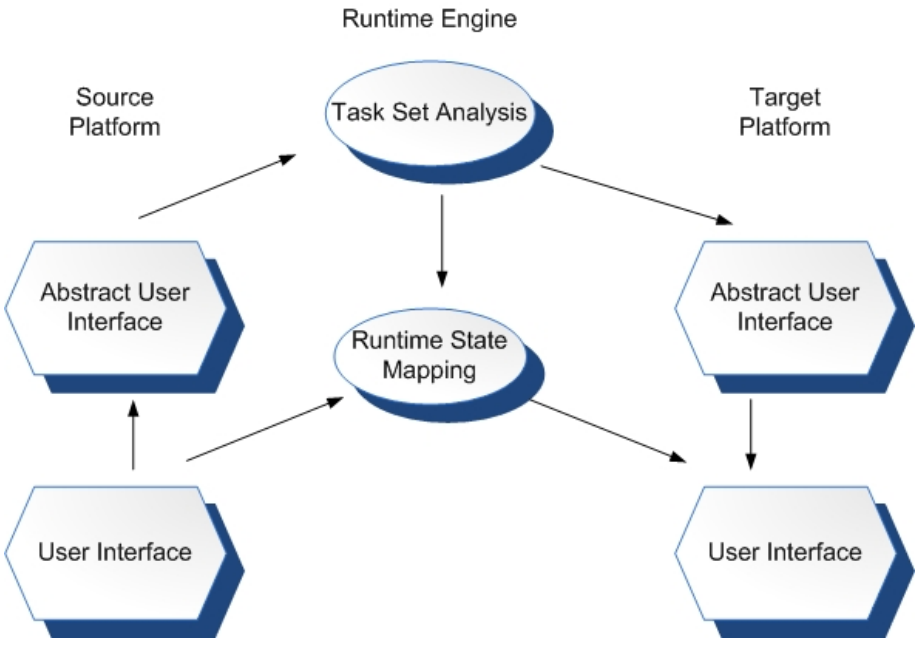
noodzakelijk een 1 op 1 mapping tussen de pagina's in de twee platformspecifieke versies van de interface. De server zal de meest gepaste pagina voor het doelplatform proberen te identificeren. Hiervoor zal de server gebruik maken van de modellen van de applicatie die hij heeft. Hij gaat kijken op niveau van het taakmodel welke logische taken ondersteund werden in de pagina op het bronplatform en gaat op basis hiervan kijken welke pagina uit de interface van het doelplatform hier het beste mee overeenkomt. Hiervoor gaat hij kijken naar het aantal logische taken dat overeenkomt. Hier zijn verschillende mogelijkheden:

- Er is één pagina in de interface voor het doelplatform die overeenkomt met de pagina uit het bronplatform. Deze pagina ondersteunt:
  - Evenveel taken
  - Minder taken
  - Meer taken

- Eén pagina uit het bronplatform komt overeen met meerdere pagina's uit het doelplatform

- Meerdere pagina's uit het bronplatform worden op het doelplatform samengebracht in één presentatie.

Indien er twijfel is tussen enkele pagina's op het doelplatform die hetzelfde aantal taken ondersteunen, dan wordt de pagina genomen die de taak ondersteunt waar de gebruiker het recentste mee bezig was op het bronplatform.
Nadat de correcte pagina op het doelplatform geïdentificeerd is krijgen de objecten op deze pagina de waarden toegewezen van de overeenkomstige objecten in de bronpagina. Indien er objecten in de doelpagina zitten die niet in de bronpagina zitten, krijgen deze hun default-waarde. Indien er objecten in de bronpagina zitten die niet in de doelpagina voorkomen, wordt deze informatie genegeerd.



Figuur 6: Presentation mapping tussen twee platformen. (Figuur geïnspireerd door [3])

2.3 User Interface Distributie

Wanneer we de fysieke resources in de omgeving van de gebruiker als één logisch geheel gebruiken om een taak uit te voeren spreken we van een *Interaction Space* (IS) [9]. Deze interaction space kan zowel bestaan uit apparaten van de gebruiker zelf, als een combinatie van individuele apparaten met publieke apparaten, zoals grote schermen in openbare gebouwen bijvoorbeeld.

Dit is precies wat we in deze thesis willen bereiken. We willen meerdere apparaten uit de omgeving van de gebruiker koppelen om zo de user interface op een zo gebruiksvriendelijk en efficiënt mogelijke manier weer te geven. Deze verzameling van apparaten noemen we dan een device federation. Deze federation noemen we homogeen of heterogeen al naargelang ze samengesteld is uit éénzelfde soort apparaten of niet. Ook kunnen we ze beschouwen als zijnde statisch of dynamisch, afhankelijk van het feit of de configuratie van de federation veranderd kan worden tijdens het runnen van het systeem [2].
In deze thesis wordt er gewerkt met een dynamische, heterogene federation.

Het logische geheel van de hardware en de services noemen we een distributed interaction space. Een interaction space is een combinatie van 3 elementen [2]: Een fysieke plaats, de interaction resources (IO-mogelijkheden) die daar aanwezig zijn, en de services die er aangeboden worden.

We kunnen twee soorten distributed interaction spaces onderscheiden [7]:

- **Personal interaction space**: Hier maakt slechts één persoon gebruik van het systeem om de gewenste taak tot een goed einde te brengen.

- **Collaborative interaction space**: Deze geeft de mogelijkheid aan meerdere personen om samen gebruik te maken van de interaction space, en laat hen toe een gezamenlijke taak uit te voeren. Deze vorm is wel complexer aangezien er eventueel locking-mechanismen nodig zijn voor bepaalde data zodat de state consistency niet in het gedrang komt wanneer meerdere mensen met dezelfde interface werken op hetzelfde moment.

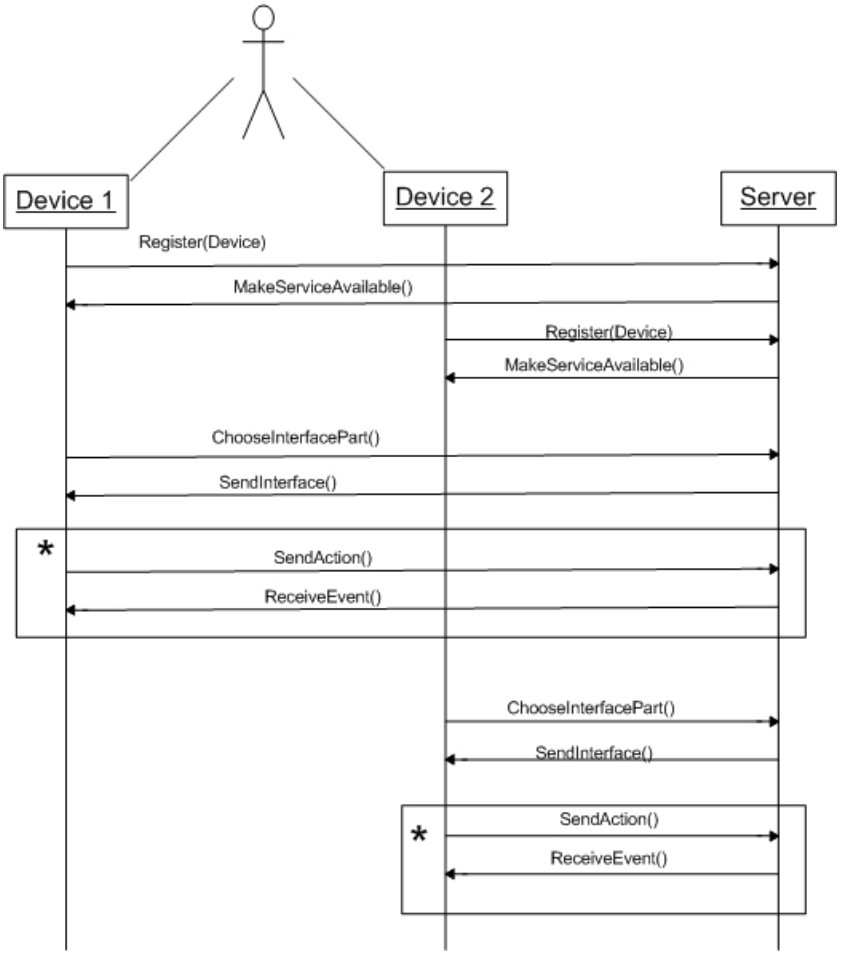
Misschien is het ook nuttig om even duidelijkheid te scheppen omtrent de term “gedistribueerde user interface”. We beschouwen een user interface als gedistribueerd wanneer de interface gebruik maakt van *Interaction Resources* (IR) die verspreid zijn over een verzameling van apparaten [2].

Distributie en de eerder behandelde term “migratie” zijn op zich twee onafhankelijke concepten. Een user interface kan op een vaste manier gedistribueerd zijn over een aantal apparaten zonder dat hier migratie aan te pas komt. Daar tegenover staat dat een user interface ook migreerbaar kan zijn tussen verschillende apparaten zonder dat deze verdeeld kan worden. Indien we werken aan een systeem dat een dynamische distributie ondersteund, zoals in deze thesis het geval is, steunt de distributie wel op het concept van user interface migratie.

2.3.1 Distributie-strategie

User-driven distribution

Bij deze vorm van distributie wordt het initiatief volledig aan de gebruiker overgelaten. De gebruiker moet hier zelf uitmaken welke apparaten er gebruikt moeten worden en welk deel van de interface deze apparaten krijgen toegewezen. Deze manier van werken heeft als voordeel dat de gebruiker de volledige controle behoudt over het proces. Dit kan wenselijk zijn omdat het vanwege de subjectieve aard van het begrip “optimaal” sowieso moeilijk is om een geautomatiseerde oplossing te ontwikkelen die een optimale verdeling geeft.



Figuur 7: User-driven distribution

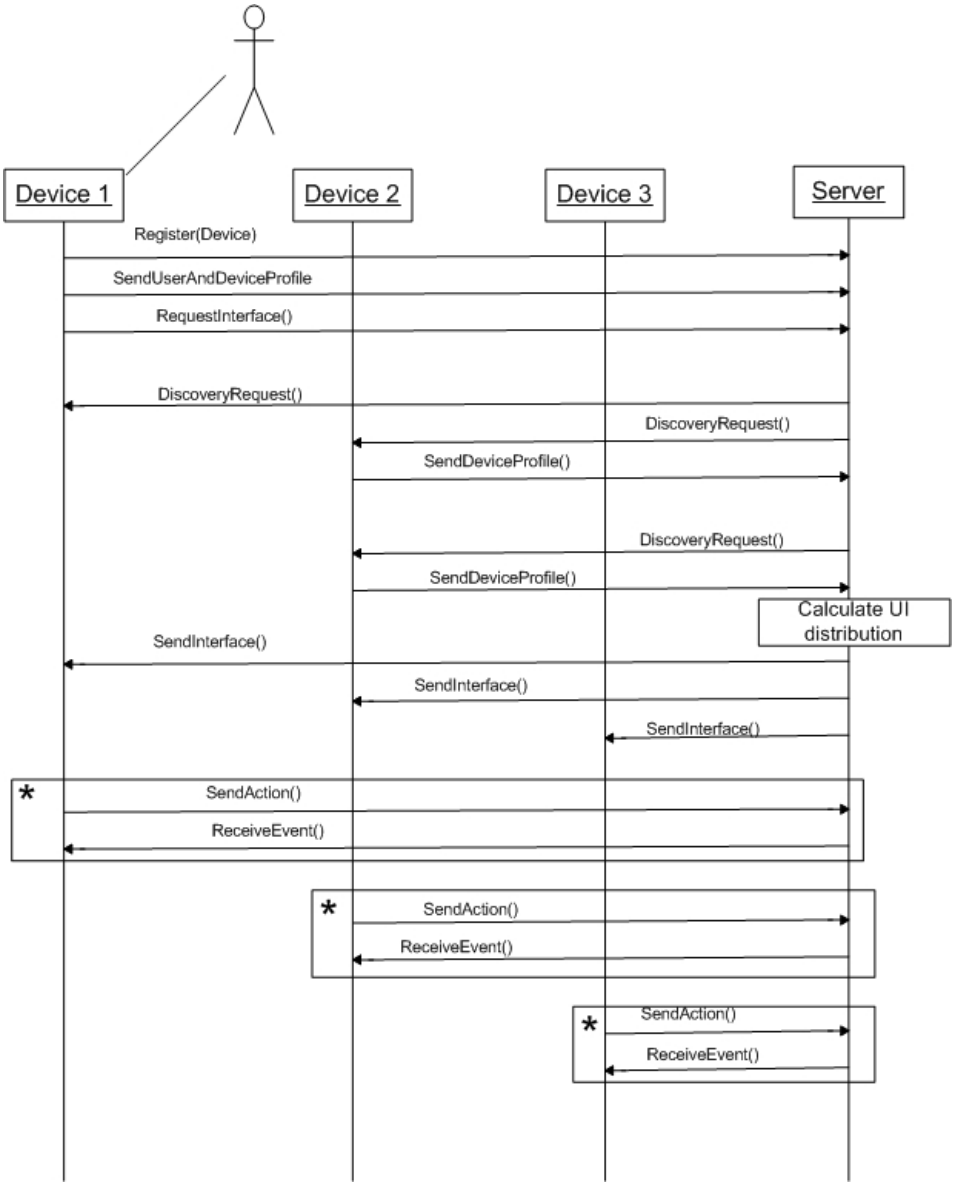
System-driven distribution

Bij een system-driven distribution bepaalt het systeem zelf welke apparaten geschikt zijn om mee te doen en wat de optimale verdeling is van de interface over deze apparaten. Het enige wat de gebruiker hoeft te doen is de interface voor zijn interaction space opvragen en daardoor het proces initiëren. Hierbij wordt eventueel een gebruikersprofiel meegestuurd waarin voorkeuren kunnen zitten waarmee het systeem rekening zal houden bij de verdeling.

Vervolgens gaat het systeem bepalen welke apparaten beschikbaar zijn in de omgeving en wat de karakteristieken ervan zijn.

Dit gebeurt door het broadcast te doen van een “discovery request”. Elk apparaat in de omgeving dat zo’n request ontvangt, gaat daarop een reply doen met het profiel van dat apparaat.

Eens de server weet wat de karakteristieken van de beschikbare apparaten zijn gaat op basis hiervan de toewijzing gebeuren van de onderdelen van de user interface aan de apparaten. Nadat alle delen van de interface zijn toegewezen volgt de effectieve migratie van de delen. Voor het berekenen van deze verdeling kan een kostenfunctie gebruikt worden [9,10]. Deze kostenfunctie kan bijvoorbeeld recursief het “gewicht” bepalen van een (gedeelte van) de interface en bepalen welk gewicht een apparaat kan dragen op basis van zijn karakteristieken. De interface zou dan enkel naar dat apparaat kunnen gedistribueerd worden indien het apparaat het gewicht kan dragen. De gewogen kost van de gedistribueerde configuratie zou dan geminimaliseerd kunnen worden om het “optimale” resultaat te geven, waarbij we nog eens herhalen dat optimaal een zeer subjectief begrip is. In de literatuur vinden we een aantal algoritmes voor partitionering en paginering van user interfaces die gebruik maken van eenzelfde soort functie [11].



Figuur 8: System-driven distribution

Continuous distribution

Bij deze aanpak wordt er rekening gehouden met het feit dat de omgeving van de gebruiker dynamisch kan veranderen terwijl er met het systeem gewerkt wordt. Deze veranderingen hebben normaal gezien twee mogelijke oorzaken [10]:

- Apparaten vallen weg
- Apparaten komen erbij

In het eerste geval valt er een apparaat weg waarop een deel van de interface beschikbaar werd gesteld. Om de toegang tot deze functionaliteit te behouden zal dit deel van de interface opnieuw toegewezen moeten worden aan een ander apparaat.

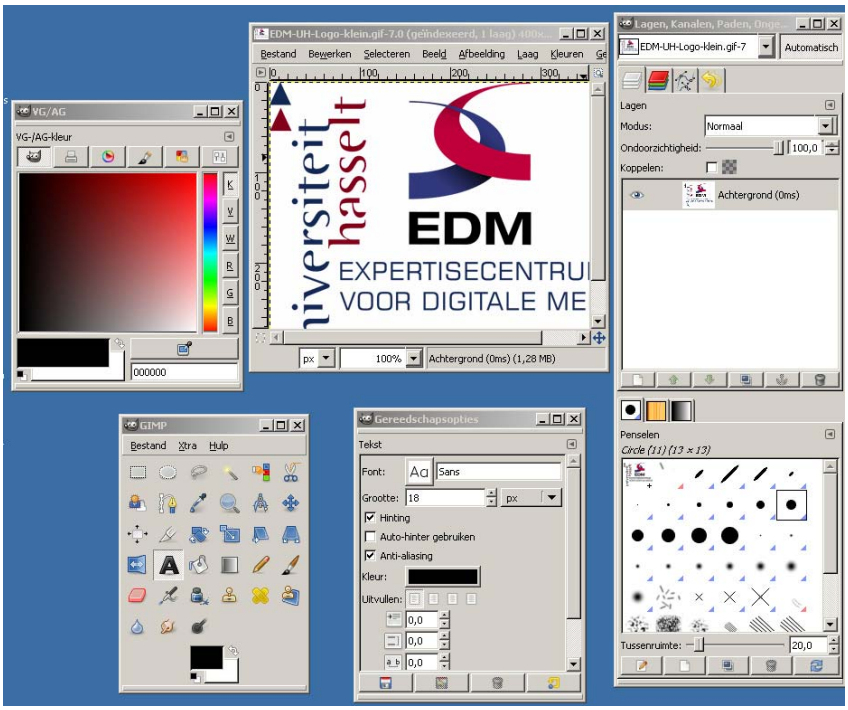
In het tweede geval is er niet noodzakelijk sprake van herdistributie. Wanneer een apparaat aangezet wordt zal het een boodschap uitsturen om zichzelf kenbaar te maken. Als deze boodschap wordt opgevangen door een server zal deze aan het apparaat zijn apparaatprofiel opvragen. Op basis hiervan kan de server gaan beslissen over eventuele herverdeling. Indien het bijkomende apparaat de mogelijkheid geeft om een betere verdeling van de user interface te bekomen, wordt er een herdistributie-stap uitgevoerd. Indien het apparaat niet kan helpen de huidige situatie te verbeteren wordt er geen actie ondernomen, maar houdt de server bij dat dit apparaat beschikbaar is voor het geval een ander apparaat moest wegvallen.

Deze manier van werken levert ook nieuwe problemen op. Zo verandert de verdeling van de interface terwijl de gebruiker ermee aan het werken is [10]. Dit kan zorgen voor verwarring en de gebruiker zelfs hinderen in zijn taak als er constant herverdelingen gebeuren als er frequent apparaten bijkomen en weggaan. Eventueel kan er gebruik gemaakt worden van de informatie uit het gebruikersprofiel om beperkingen op te leggen.

2.3.2 Voorbeelden van gedistribueerde user interfaces

De verzameling apparaten die de device federation vormen voor een distributed interaction space hoeven niet noodzakelijk zelf verwerkingsmogelijkheden te hebben. Zo hebben we op een traditionele desktop bijvoorbeeld ook de mogelijkheid om delen van een applicatie over verschillende locaties te verspreiden indien er meerdere schermen zijn aangesloten op de computer en de gebruikte applicatie toelaat om delen ervan op een willekeurige plaats op het scherm te zetten. Bij het beeld bewerkingsprogramma Gimp[2] kunnen we bijvoorbeeld de vensters met de tekengereedschappen en de kleurenpaletten naar een tweede monitor verplaatsen. Dit geeft ons dan de metafoor van een schilderspalet.

[2] http://www.gimp.org/



Figuur 9: Gebruik van meerdere vensters in GIMP

Vervolgens hebben we een voorbeeld waarin meerdere systemen geïntegreerd worden tot een interaction space die zowel individueel als voor samenwerking gebruikt kan worden [12]. In dit systeem worden de laptops van gebruikers aangevuld met een aantal vaste computers die in de omgeving geïntegreerd zijn en die een display surface maken van zowel de tafel als enkele muren van de werkomgeving. Bij aanwezigheid van meerdere deelnemers kunnen documenten op één van deze displays gesleept worden, waarna andere deelnemers er ook aan kunnen.


Figuur 10: Digitale documenten uitwisselen via de tafel (Figuur overgenomen uit [12])

Het kunnen distribueren van delen van een interface biedt ook mogelijkheden om mindervaliden meer mogelijkheden te geven om met computers om te gaan [13]. Bij iemand die motorisch gehandicapt is en daardoor geen normale invoer- en uitvoerapparaten kan

gebruiken, kan het gepaste deel van de user interface gemigreerd worden naar een geschikter apparaat. Een voorbeeld hiervan werd ontwikkeld in het Pebbles-project[3] [13], waarin men onderzoek doet naar de samenwerking tussen mobiele, handheld apparaten en gewone computers. Via de programma's Remote Commander en Shortcutter die in dit project ontwikkeld werden, kan een PDA als input device gebruikt worden voor een gewone computer. Ook niet-gehandicapte gebruikers kunnen hun PDA op deze manier gebruiken als extra inputapparaat, bovenop de traditionele combinatie muis/keyboard.

Figuur 11: Gebruik van een PDA als extra inputapparaat

2.3.3 Gebruiksvriendelijkheid & evaluatiemogelijkheden

De traditionele user interfaces zoals wij die kennen zijn doorgaans gericht op het werken met een set-up die slechts beschikt over één of twee rechtstreeks aangesloten schermen, en beschikt over één set invoerapparaten [14]. Wanneer we deze interfaces gaan distribueren over meerdere fysieke apparaten krijgen we niet alleen meer mogelijkheden in het uitvoeren van onze taken, maar ook meer mogelijkheden tot problemen. Enkele usability "issues" die naar voren gebracht worden in verband met gedistribueerde user interfaces zijn onder andere: moeilijkheden met periferisch zicht, verhoogde cognitieve belasting, moeilijkheden bij het configureren van het systeem [15].

Dit lijkt ook te impliceren dat er voor een effectieve evaluatie van dit soort user interfaces niet in alle gevallen gewerkt kan worden met de traditionele evaluatietechnieken zoals we die hebben leren toepassen voor de meer klassieke types van user interfaces.

Wanneer we de algemene gebruiksvriendelijkheid willen meten met bijvoorbeeld testen die na gebruik van het systeem een subjectieve mening van de gebruiker vragen, of dingen meten zoals de benodigde tijd en foutenlast bij het uitvoeren van taken, is het geen enkel probleem om gebruik te maken van traditionele technieken. Willen we echter specifieke handelingen gaan beoordelen, het systeem evalueren op basis van een aantal metrics of op basis van de kenmerken van het systeem een vergelijking maken met soortgelijke systemen, dan kan het gepast zijn om meer specifieke methodes te hanteren. We zullen nu uit de literatuur kort enkele metrics aanbrengen die gehanteerd kunnen worden bij het evalueren van

[3] http://www.pebbles.hcii.cmu.edu/index.php

gedistribueerde user interfaces en twee methodes die gebruikt kunnen worden bij het bespreken en vergelijken van dergelijke systemen. De beste manier om een gebruiksvriendelijk eindproduct te garanderen blijft nog altijd veel testen uitvoeren met eindgebruikers onder realistische omstandigheden.

2.3.3.1 Metrics

Luyten & Coninx halen enkele mogelijke metrics aan die we kunnen gebruiken voor het evalueren van een distributed interaction space [9].

Completeness

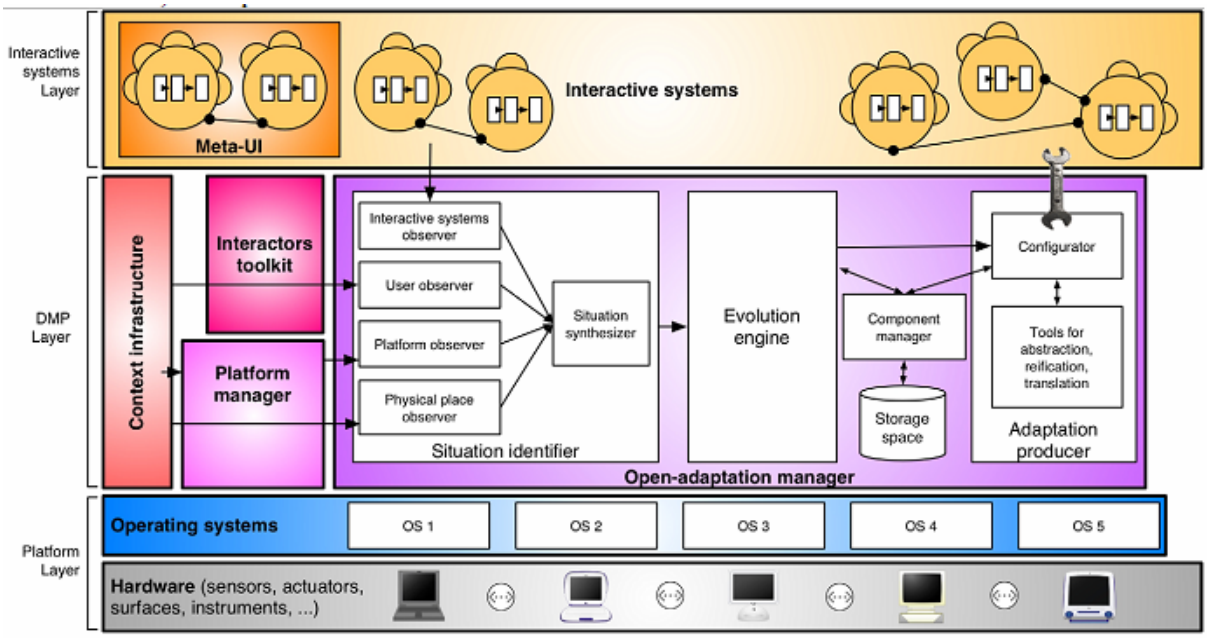
Deze metric kijkt of alle user interface-elementen die nodig zijn om een bepaalde taak uit te voeren beschikbaar zijn op een bepaald moment, ongeacht de apparaten die beschikbaar zijn om de interface weer te geven. Voor dynamisch gedistribueerde systemen is het in de praktijk onmogelijk om onder alle omstandigheden aan deze metric te voldoen aangezien op elk moment apparaten kunnen bijkomen of wegvallen. Om toch voorbereid te zijn kunnen eventueel meerdere alternatieven voorzien worden voor onderdelen uit de interface. Voor een onderdeel dat een bepaalde modaliteit gebruikt kunnen bijvoorbeeld alternatieven in andere modaliteiten voorzien worden. Wanneer dit niet het geval is kan een deel van de interface misschien niet weergegeven worden bij gebrek aan IO-ondersteuning voor een bepaalde modaliteit.

Continuity

Continuity wil zeggen dat de gebruiker een duidelijk beeld houdt van de toestand van het systeem, ondanks het feit dat er meerdere IO-apparaten gebruikt worden en het feit dat de verdeling eventueel kan veranderen tijdens het werken. Om hier zo goed mogelijk aan te voldoen kan het systeem een aantal regels opstellen die in acht moeten genomen worden bij herdistributie. Een voorbeeld hiervan is het bestaan van een sub-taak die een onderdeel is van twee verschillende taken [9]. We kunnen dan zeggen dat wanneer een overgang gemaakt wordt tussen de twee taken, de gemeenschappelijke sub-taak best niet hergedistribueerd wordt.

2.3.3.2 CAMELEON Run-Time

Indien we een systeem dat we ontwikkeld hebben willen evalueren tegenover andere systemen kunnen we gebruik maken van CAMELEON-RT [2]. Dit is een conceptueel referentiemodel in 3 lagen voor de architectuur van een systeem dat werkt met distributie en migratie van user interfaces. Het is een high-level ontleding van de functionele componenten die in een dergelijk systeem aanwezig zouden moeten zijn. Het geheel is echter redelijk uitgebreid wat maakt dat voor veel systemen een deel van de componenten die erin gedefinieerd zijn niet van toepassing zal zijn.



Figuur 12: Het CAMELEON-RT referentiemodel (Figuur overgenomen uit [2])

2.3.3.3 4C Referentiemodel

Een ander referentiemodel dat door enkele van dezelfde auteurs voorgesteld wordt om gedistribueerde user interfaces te bespreken is het 4C referentiemodel [15]. In dit model wordt de interface opgedeeld in 4 “C”-dimensies:

- Computation: Welke elementen worden er gedistribueerd?
- Communication: Wanneer worden deze elementen gedistribueerd?
- Coordination: Wie is verantwoordelijk voor de distributie van het systeem?
- Configuration: Waar worden de delen naartoe gedistribueerd?

Een laatste mogelijkheid die ook nog vermeld wordt door Luyten & Coninx [9], is het gebruik van de CARE properties [16]. Dit is een manier die gebruikt wordt om de gebruiksvriendelijkheid van multi-modale systemen te beoordelen. Deze werkwijze is dus enkel nuttig wanneer er meerdere modaliteiten gebruikt worden.

2.4 Conclusie

In dit hoofdstuk zijn we iets dieper ingegaan op twee concepten die zeer belangrijk zijn voor deze thesis: migratie en distributie van user interfaces. In deze thesis is het de bedoeling om te werken met dynamische distributie van user interfaces, en om dat te verwezenlijken hebben we ook migratie nodig. Dit hoofdstuk heeft achtergrond gegeven over wat deze twee concepten juist inhouden en de werking ervan. In het volgende hoofdstuk gaan we bekijken hoe we meer abstractie kunnen bekomen bij het voorstellen van een user interface. Deze abstractie zal ons helpen met het aanpakken van variaties in de gebruikte toolkits bij het weergeven van de interface op de verschillende apparaten uit onze device federation.

3 User Interface Beschrijvingstalen en User Interface Markup Language

3.1 Inleiding

In een situatie zoals degene uit ons voorbeeldscenario wordt een user interface verdeeld over een verzameling apparaten, die mogelijk zeer divers is en waarin de apparaten kunnen verschillen van softwareplatform. Hierdoor is het noodzakelijk dat we een meer abstracte voorstelling hebben van de user interface die we willen distribueren. Hiervoor kunnen we gebruik maken van een interface beschrijvingstaal. In dit hoofdstuk zullen we eerst algemeen bekijken wat beschrijvingstalen zijn en welke voordelen deze hebben. Daarna zullen we een toelichting geven van UIML, de beschrijvingstaal waar in deze thesis gebruik van gemaakt wordt, waarbij we de basiselementen van een interface beschrijving in deze taal toelichten.

3.2 Interface beschrijvingstalen

Wanneer we een groot aantal verschillende apparaten rondom ons hebben willen we ook graag de mogelijkheid hebben om toegang te hebben tot onze persoonlijke data en hiermee te werken vanop verschillende types van apparaten. De interactietechnieken die gebruikt worden bij deze heterogene verzameling apparaten verschillen mogelijk radicaal en ook de verwerkingsmogelijkheden, en andere kenmerken zoals bijvoorbeeld schermgrootte, kunnen ver uiteenlopen.

Het is dan ook geen kleine uitdaging voor ontwikkelaars om te zorgen dat dezelfde taken op een effectieve en gebruiksvriendelijke manier ondersteund worden op een dergelijk groot aantal verschillende platformen, waarbij we een platform kunnen definiëren als een combinatie van een apparaat, een besturingssysteem en een toolkit.

Een voor de hand liggend probleem hierbij is de inspanning die vereist is bij het ontwikkelen van de user interfaces voor al deze apparaten. Het is quasi ondoenbaar geworden om voor elk type apparaat dat uitkomt de interface opnieuw te gaan implementeren voor dat specifieke platform.
Het geheel wordt nog eens bemoeilijkt door het feit dat de evolutie van een user interface zelden stopt na de initiële ontwikkeling. Wanneer de functionaliteit van de applicatie evolueert zal ook de interface moeten volgen. Wanneer dit op elk type apparaat apart moet gebeuren is de nodige inspanning al gauw niet meer te overzien.

Ook moet er vanuit het standpunt van gebruiksvriendelijkheid rekening gehouden worden met het feit dat interfaces die dezelfde functionaliteit aanbieden op verschillende apparaten toch enigszins consistent moeten zijn.

3.2.1 High-level User Interface Description Language

Als antwoord op deze uitdagingen is er al een aantal jaren een opkomst aan de gang van een aanpak waarbij men de interface op een hoger niveau van abstractie probeert te beschrijven dan dat van de implementatie.
De essentie van de interface gaat men dan proberen uit te drukken in een bepaalde beschrijvingstaal, een *High-level User Interface Description Language* (HLUIDL). De bedoeling is een interface te ontwikkelen op een platformonafhankelijke manier om deze daarna met een minimum aan inspanning te kunnen overzetten naar een brede waaier van apparaten. Deze aanpak volgt eigenlijk de algemene benadering van modelgebaseerde ontwikkeling waarbij er een model op hoog niveau wordt aangemaakt dat dan (deels) automatisch wordt omgezet naar het gewenste resultaat. Bij de meeste beschrijvingstalen voor user interfaces werkt men echter slechts één niveau boven de concrete user interface, daar waar men bij andere beschrijvingstalen of andere soorten modelgebaseerde ontwikkeling courant op een nog hoger niveau werkt, zoals bijvoorbeeld dat van een taakmodel [17,18,7]. Eigenlijk kunnen we een beschrijving die in een dergelijke taal is opgesteld zien als een opsomming van *Abstracte Interactie Objecten* (AIO) die daarna wordt vertaald naar *Concrete Interactie Objecten* (CIO). Deze (deels) geautomatiseerde werkwijze kan een hoop tijd en werk uitsparen doordat er geen dubbele inspanning meer moet gebeuren.

Een abstracte user interface-beschrijving is dan ook precies wat we in deze thesis nodig hebben. Het is de bedoeling om een interface voor een programma één keer te schrijven, en dan delen hiervan te kunnen distribueren naar andere apparaten, die niet noodzakelijk gebruik maken van hetzelfde platform zoals een laptop of een pda. De apparaten die een beschrijving van een deel van de interface krijgen om weer te geven, zullen deze dan at runtime omzetten in een concrete interface om weer te geven.

Er bestaan reeds een groot aantal dergelijke initiatieven. De meeste van deze beschrijvingstalen werken met een syntax die gebaseerd is op de *eXtensible Markup Language* (XML[4]).
Het gebruik van XML-gebaseerde talen biedt een aantal voordelen.

- Het gebruik van XML is licentievrij.
- De standaard is goed gekend en heeft een zeer brede ondersteuning.
- XML is platformonafhankelijk aangezien deze gebruikt kan worden op elk platform waarvoor een XML-parser bestaat, of waarvoor men een parser kan schrijven.
- De syntax van XML is vrij simpel, wat de standaard ook geschikt maakt voor gebruik door mensen die minder technisch aangelegd zijn.
- Delen van een XML-bestand kunnen gemakkelijk herbruikt worden.
- Talen die gebaseerd zijn op XML zijn gemakkelijk uit te breiden.
- Door gebruik te maken van *Extensible Stylesheet Language Transformations* (XSLT[5]) is elk XML-document gemakkelijk om te vormen naar een ander type document.
- De correctheid van beschrijvingen in XML-gebaseerde talen kan gemakkelijk worden afgedwongen door gebruik te maken van schema's zoals *Document Type Definitions* (DTD[6]), XML Schema's[7], RelaxNG schema's[8],…

[4] http://www.w3.org/XML/
[5] http://www.w3.org/Style/XSL/
[6] http://www.w3schools.com/dtd/default.asp
[7] http://www.w3.org/XML/Schema

We zouden het gebruik van HTML voor het opstellen van webpagina's kunnen aanzien als een voorbeeld van het gebruik van een user interface beschrijvingstaal voor het browserplatform. Er wordt een brondocument opgesteld dat de webpagina weergeeft. De specifieke weergave van de onderdelen van deze pagina op de client-machine hangt uiteindelijk echter af van de browser die de pagina interpreteert en weergeeft. Zo zal een knop op een webformulier een iets ander uitzicht hebben in een browser die draait op Macintosh dan op Windows.

Het gebruik van een taal om de user interface op een hoger, abstract niveau te beschrijven heeft buiten het verminderen van de benodigde inspanning voor ontwikkeling en onderhoud, nog een aantal voordelen [19].

- Het biedt de mogelijkheid om de interface te laten ontwikkelen door mensen die slechts zeer weinig, of zelfs helemaal geen technische kennis hebben van de omgeving waarin deze user interface uiteindelijk zal gerealiseerd worden. Zo kan een grafische designer bijvoorbeeld ingezet worden om een interface te ontwikkelen voor een embedded systeem zonder dat hij iets moet afweten van de ingewikkelde technische details van deze klasse van systemen.

- Deze werkwijze biedt ook een betere ondersteuning voor de toepassing van rapid prototyping. Dit zorgt voor een snellere ontwikkelingscyclus waarbij wijzigingen sneller gerealiseerd kunnen worden. In een markt met veel concurrentie kan dit een grote hulp zijn om gebruikerstesten vlotter te laten verlopen.

- Een abstracte beschrijving van de te realiseren user interface kan de communicatie tussen de verschillende stakeholders bevorderen tijdens de ontwikkeling.

- Wanneer een interface wordt opgesteld op een abstract in plaats van een concreet niveau biedt dit mogelijkheden om de user interface aan te passen per gebruiker of per gebruikersgroep [20]. Men kan bijvoorbeeld bepaalde delen van een user interface weglaten voor bepaalde types gebruikers [19].

Ondanks alle voordelen die deze aanpak biedt is het echter vaak toch niet helemaal mogelijk om slechts één beschrijving van een user interface te kunnen gebruiken voor alle mogelijke apparaten. Er zullen altijd extreme omstandigheden zijn die een aparte aanpassing vergen zoals apparaten met zeer kleine schermen, of juist meerdere schermen, of bijvoorbeeld bij het gebruik van een semafoon (ook wel "pager" genoemd) die slechts beschikt over enkele lijnen tekstweergave.

Er bestaan op het gebied van beschrijvingstalen verschillende initiatieven zoals UIML[9], AUIML[10], XIML[11], Seescoa XML[12], Teresa XML[13], en nog een aantal anderen. Voor een uitgebreider overzicht en een vergelijking van de mogelijkheden die deze verschillende talen bieden verwijzen we naar [21]. Alle genoemde initiatieven hebben hun sterktes en zwaktes en

[8] http://relaxng.org/
[9] http://www.uiml.org
[10] http://www.alphaworks.ibm.com/tech/auiml
[11] http://www.ximl.org/
[12] http://www.cs.kuleuven.ac.be/cwis/research/distrinet/projects/SEESCOA/
[13] http://giove.isti.cnr.it/teresa/teresa_xml.html

de keuze voor een bepaalde taal zal dan ook vaak afhangen van de specifieke situatie waarvoor men ze nodig heeft

In deze thesis maken we gebruik van UIML. Deze meta-taal is ontwikkeld voor multi-platform gebruik en ondersteunt een duidelijke “separation of concerns”. Ook is er voor het gebruik van deze taal reeds software beschikbaar die ontwikkeld werd op het EDM zelf: UIML.Net (zie verder). De broncode van deze renderer is vrij beschikbaar en kan dus voor het gebruik aangepast worden waar nodig. Dit is een zeer groot pluspunt voor het gebruik van UIML in het kader van het probleem uit deze thesis. We hebben sowieso een goede renderer nodig, welke taal we ook zouden gebruiken, maar bij het distribueren van user interfaces is de kans zeer groot dat we de gebruikte renderer zullen moeten aanpassen.

3.3 User Interface Markup Language

Eén van de vele standaarden die er bestaan voor user interface beschrijvingstalen is de *User Interface Markup Language* (UIML). Zoals de meeste beschrijvingstalen is ook UIML gebaseerd op XML. Bij het gebruik van UIML is men dus niet afhankelijk van de beschikbaarheid van tools, maar kan het opstellen van een interface-beschrijving eventueel gewoon met de hand gebeuren.

UIML is bedoeld om volledig platformonafhankelijk te zijn en kan door iedereen vrij gebruikt worden voor het maken van interfaces die gebruik maken van verschillende modaliteiten, daar waar andere initiatieven voor beschrijvingstalen soms specifiek gericht zijn op grafische user interfaces.

De meta-taal UIML is ondertussen tien jaar oud, en wordt beschouwd als één van de meer volwassen initiatieven op het gebied van user interface beschrijvingstalen. Er zijn een aantal projecten die zich bezig houden met het ontwikkelen van UIML-renderers voor gebruik op verschillende platformen zoals bijvoorbeeld UIML.Net[14] [22,23] dat op het EDM zelf ontwikkeld wordt (zie ook verder) voor het .Net platform.
Ook zijn er renderers ontwikkeld voor Java, HTML, VoiceXML, WML,…
Voor een volledige lijst verwijzen we graag naar de officiële website[15].

Op dit moment wordt de UIML-specificatie beheerd door OASIS[16] en is men bezig om van UIML een officiële standaard te maken.

Hier volgt een korte beschrijving van de structuur van een UIML-document, op basis van de UIML 3.0 specificatie [24]. Wat volgt is slechts een korte bespreking van een aantal basiselementen. Voor een meer gedetailleerde uitleg verwijzen we naar de specificatie, en voor meer achtergrond rond de beslissingen bij het ontwerp van de taal wordt er verwezen naar [25].

[14] http://research.edm.uhasselt.be/uiml
[15] http://www.uiml.org/
[16] http://www.oasis-open.org/committees/uiml

3.3.1 Het document

Een interface wordt beschreven in een UIML-document dat is gestructureerd in een aantal delen.

```
<uiml>
<head>…</head>
<interface>
        <structure>…</structure>
        <style>…</style>
        <content>…</content>
        <behavior>…</behavior>
</interface>
<peers>
        <presentation>…</presentation>
        <logic>…</logic>
</peers>
<template>…</template>
</uiml>
```

Listing 1: Structuur van een UIML-document

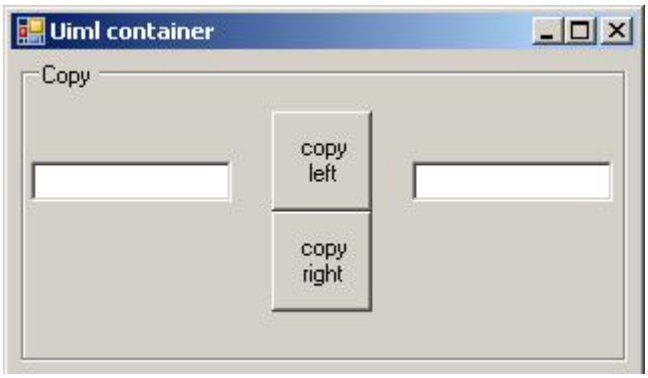
Op het hoogste niveau zijn er vier elementen, nl: “Head”, “Interface”, “Peers” en “Template”.

Het “head”-element bevat meta-informatie over het UIML-document zelf zoals bijvoorbeeld de auteur. Deze sectie is vooral handig voor tools die UIML-documenten produceren [26]. Het “interface”-element beschrijft de interface zelf, in vier verschillende aspecten. Het “peers”-element dient om aan te geven hoe de mapping moet gebeuren van de abstracte elementen uit de UIML-beschrijving naar concrete widgets in een specifieke toolkit, en hoe externe programmalogica kan worden aangesproken vanuit de UIML-interface. Via het “template”-element tenslotte kunnen we herbruikbare stukken UIML definiëren, zoals bijvoorbeeld vaak voorkomende onderdelen van een user interface.

Hieronder volgt een listing met een voorbeeld van een UIML-document, en daarna een screenshot van het gerenderde resultaat.

```
<?xml version="1.0"?>
<uiml>
	<interface>
		<structure>
			<part class="Frame" id="Frame">
				<part class="Entry" id="leftentry"/>
				<part class="Button" id="copyleft"/>
				<part class="Button" id="copyright"/>
				<part class="Entry" id="rightentry"/>
			</part>
		</structure>
		<style>
			<property part-name="Frame" name="position">5,5</property>
			<property part-name="Frame" name="size">302,150</property>
			<property part-name="leftentry" name="position">5,50</property>
			<property part-name="leftentry" name="size">100,25</property>
			<property part-name="rightentry" name="position">195,50</property>
			<property part-name="rightentry" name="size">100,25</property>
			<property part-name="copyleft" name="position">125,25</property>
			<property part-name="copyleft" name="size">50,50</property>
			<property part-name="copyright" name="position">125,75</property>
			<property part-name="copyright" name="size">50,50</property>
			<property part-name="Frame" name="label">Copy</property>
			<property part-name="copyleft" name="label">copy left</property>
			<property part-name="copyright" name="label">copy right</property>
			<property part-name="leftentry" name="text"></property>
			<property part-name="rightentry" name="text"></property>
		</style>
		<behavior>
			<rule>
				<condition>
					<event class="ButtonPressed" part-name="copyleft"/>
				</condition>
				<action>
					<property part-name="rightentry" name="text">
						<property part-name="leftentry" name="text"/>
					</property>
				</action>
			</rule>
			<rule>
				<condition>
					<event class="ButtonPressed" part-name="copyright"/>
				</condition>
				<action>
					<property part-name="leftentry" name="text">
						<property part-name="rightentry" name="text"/>
					</property>
				</action>
			</rule>
		</behavior>
	</interface>
	<peers>
		<presentation base="swf-1.1.uiml"/>
	</peers>
</uiml>
```

Listing 2: Voorbeeld van een UIML-document



Figuur 13: Gerenderd resultaat van het voorgaande document

We zullen nu de twee belangrijkste elementen van een UIML-document en hun sub-elementen bespreken: “interface” en “peers”.

3.3.1.1 Interface

Het “interface”-element kan eigenlijk beschouwd worden als het meest essentiële element binnen een UIML-document. Alles wat de eigenlijke interface beschrijft zit hierin vervat.

```
<interface>
        <structure>…</structure>
        <style>…</style>
        <content>…</content>
        <behavior>…</behavior>
</interface>
```

Listing 3: De inhoud van het “interface”-element

De beschrijving van de interface zelf wordt opgedeeld in vier verschillende aspecten: Structure, Style, Content en Behaviour.

3.3.1.2 Structure

Het “structure”-element definieert welke onderdelen er in de user interface voorkomen, en wat de onderlinge hiërarchie is aangezien onderdelen in elkaar genest kunnen worden.

Elk onderdeel in UI wordt vertegenwoordigd door een “part”-element. Dit element heeft steeds een unieke naam en een klasse die aangeeft wat voor abstract UI-element dit part voorstelt. De klasse-naam wordt bij het renderen gebruikt om in de vocabulary op te zoeken welk concreet widget er gebruikt moet worden.

```
<structure>
        <part id=”top” class=”Container”>
                <part id=”lbl_name” class=”Label”>
                <part id=”txt_name” class=”Entry”>
        </part>
</structure>
```

Listing 4: Voorbeeld van het gebruik van het “structure”-element

3.3.1.3 Style

Het “style”-element definieert een aantal eigenschappen van de parts in het structure-element zoals bijvoorbeeld positie, afmetingen,…

Dit gebeurt in de vorm van een reeks “property”-elementen die telkens de naam van het bewuste part aangeven, de naam van de gewenste property, en een waarde. Het is ook mogelijk om een property in te stellen op klasse-niveau voor bijvoorbeeld alle buttons in plaats van voor een individueel part.

Deze property-elementen kunnen ook genest zijn zodat bepaalde properties als waarde de inhoud van een andere property toegewezen kunnen krijgen of ze kunnen een referentie bevatten naar een waarde die gedefinieerd is in het content-gedeelte.

```
<style>
        <property  part-name=”SubmitButton”  name=”label” >Submit!</property>
        <property  part-class=”Label”  name=”font” >Times New Roman </property>
        <property  part-name=”RightEntry”  name=”label” > <property part-
                name=”LeftEntry”/></property>
        <property  part-name=”TitleLabel”><reference constant-name=”titel” /></property>
</style>
```

Listing 5: Voorbeeld van de mogelijkheden van de “style”-sectie

Tenslotte is het ook mogelijk om meerdere style-secties te definiëren in eenzelfde UIML-document die kunnen dienen om de interface in meerdere contexten bruikbaar te maken, zoals bijvoorbeeld bij gebruik door iemand die kleurenblind is.

3.3.1.4 Content

Het “content”-element dient voor het scheiden van het uitzicht van een user interface en de inhoud ervan.
Op deze manier kan de inhoud van de uiteindelijke user interface variabel gemaakt worden om ze bijvoorbeeld in verschillende talen te kunnen aanbieden.

Om dit te realiseren worden er verschillende content-elementen aangemaakt, met een uniek id. Deze elementen bevatten definities van de inhoud. Deze definities kunnen dan gebruikt worden in het property-gedeelte op dezelfde manier als het gebruik van variabelen bij programmeren.

```
<content id=”Dutch”>
        <constant id=”NameLabel”>Naam</constant>
        <constant id=”AgeLabel”>Leeftijd</constant>
</content>

<content id=”French”>
        <constant id=”NameLabel”>Nom</constant>
        <constant id=”AgeLabel”> âge</constant>
</content>
```

Listing 6: Het gebruik van het “content”-element

3.3.1.5 Behavior

Het “behavior”-element definieert het interactieve gedrag van de user interface. Dit element bestaat uit een aantal “rules” die elk op hun beurt bestaan uit een “condition”- en een “action”-gedeelte. Wanneer aan de voorwaarde voldaan is, wordt de actie uitgevoerd. Er zijn verschillende soorten conditions en actions mogelijk.

Een condition in UIML kan zijn het plaatsvinden van een event in de UI zoals bijvoorbeeld het indrukken van een knop, eventueel in combinatie met een expressie die een bepaalde waarde moet hebben.

Voor de actions zijn er meerdere mogelijkheden. Een action die uitgevoerd wordt kan de waarde van een property instellen, een aanroep doen van externe applicatielogica, een event oproepen in de interface of de interface herstructureren. Een combinatie van de voorgaande opties behoort uiteraard ook tot de mogelijkheden.

```
<behavior>
        <rule>
                <condition>
                        <event class=”ButtonSelected  part-name=”b5”/>
                </condition>
                <action>
                        <call name=”Server.Start” />
                </action>
        </rule>
</behavior>
```

Listing 7: Gebruik van conditions en actions

3.3.1.6 Peers

Het “peers”-gedeelte van het UIML-document document beschrijft de mapping naar buiten toe. Het definieert de mapping tussen entiteiten die gebruikt worden binnen het UIML-document en externe entiteiten.

Er zijn twee soorten entiteiten waarvoor een mapping bestaat in UIML: van abstracte klassen naar concrete widgets en van method calls naar externe logica.

Deze twee mappings krijgen elk een element binnen het peers-gedeelte: de presentation-peers en de logic-peers.

3.3.1.7 Presentation

In de presentation-peer wordt gedefinieerd welke vocabulary er gebruikt wordt om deze interface te renderen. Een vocabulary definieert hoe de mapping gebeurt van de abstracte klassen naar de concrete widgets voor de gebruikte toolkit, en welke properties en events er beschikbaar zijn voor deze klassen in het UIML-document. Een concreet voorbeeld van een mapping is het mappen van een part met de klasse “Button” naar een “System.Windows.Forms.Button”[17] voor gebruik binnen het .Net platform.

Het is vrij gemakkelijk om zelf een vocabulary te definiëren maar op de officiële site van de UIML-standaard zijn er reeds een aantal voorgedefinieerde vocabularies[18] te vinden.

```
<peers>
        <presentation  base=”swf-1.1.uiml” />
</peers>
```

Listing 8: Het refereren naar een vocabulary

Bovenstaand voorbeeld toont hoe een referentie naar de “windows forms”-vocabulary wordt aangegeven binnen een UIML-document. Er wordt geen voorbeeld gegeven van het zelf definiëren van een vocabulary omdat dit toch geen nut heeft in het kader van deze thesis.

3.3.1.8 Logic

Het “logic”-element geeft aan hoe method-calls die binnen het UIML-document gebeuren via de “call”-tag, vertaald moeten worden naar aanroepen van de externe applicatielogica. De applicatielogica die aangesproken moet worden kan zich lokaal bevinden in de vorm van libraries, benaderd worden met technieken als webservices of gewoon in het UIML-document zitten in de vorm van een scriptje.

[17] http://msdn2.microsoft.com/en-us/library/system.windows.forms.button.aspx
[18] http://uiml.org/toolkits

```
<peers>
        <logic>
                <d-component name=”Server” maps-to=”Edm.ServerClass”>
                        <d-method  name=”Start”  maps-to=”StarServer” />
                </d-component>
        </logic
</peers>
```

Listing 9: De mapping naar externe applicatielogica

3.3.2 Lay-out van de user interface

Ondanks alle voordelen die UIML biedt is er toch ook een belangrijk nadeel [27]. De standaard definieert (nog) geen platformonafhankelijke manier om de lay-out binnen een grafische user interface aan te geven.

Sommige toolkits, zoals Java Swing[19] of GTK[20] werken met lay-out managers. Anderen, zoals System.Windows.Forms[21] in .Net werken met een absolute positionering.

Dit heeft tot gevolg dat bij het gebruik van UIML toch moet gewerkt worden met platformspecifieke constructies, wat er in de praktijk op neerkomt dat er vaak platformspecifieke informatie over de lay-out vervat zit in de structure- en style-elementen.

Dit impliceert dat er meerdere versies van (stukken van) het UIML-document gemaakt moeten worden, waarbij het de taak van de ontwikkelaar is om ervoor te zorgen dat deze versies qua lay-out consistent blijven. Dit zorgt weer voor een hoop extra werk en gaat rechtstreeks in tegen de gedachte van hergebruik, die in UIML centraal staat. Eenzelfde interface in UIML zou dan slechts bruikbaar zijn op een beperkt aantal soorten apparaten.

Een mogelijke oplossing voor dit probleem is een aanpak die de UIML-standaard uitbreidt met ondersteuning voor de specificatie van de lay-out via constraints die de spatiale relaties tussen de (abstracte) interactie-elementen vertegenwoordigen [28,29]. De constraints worden dan door een constraint solver opgelost bij het renderen voor een specifiek platform, rekening houdend met de mogelijkheden van dit platform. Zo wordt er een platformspecifieke lay-out geproduceerd die dan gerenderd kan worden. Voor het oplossen van de constraints hebben de auteurs van de eerder aangehaalde papers de constraint solving toolkit Cassowary[22] geport naar het .Net platform onder de naam Cassowary.Net[23].

Deze uitbreiding is gekend onder de naam *Device-Independent UIML* (DI-UIML) en behoort (nog) niet to de UIML-standaard.

[19] http://java.sun.com/docs/books/tutorial/uiswing/
[20] http://www.gtk.org/
[21] http://msdn2.microsoft.com/en-us/library/system.windows.forms.aspx
[22] http://sourceforge.net/projects/cassowary/
[23] http://jozilla.net/software/cassowary_net

3.4 Conclusie

In dit hoofdstuk hebben we gezien hoe we de nodige abstractie kunnen verkrijgen bij het opstellen van de user interface via een beschrijvingstaal en wat de voordelen van deze aanpak zijn. Verder hebben we de basisbeginselen besproken van UIML, de beschrijvingstaal die we in deze thesis zullen gebruiken. In het volgende hoofdstuk zullen we aandacht besteden aan bestaande software die we zullen gebruiken om ons distributie-systeem te ontwikkelen.

4 Architectuur

4.1 Inleiding

In deze thesis willen we een distributie-systeem ontwikkelen dat gebruikt kan worden in een situatie zoals die uit ons voorbeeldscenario. De nadruk ligt hierbij op het vinden van een simpele en gebruiksvriendelijke manier om de verdeling uit te voeren. Voor het implementeren van onze software gaan we daarom niet van nul beginnen, maar gebruik maken van reeds bestaande projecten. In dit hoofdstuk bespreken we achtereenvolgens UIML.Net, de renderer die we zullen gebruiken om onze UIML-beschrijving om te zetten in een concrete interface en Web To Peer, het middleware framework dat we gebruiken voor de communicatie tussen de verschillende apparaten uit onze device federation.

4.2 UIML.Net

UIML.Net[24] [22,23] is een renderer voor het renderen van de UIML-documenten die we reeds eerder in deze thesis besproken hebben.

Deze renderer is ontwikkeld voor het .Net platform, en ondersteunt zowel het gewone .Net framework[25] en het .Net compact framework[26] als Mono[27], aangezien deze allemaal op dezelfde ECMA-standaard[28] gebaseerd zijn. In deze thesis maken we enkel gebruik van de eerste twee.

Het project is gestart in 2003 met als doel het ontwikkelen van een adaptieve en flexibele renderer die op meerdere platformen kan draaien en die evolutie in widget-sets en variaties in de gebruikte vocabularies kan ondersteunen.
De renderer is gratis beschikbaar en wordt uitgebracht onder de GNU Lesser General Public License[29]. De huidige versie van de software is een implementatie van versie 3.0 van de UIML-specificatie [24].

UIML.Net kan concrete interfaces renderen op basis van de volgende widget-sets:

- System.Windows.Forms[30]
- System.Windows.Forms op het .Net compact framework
- GTK#[31]
- Een gedeelte van Wx.Net[32]

[24] http://research.edm.uhasselt.be/uiml
[25] http://msdn2.microsoft.com/en-us/netframework/aa569263.aspx
[26] http://msdn2.microsoft.com/en-us/netframework/aa497273.aspx
[27] http://www.mono-project.com
[28] http://www.ecma-international.org/publications/standards/Ecma-335.htm
[29] http://www.gnu.org/licenses/lgpl.html
[30] http://msdn2.microsoft.com/en-us/library/system.windows.forms.aspx
[31] http://gtk-sharp.sourceforge.net/
[32] http://wxnet.sourceforge.net/

In deze thesis hebben we er voor gekozen enkel Windows Forms te gebruiken, zowel op het gewone framework als de compact-versie.

Bij het werken met beschrijvingstalen zoals UIML zijn er twee mogelijkheden voor het verkrijgen van een concrete interface [22].

- Het document op voorhand compileren en het resultaat gebruiken bij het uitvoeren. Dit resultaat kan programmacode zijn in een bepaalde taal zoals C#, of een document in een markup taal zoals HTML.

- Het document interpreteren op het moment dat de interface weergegeven moet worden.

De eerste aanpak heeft als voordeel dat er niet zomaar op het resultaat vertrouwd moet worden, maar dat dit op voorhand beschikbaar is en nog door de ontwikkelaar aangepast kan worden indien gewenst. Het nadeel is dan weer dat deze compilatie-stap nodig is, telkens er aan het ontwerp van de interface iets veranderd wordt.

De tweede aanpak is eigenlijk vergelijkbaar met de werking van bijvoorbeeld een PHP[33]-script. Het UIML-document wordt pas tijdens het uitvoeren omgezet naar een concrete interface in een specifieke toolkit, zonder dat daarvoor nog een tussenstap nodig is na het editeren van het document. Dit is bijvoorbeeld zeer handig indien er gewerkt wordt met rapid prototyping. Ook is de flexibiliteit die deze aanpak biedt zeer wenselijk wanneer het de bedoeling is dat een renderer op meerdere platformen kan draaien. Dit geeft eventueel de mogelijkheid om platformspecifieke aanpassingen aan het UIML-document te doen alvorens dit gerenderd wordt om zo een grotere gebruiksvriendelijkheid na te streven. Het nadeel is wel dat de benodigde renderer hiervoor complexer is.

UIML.Net is een renderer van het tweede type. Dit is wat we nodig hebben in deze thesis zodat we UIML-documenten kunnen doorsturen naar de gebruikte apparaten, waar ze onmiddellijk gerenderd en weergegeven worden. Het heeft geen zin op voorhand een extra compilatiestap te doen omdat er toch geen aanpassing meer gebeurt door de ontwikkelaar en de flexibiliteit die deze aanpak biedt geeft ons de mogelijkheid om te werken met variaties op de gebruikte toolkit (SWF voor gewoon .Net framework en Compact framework) en in toekomstig werk eventueel andere toolkits zoals GTK#.

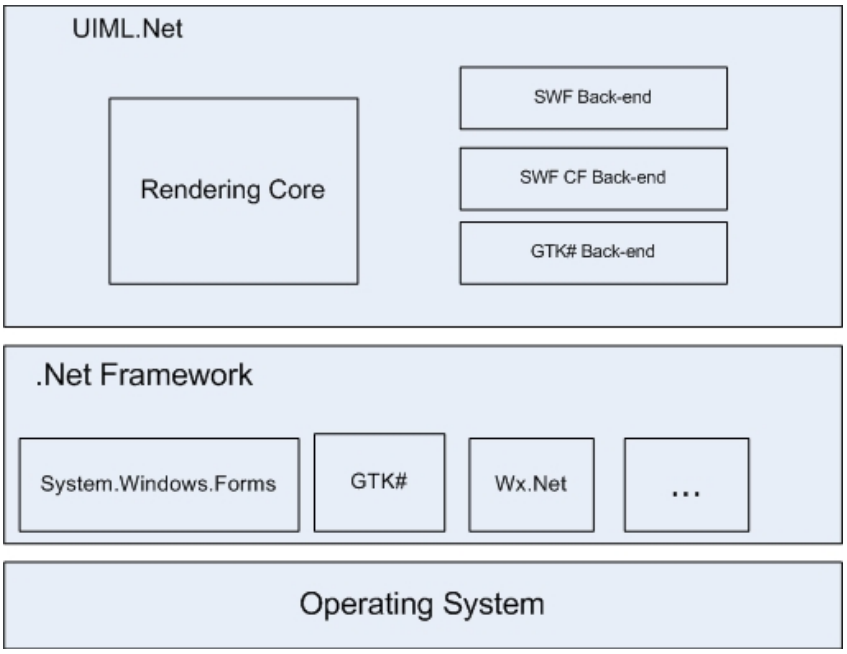
4.2.1 Architectuur

De implementatie van de renderer is gebouwd bovenop het .Net platform en de aanwezigheid van een .Net virtual machine is dan ook een vereiste voor elk platform waarop de software uiteindelijk moet draaien. Er is voor de rest geen afhankelijkheid van een specifieke versie van de virtual machine.

In grote lijnen bestaat de architectuur van UIML.Net uit twee delen. Er is één generieke rendering-core en er zijn een aantal backends. Deze backends zijn widgetset-specifiek en hier wordt dus tijdens elke uitvoering van het rendering-proces maar één exemplaar van gebruikt. Binnen deze thesis gebruiken we de SWF back-end of de SWF CF back-end, afhankelijk van het apparaat waarop het rendering-proces plaatsvindt.

[33] http://www.php.net

De core is verantwoordelijk voor zaken zoals het inlezen en verwerken van een UIML-document dat gerenderd moet worden. De backends op zijn op hun beurt verantwoordelijk voor het uitvoeren van code die specifiek is voor de gebruikte widget-set.



Figuur 14: Architectuur van UIML.Net (Figuur geïnspireerd door [22])

Er zijn twee manieren om een renderer te implementeren voor een beschrijvingstaal [22]:

- **Statisch**: Bij deze aanpak wordt er in de implementatie specifieke informatie over de widget-set verwerkt. De informatie over de GUI library die gebruikt zal worden bij het renderen moet gekend zijn op het moment dat de renderer gecompileerd wordt. Deze aanpak biedt dus weinig flexibiliteit.

- **Dynamisch**: Deze aanpak vergt op voorhand geen specifieke kennis van de gebruikte widget-set. Er wordt gesteund op de mapping-informatie die voorzien wordt door de gebruikte vocabulary. Met deze informatie worden de nodige widgets dan aangemaakt door middel van reflectie. Deze aanpak is flexibeler en levert een renderer die gemakkelijker herbruikt kan worden voor andere widget-sets of variaties van eenzelfde widget-set op verschillende platformen.

UIML.Net is een renderer van het tweede type. We hebben dit soort dynamische renderer nodig in deze thesis omdat er gewerkt wordt met de gewone SWF widget-set en die voor het .Net compact framework, waar we twee verschillende vocabularies met mapping-informatie voor nodig hebben.

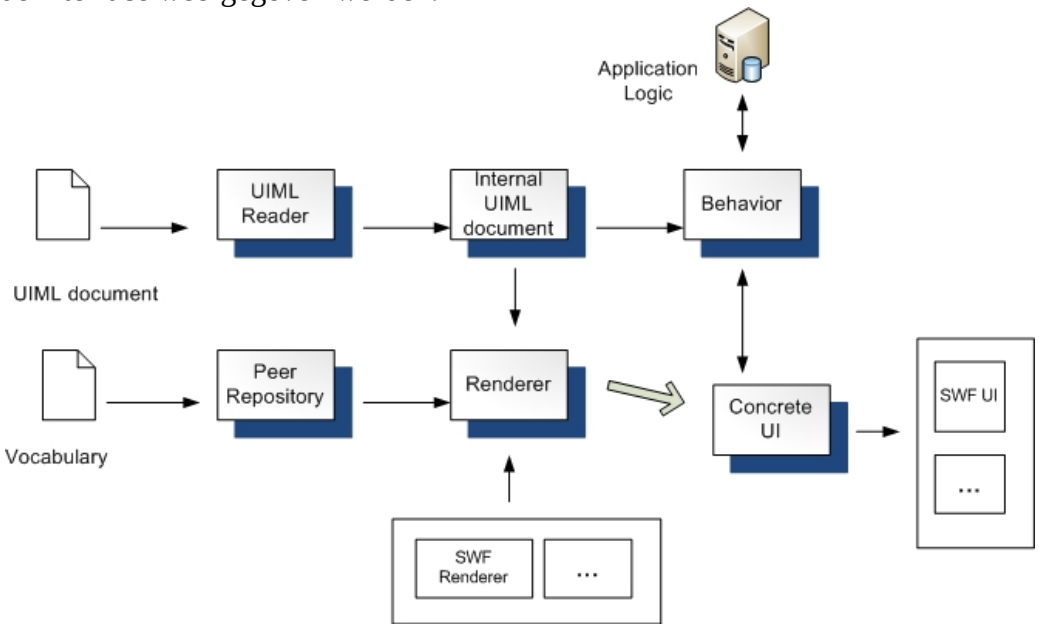
4.2.2 Rendering-proces

Wanneer we een UIML-document willen renderen naar een interface zorgt de renderer er eerst ervoor dat het UIML-document ingelezen wordt en verwerkt wordt tot een datastructuur in het geheugen zodat er gemakkelijk mee te werken valt. Deze datastructuur heeft de vorm van een tree waarbij bijvoorbeeld elk part vertegenwoordigd is op de correcte plaats in de hiërarchie van abstracte interface-elementen. Deze abstracte interface elementen hebben allemaal koppelingen naar de property-elementen die op hen van toepassing zijn.

De core renderer kijkt welke vocabulary er in het UIML-document gebruikt wordt en zal op basis hiervan de juiste back-end laden die dan het genereren van de concrete interface met de correcte widget-set op zich neemt.

Wanneer de interface dan effectief gerenderd wordt, geeft de rendering-core de geheugenstructuur met het UIML-document door aan een instantie van de widgetset-specifieke renderer. Deze gaat dan heel de tree af en zoekt voor elk abstract element het concreet type van widget nodig voor het renderen. Van dit widget wordt vervolgens een object aangemaakt, de nodige properties uit de style worden toegepast en het part in de geheugenstructuur krijgt een referentie naar dit concrete object. Dit is belangrijk voor het kunnen manipuleren van de interface eens het gerenderde resultaat op het scherm staat.

We hebben in deze thesis ook een belangrijke wijziging aangebracht aan de standaardcode van de renderer. Deze wijziging zorgt ervoor dat er tijdens het renderen een hashtabel gevuld wordt met de concrete widgets en de UIML-parts die ze voorstellen. De objectreferentie van het concrete widget wordt als key gebruikt en het corresponderende UIML-part als waarde. Dit stelt ons in staat om nadat de interface gerenderd is een UIML-part te vinden op basis van een concreet widget. In hoofdstuk 5, dat gaat over de implementatie van onze oplossing, zullen we zien dat dit belangrijk is om op een efficiënte manier erachter te komen welke UIML-parts de gebruiker wil doorsturen naar een ander apparaat om te laten weergeven. Nadat de concrete interface gerenderd is wordt de applicatielogica hier nog aan gekoppeld en dan kan de interface weergegeven worden.



Figuur 15: Verwerking UIML-document door de renderer (Figuur geïnspireerd door [22])

4.3 Web to Peer

In deze thesis werken we met gedistribueerde “device federations”. Een basisvereiste voor het functioneren van het uiteindelijke systeem is dan uiteraard dat er communicatie mogelijk moet zijn tussen alle apparaten die deel uitmaken van het geheel.

Om de communicatie te verzorgen binnen deze heterogene verzameling van apparaten, met mogelijk verschillende softwareplatformen per apparaat, kunnen we gebruik maken van middleware.

Middleware[34] is technologie die ervoor zorgt dat verschillende applicaties met elkaar kunnen communiceren en hen zo in staat stelt gegevens uit te wisselen. Deze functionaliteit kan gebruikt worden om bijvoorbeeld oudere applicaties, zogenaamde “legacy software”, te koppelen aan nieuwere applicaties wanneer men deze samen moet laten werken.
Middleware wordt echter ook gebruikt bij het werken met gedistribueerde applicaties, en dat is precies wat er in deze thesis gebeurt.

We hebben middelware-laag nodig die zich bovenop het besturingssysteem bevindt en de communicatie verzorgt tussen de verschillende apparaten in het systeem. Deze laag abstraheert dan de heterogeniteit van de platformen die in het systeem draaien, en het feit dat er gedistribueerd gewerkt wordt. Zo kunnen we een omgeving implementeren waarbij er een aantal verschillende apparaten zijn die services aanbieden, services verbruiken, of een combinatie van beiden.

4.3.1 Web to Peer Middelware

Het W2P[35]-middleware framework, beschikbaar onder de LGPL-licentie, is ontworpen om de integratie van webgebaseerde services in een heterogene omgeving te bevorderen [30]. Aangezien tegenwoordig steeds meer (mobiele) apparaten beschikken over de mogelijkheid tot webtoegang maakt W2P gebruik van webtechnologie om apparaten onderling te laten communiceren op een peer-to-peer wijze waarbij alle aangesloten apparaten evenwaardig zijn en niet noodzakelijk in een vaste rol van server of client geduwd worden.

Om applicaties de mogelijkheid te geven te communiceren binnen het netwerk maakt W2P gebruik van het concept *Mobile Web Services* (MWS). Deze zijn een light-weight variant op gewone web services die zowel gebruikt worden voor het aanbieden van diensten als het consumeren hiervan.

Bij gewone webservices wordt er gebruik gemaakt van een servlet container zoals Apache Tomcat[36] of Jetty[37], die beiden beschikbaar zijn onder de Apache Licentie.
Deze container dient voor de communicatie tussen de web service en de buitenwereld. De container handelt het ontvangen en versturen van berichten af zodat de web service zich enkel moet bezighouden met zijn kerntaken. De client applicaties communiceren met de web service door het sturen van requests. Deze worden door de container doorgegeven aan de web

[34] http://middleware.objectweb.org/
[35] http://research.edm.uhasselt.be/w2p
[36] http://tomcat.apache.org/
[37] http://jetty.mortbay.com/

service, en het antwoord volgt de omgekeerde weg. Dit request/response mechanisme is het normaal dus zo dat de client de communicatie initieert met een uitgaand request en de server hierop een reply geeft.

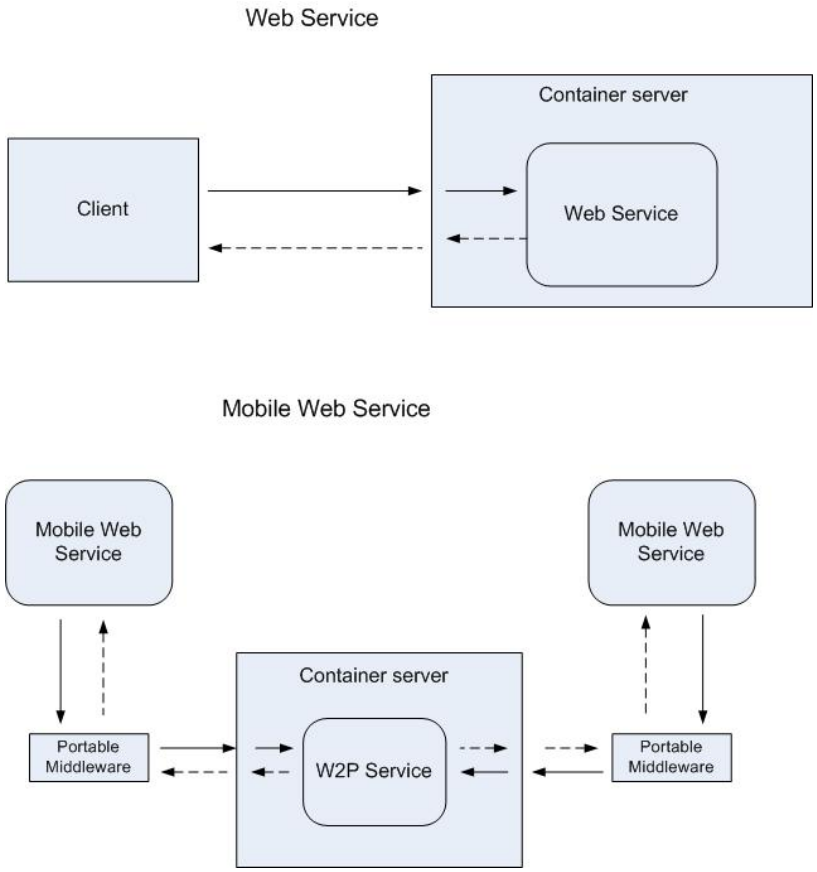
Gewone web services zijn echter niet zomaar geschikt voor het doel dat wij voor ogen hebben. Web services hebben twee grote nadelen die hen ongeschikt maken voor gebruik binnen het systeem dat wij in deze thesis willen realiseren.

- Omwille van het gebruik van een container voor het afhandelen van de applicatie hebben gewone web services nog altijd een vrij grote “footprint” wat betreft resource-gebruik. In de device federations waar ons systeem voor bedoeld is kunnen zich echter ook apparaten zoals een PDA bevinden, die slechts beschikken over een beperkte hoeveelheid resources.

- Gewone web services zijn fel gericht op de ”client/server” manier van werken, waarbij er een server is die zijn diensten aanbiedt in de vorm van een web service en een client hiervan gebruik maakt via een request/response mechanisme. In ons systeem is het echter nodig dat de communicatie langs beide kanten geïnitieerd kan worden.

Mobile web services zijn gebaseerd op dezelfde concepten als gewone web services met het verschil dat er bij gebruik van mobile web services geen nood is aan een container om de communicatie af te handelen. Voor communicatie maken ze gebruik van de speciaal ontwikkelde *Portable Middleware* (PM) library [30].

Bij het gebruik van mobile web services is de portable middleware library nodig voor zowel het aanbieden van diensten als het verbruiken hiervan. Het is de PM die de verbinding afhandelt tussen de entiteit die wil communiceren en de W2P service, de centrale communicatie-server die verantwoordelijk is voor het doorsturen van de berichten naar zowel individuele entiteiten als groepen.



Figuur 16: Werking van gewone web service en mobile web services (Figuur geïnspireerd door [30])

Bij het aanbieden of gebruiken van mobile web services moet er via de portable middleware library verbinding gemaakt worden met de W2P Service, welke een gewone webservice is die draait binnen een container server. Binnen het systeem dat wij in deze thesis ontwikkeld hebben wordt hiervoor gebruik gemaakt van Jetty.

Het is dan de taak van de PM om te zorgen dat er een constante verbinding behouden wordt met de W2P service, zodat communicatie in beide richtingen mogelijk is. Wanneer de verbinding even wegvalt kan de PM nog terug verbinding maken en de sessie gewoon voortzetten. Binnenkomende berichten die niet afgeleverd konden worden, worden dan op de server bijgehouden en bij het terug verbinden afgeleverd. Dit kan handig zijn in het systeem uit deze thesis wanneer er gewerkt wordt met een draadloze verbinding aangezien er altijd een mogelijkheid bestaat dat de verbinding kortstondig wegvalt. Wanneer deze binnen een beperkte tijd terug opgenomen wordt werkt het systeem gewoon verder.

Eens een verbinding wegvalt blijft de sessie echter maar een beperkte tijd actief op de server. Indien er niet opnieuw verbonden wordt binnen een bepaalde tijd, valt de sessie definitief weg en moet er volledig opnieuw verbonden worden. Dit kan in het systeem dat wij gerealiseerd hebben bijvoorbeeld voorvallen wanneer een gebruiker met een PDA met draadloze netwerkverbinding zich te lang buiten het bereik van het signaal verplaatst. Uiteraard is het ook mogelijk dat de software op een apparaat in onze federation gecrasht is of dat het apparaat fysiek beschadigd is geraakt.

Op dit moment zijn er client libraries[38] van W2P beschikbaar voor Java, C++ en C#. Deze kunnen gebruikt worden om een dienst aan te spreken vanuit programma's die geschreven zijn in één van deze talen, of ze kunnen gebruikt worden om programma's die in deze talen geschreven zijn aanspreekbaar te maken. Op deze manier is het ook relatief eenvoudig om een plugin te ontwikkelen voor bestaande software, zodat deze zijn functionaliteit via webtechnologie kan aanbieden. Een mooi voorbeeld hiervan is de plugin[39] die gemaakt werd voor de XMMS mediaspeler.

In deze thesis wordt enkel gebruik gemaakt van de client library voor C# aangezien dit de enige taal is die wij gebruiken om ons systeem te programmeren.

4.3.1.1 Messaging

Web to Peer is een messaging framework [30] wat wil zeggen dat de communicatie tussen de delen van het systeem gebeurt door het uitwisselen van berichten in de vorm van messages.

In het systeem dat wij ontwikkelen zullen dus vooral berichten uitgewisseld worden van apparaten die zich aanmelden bij de device federation om zich beschikbaar te stellen en berichten die een UIML-document bevatten voor deze apparaten om weer te geven.

Om adresseerbaar te zijn binnen het systeem krijgt elk stuk software dat zich aanmeldt een naam, die uiteraard uniek moet zijn voor een bepaalde instantie van onze device federation. Hiervoor zijn 2 mogelijkheden:

- De geregistreerde entiteit krijgt een naam die door het systeem automatisch gegenereerd wordt. Standaard is een dergelijke naam van de vorm "Peer0", "Peer1", enz…

- De geregistreerde entiteit kan een vaste zelf een bepaalde naam registreren, indien deze nog niet in gebruik is.

In ons systeem wordt de tweede manier gebruikt. De apparaten die zich willen aanmelden om een deel van de interface te krijgen, geven bij hun aanmelding een apparaatnaam mee. Indien deze naam reeds bezet is volgt er een foutmelding. Wanneer de naam beschikbaar is, wordt de verbinding gemaakt en wordt de naam in het venster van de distributie-server vermeld bij de visualisatie die het apparaat voorstelt. Zo kan de gebruiker zien welke visualisatie overeenkomt met welk apparaat.

Wanneer entiteiten hun unieke naam verkregen hebben kunnen ze berichten uitwisselen met elkaar. Deze berichten bestaan telkens uit drie delen:

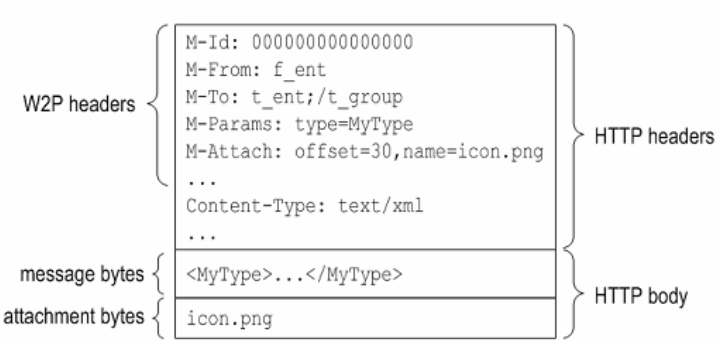
- Een **header** die meta-data bevat over het bericht. Hierin zitten de naam van de afzender, de naam van de geadresseerde of de groep waar het bericht voor bestemd is, en bij ons systeem ook het type van bericht om aan te geven of het gaat om een aan- of afmeldingsbericht, of een stuk van de interface dat weergegeven moet worden.

[38] http://research.edm.uhasselt.be/w2p/download.php
[39] http://research.edm.uhasselt.be/w2p/projects.php

- Het **content-gedeelte** bevat de eigenlijke data van het bericht. Dit gedeelte kan in principe elk soort data bevatten van gewone tekst tot binaire bestanden, maar in het systeem dat wij gerealiseerd hebben worden alle berichten uitgewisseld in XML-formaat.

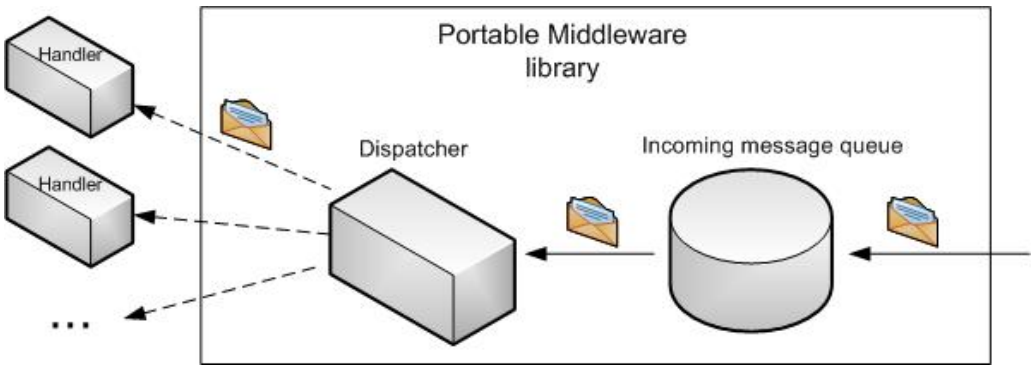
- Een bericht kan eventueel nog **attachments** hebben. In ons voorbeeldscenario zou deze mogelijkheid gebruikt kunnen worden om de foto die weergegeven moet worden op de televisie door te sturen naar het apparaat.

Standaard worden berichten in W2P verstuurd via het HTTP-protocol[40] en in ons systeem is dit ook het geval. De berichten krijgen dan de volgende structuur:



Figuur 17: Structuur van een W2P-bericht (Figuur overgenomen uit [30])

Voor het afhandelen van binnenkomende berichten wordt in ons systeem ook gebruik gemaakt van een dispatcher. Een dispatcher is een object dat de binnenkomende berichten opvangt en verwerkt op basis van een aantal gemakkelijk in te stellen regels. Op basis van bijvoorbeeld de afzender van het bericht, wordt dit bericht doorgestuurd naar een gepaste message-handler. Deze handler zal dan zorgen voor de verdere verwerking van het bericht. Het gebruik van een dispatcher zorgt ervoor dat ons programma niet constant blocking-calls moet doen van Peer.Receive() om berichten te ontvangen en zorgt ook dat we de periodieke ping-berichten niet doorkrijgen voor verwerking maar dat deze gewoon beantwoord worden zonder de applicatie verder aan te spreken.



Figuur 18: Dispatcher in W2P (Figuur geïnspireerd door [30])

[40] http://www.w3.org/Protocols/

4.3.1.2 Groepen en events

Een van de dingen die W2P zo flexibel maken is het feit dat er op een eenvoudige manier gebruik gemaakt kan worden van groepen om berichten in één keer te versturen naar een verzameling entiteiten in plaats van deze individueel te moeten adresseren.
Beheer van groepen wordt volledig transparant afgehandeld door de W2PS service waarbij de creatie van een groep automatisch gebeurt zodra een entiteit zichzelf abonneert op deze groep. De groep blijft bestaan zolang deze geregistreerde leden heeft. Om een bericht naar de volledige groep te sturen geeft een afzender als geadresseerde gewoon de groepsnaam op (vb "/interface") en de W2P service handelt de rest af.

De groepen die gevormd worden binnen W2P zijn ook hiërarchisch en kunnen dus subgroepen hebben. Wanneer een boodschap naar de leden van een groep wordt doorgestuurd wordt deze boodschap automatisch ook bezorgd aan de leden van de parent-groep. Deze manier van werken stelt het systeem in staat om op een eenvoudige manier te werken met events.

Het gebruik van events in een gedistribueerd systeem zorgt voor een vrij losse koppeling tussen de verschillende onderdelen, wat de flexibiliteit en robuustheid van het geheel dan weer ten goede komt. Door het gebruik van events kan een deel van het systeem kenbaar maken dat het geïnteresseerd is in het ontvangen van een bericht telkens er zich een vooraf bepaald event voordoet in (een deel van) het systeem. De W2P-service zorgt dan dat de geïnteresseerde entiteit dat bericht ontvangt zonder dat deze entiteit contact moet onderhouden met de entiteit waar het event van uitgaat.

In ons systeem is op dit moment nog geen ondersteuning voor het effectief functioneren van de interface eens deze gedistribueerd is over de apparaten. Dit is naar de toekomst toe een belangrijke uitbreiding en bij de implementatie ervan zal het eventing-systeem van W2P waarschijnlijk zeer nuttig zijn voor het doorgeven van wijzigingen in delen van de interface naar de apparaten die deze delen weergeven. Daarom vermelden we dit systeem toch al kort.

Concreet moeten alle delen van het systeem die hun events kenbaar willen maken aan de rest van het systeem één regel in acht nemen. Wanneer een event zich voordoet moet de entiteit een bericht sturen dat geadresseerd is aan een groep met een naam van de vorm "/naam_van_entiteit/naam_van_event". Zo kan een andere entiteit gemakkelijk events ontvangen door simpelweg gebruik te maken van het eerder aangehaalde systeem van groepen en subgroepen. De ontvanger moet zich gewoon abonneren op de gewenste groep. Het groepssysteem biedt zo aan een entiteit ook de mogelijkheid om zich in één keer te abonneren op alle events van een bepaald deel van het systeem door gewoon te abboneren op de hoofdgroep.

4.4 Conclusie

In dit hoofdstuk hebben de reeds bestaande software besproken waar we op steunen bij het ontwikkelen van onze oplossing. UIML.Net wordt ingezet als rendering-engine voor onze UIML-beschrijvingen om zo interfaces te kunnen weergeven op diverse platformen, en het Web To Peer middleware-framework zal de interplatform-communicatie afhandelen tussen de devices uit onze federation. In het volgende hoofdstuk gaan we kijken naar de concrete oplossing die er in deze thesis ontwikkeld werd.

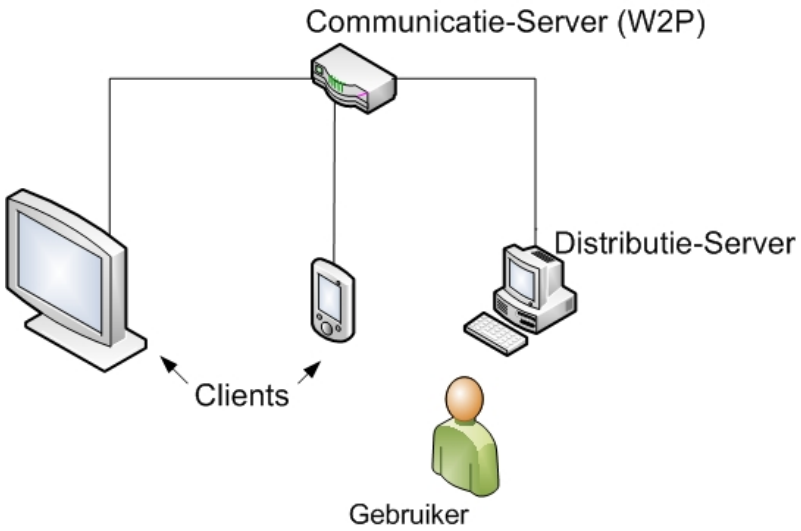
5 Implementatie

5.1 Inleiding

In de voorgaande hoofdstukken werden de principes en de bestaande software besproken waarop gesteund werd bij het ontwikkelen van de softwareoplossing voor deze thesis. In dit hoofdstuk wordt alles samengebracht en wordt de concrete software beschreven. We zullen zien hoe de gebruiker uit ons voorbeeldscenario het gerealiseerde systeem kan gebruiken om zijn probleem op te lossen en de interface van een applicatie te verdelen over verschillende apparaten.

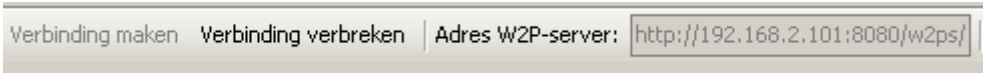
5.2 Architectuur

Het systeem dat ontwikkeld werd bestaat op logisch niveau uit drie delen, zoals afgebeeld in de onderstaande figuur:



Figuur 19: Architectuur van de implementatie

Wanneer de gebruiker uit het voorbeeldscenario de interface van zijn programma wil verdelen over meerdere apparaten zal hij hiervoor het initiatief nemen via de distributie-server. Nadat hij de distributie-server opgestart heeft maakt hij verbinding met de communicatie-server via een op voorhand gekend service-adres.



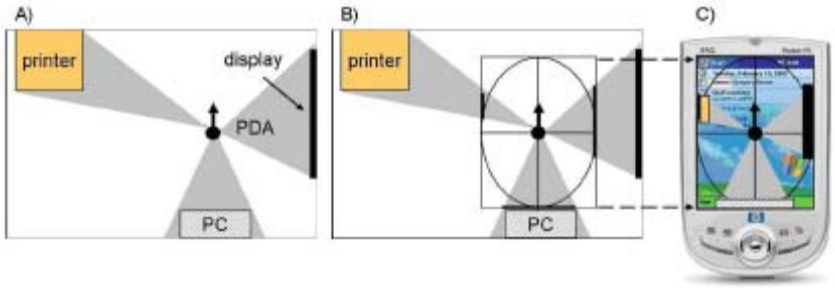
Figuur 20: Aanmelden van distributie-server bij de communicatie-server

Vervolgens start hij op elk van de client-apparaten de client-software die in deze thesis ontwikkeld werd (zie verder). Hiermee maakt hij vanop deze apparaten ook verbinding met de communicatie-server. De distributie-server registreert dit en het apparaat wordt in het venster van de distributie-server weergegeven. Vanaf dan is het apparaat beschikbaar om stukken van de interface te ontvangen.

5.3 Visualisatie van verbonden clients

Het visualiseren van de verbonden clients op de distributie-server is geïnspireerd door het systeem van RelateGateways [32,33,34].
Het project van RelateGateways heeft als bedoeling een systeem te ontwikkelen dat een gebruiker ondersteunt bij het ontdekken en gebruik maken van aangeboden services in zijn onmiddellijke omgeving via een mobiel apparaat [33].

Het systeem beschikt over een aantal widgets die rondom de rand van het scherm geplaatst worden. Deze widgets stellen de apparaten in de omgeving voor die services aanbieden waar de gebruiker mee kan werken. De locatie van een widget op het scherm is verbonden met de fysieke locatie waar het overeenkomstige apparaat zich bevindt. Hiervoor wordt gebruik gemaakt van sensortechnologie en een kompas-metafoor. De kompas-metafoor wordt gebruikt om de relatieve positie van de apparaten ten opzichte van de gebruiker te visualiseren. Deze weergave wordt continu geüpdate.



Figuur 21: Kompas-metafoor van RelateGateways (Figuur overgenomen uit [34])

De bedoeling van dit systeem is tweeledig: Ten eerste wordt de gebruiker ervan bewust gemaakt welke services er beschikbaar zijn in zijn omgeving, en welke apparaten deze aanbieden. Ten tweede biedt dit systeem een mogelijkheid om op een eenvoudige en gebruiksvriendelijke manier van deze services gebruik te maken via de widgets.

De widgets representeren dus niet enkel de positie van het apparaat maar bieden ook mogelijkheden tot interactie door middel van bijvoorbeeld “drag-and-drop”-functionaliteit.

De widgets hebben een beperkte afmeting en zijn langs de rand van het scherm geplaatst om niet storend te werken, maar toch goed genoeg geïntegreerd om interactie mogelijk te maken.

Dit is een interessante manier van werken voor het gerealiseerde systeem uit deze thesis omdat dit de mogelijkheid geeft om op een niet-storende manier de clients te visualiseren en er interactie mee te hebben. De real-time tracking van de apparaten hebben is echter niet overgenomen. Hoewel het ongetwijfeld een mooi resultaat kan opleveren wanneer dit

geïntegreerd wordt in het systeem, ligt hierop niet de focus in deze thesis en lijkt het trouwens niet zo voor de hand liggend om zulk een systeem te implementeren. Dit lijkt bevestigd te worden door het feit dat de ontwikkelaars van het systeem in hun paper [33] vermelden dat hun prototype zelf ook geen werkend sensorsysteem heeft en dat gebruikerstesten gebeuren via de“Wizard-of-Oz”-techniek[41].

De visualisatie van de verbonden clients is dus overgenomen voor het systeem uit deze thesis en ook het feit dat deze visualisatie de mogelijkheid biedt tot interactie.

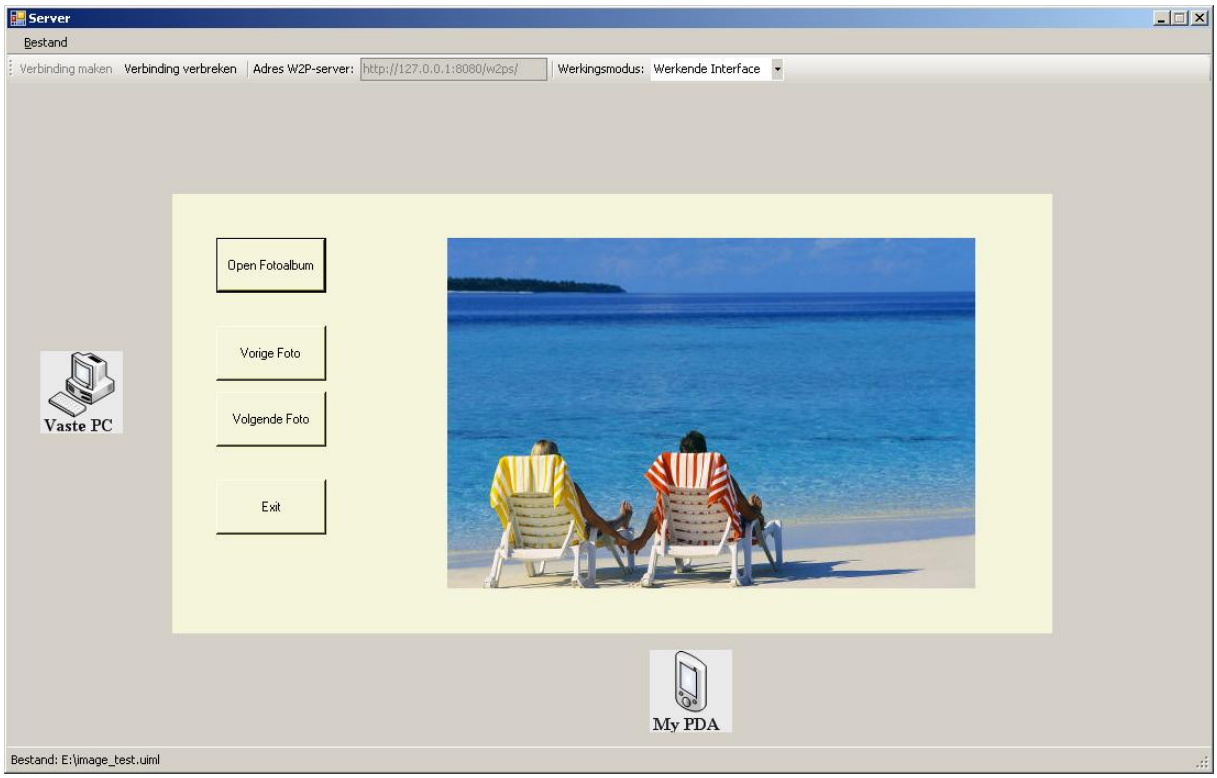
Voor elke client die zich aanmeldt komt er een afbeelding bij langs één van de vier randen van het scherm van de distributie-server. Omdat het enkel draait om de visualisatie worden de afbeeldingen op een willekeurig uitgekozen plaats langs een willekeurige rand gezet. Als een afbeelding wordt toegewezen aan een rand waar zich al afbeeldingen bevinden, wordt er gezorgd dat deze elkaar niet kunnen overlappen. Zo kunnen de vier randen van het server-venster volledig vol gezet worden met afbeeldingen van aangemelde clients.

Als een client zich afmeldt, of de communicatie-server zijn sessie beëindigt omdat de verbinding te lang is weggevallen, wordt de afbeelding op de distributie-server ook verwijderd. Zo wordt dus een dynamische weergave verkregen die op elk moment op een niet-storende manier, weergeeft welke clients beschikbaar zijn. Wanneer de verbinding met een client-apparaat wegvalt wordt de gebruiker hier in het server-venster op attent gemaakt via een messagebox. Het verschijnen van een messagebox bij elke afmelding lijkt op het eerste zicht niet storend te zijn, zolang er gewerkt wordt met een realistisch aantal clients. Voor een toekomstige versie van het systeem zou echter de mogelijkheid bekeken kunnen worden om de client-afbeeldingen over een korte periode (één à twee seconden bijvoorbeeld) te laten vervagen tot ze verdwenen zijn. De twee mogelijke alternatieven zouden eventueel via een gebruikerstest geëvalueerd kunnen worden om de meeste gebruiksvriendelijke optie te achterhalen.

Figuur 22: Melding bij het wegvallen van een client

Wanneer het programma in de juiste modus zit kan met een klik op de client-afbeelding gezorgd worden dat het geselecteerde deel van de interface naar het overeenkomstige apparaat gedistribueerd wordt. (zie verder)

[41] http://www.usabilityfirst.com/glossary/term_105.txl



Figuur 23: Client-visualisatie in voorbeeldscenario

In het voorbeeldscenario ziet de gebruiker de verbonden clients gevisualiseerd als op de bovenstaande figuur. De afbeeldingen bevinden zich langs de rand en zijn klein genoeg om niet storend te werken. Ze bevinden zich echter dicht genoeg bij de te verdelen interface om gebruiksvriendelijke interactie mogelijk te maken. Wanneer een deel van de interface geselecteerd is kan de gebruiker interageren met het apparaat door op de client-afbeelding te klikken.

Figuur 24: Identificatie van de client via het label

Om de gebruiker de mogelijkheid te bieden de afbeeldingen uit het server-venster te kunnen identificeren en zo te zien welk apparaat ze vertegenwoordigen wordt er gebruik gemaakt van labels onderaan de afbeelding. Deze labels geven de naam weer die de gebruiker aan het apparaat gegeven heeft via de distributie-client. (zie verder)

5.4 Distributie van de User Interface

Het systeem dat gebruikt wordt om de gebruiker te laten selecteren welk deel van de interface hij wil distribueren is gebaseerd op het systeem van User Interface Façades[42] [31].

Dit systeem heeft origineel als doel om de gebruiker een simpele manier te geven waarop hij de interface van een willekeurig programma kan aanpassen naar zijn wensen om het gebruik van de beschikbare schermruimte te optimaliseren. Via directe manipulatie wordt de mogelijkheid geboden om stukken uit een interface te halen en als apart venster te gebruiken, stukken uit meerdere interfaces te combineren in één venster, en widgets binnen een venster te vervangen door andere types van widgets.

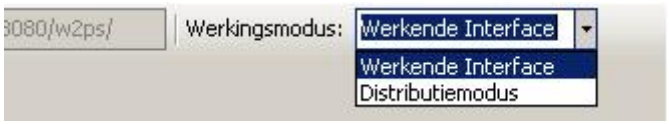
Om een deel van een interface te “extracten” en er een apart venster van te maken hoeft een gebruiker slechts het gewenste deel van de interface te selecteren met de muis door een rechthoek te trekken rond de widgets die hij wil. Daarna wordt deze selectie aangeklikt en buiten het venster gesleept. Daar wordt er automatisch een nieuw venster van gemaakt.

Figuur 25: Voorbeeld van selectie in User Interface Façades (Figuur overgenomen uit [31])

Dit concept werd overgenomen in het gerealiseerde systeem voor deze thesis omdat het een interessante manier biedt om de gebruiker op een zeer eenvoudige en natuurlijke manier te laten selecteren welk deel van de user interface hij wil distribueren.

Wanneer de server het UIML-document gerenderd heeft krijgt de gebruiker de werkende applicatie voor het bekijken van zijn foto’s te zien in het midden van het server-venster.

Wanneer hij dan de stukken van de interface wil aanduiden die hij graag zou verdelen naar zijn PDA en naar het grote scherm in zijn living, moet hij eerst het programma veranderen van modus. Hiervoor selecteert hij in de knoppenbalk bovenaan bij Werkingsmodus de optie “Distributiemodus”, zoals getoond in onderstaande figuur.

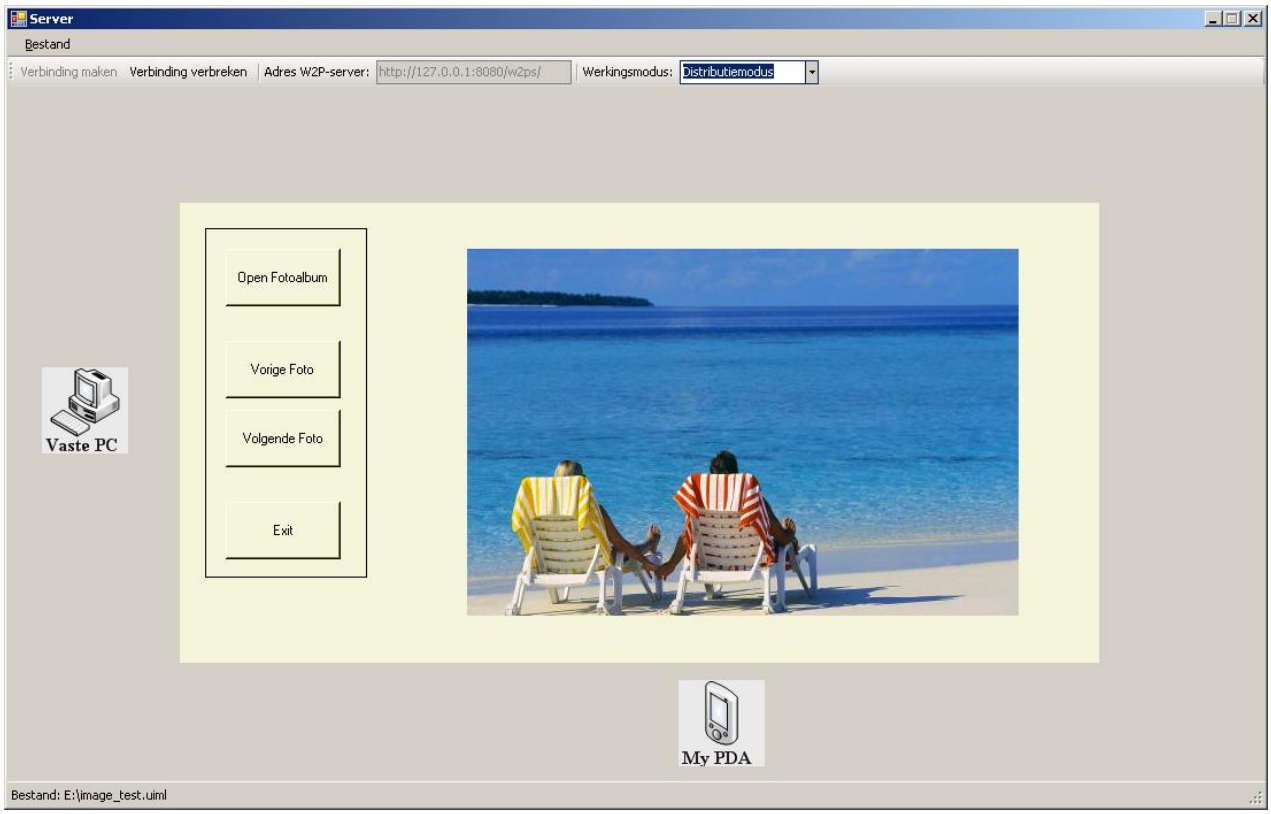


Figuur 26: Van modus veranderen om te distribueren

[42] http://www.cse.yorku.ca/~wolfgang/facades/

Daarop maakt het systeem een bitmap van de gerenderde interface. Deze afbeelding wordt dan in de plaats gezet van de werkende interface, in het midden van het server-venster. De cursor verandert ook in een crosshair wanneer hij boven de afbeelding komt, om aan te geven dat de gebruiker erop kan selecteren.

Nu kan de gebruiker een deel van de interface selecteren dat hij wil distribueren. Wanneer hij als eerste de knoppenbalk aan de linkerkant wil selecteren om de controls van het programma naar zijn PDA te distribueren klikt hij met zijn muis op de afbeelding en sleept een rechthoek rond deze knoppen, zoals te zien op de afbeelding.



Figuur 27: Selectie van interface-gedeelte voor distributie

Nadat de rechthoek getekend is hoeft de gebruiker slechts te klikken op de afbeelding aan de rand van het server-venster die zijn PDA voorstelt. Dan zal het systeem het distributieproces initiëren dat in het volgende stuk uitgelegd wordt. Het resultaat daarvan is dat de knoppen gedistribueerd worden naar zijn PDA.

5.5 Het distributieproces

Het eerste dat het systeem doet is controleren of er wel effectief een selectie gemaakt is op de afbeelding van de interface door middel van het tekenen van een rechthoek.

Indien er een selectie gemaakt is wordt er gezocht welke concrete widgets zich binnen deze selectie bevinden. Dit gebeurt door eerst een lijst aan te maken met de referenties van alle widgets die zich in het panel met de gerenderde interface bevinden. Deze lijst bevat dus alle widgets in de volledige hiërarchie en niet enkel de directe kinderen van het gerenderde panel.

Van elk van deze widgets wordt dan om de beurt de omvattende rechthoek (bounds) opgevraagd. Deze rechthoek bevat de locatie, de lengte en breedte van de control. Zo wordt de rechthoek voor elk widget gebruikt om te testen of deze binnen de rechthoek ligt die de gebruiker geselecteerd heeft op de afbeelding van de interface.

Indien de rechthoek erbinnen ligt, gaat het systeem op zoek naar het UIML-part dat bij dit concreet widget hoort. Dit wordt mogelijk gemaakt door een aanpassing die ik gedaan heb aan de UIML.Net rendering-engine. Ik heb gezorgd dat er binnen de renderer een hashtabel bijgehouden wordt waarin alle gerenderde UIML-parts zitten, met de referentie van het concrete widget als key.

Wanneer het systeem de verwijzing heeft naar het UIML-part wordt dit toegevoegd aan het structure-gedeelte van het UIML-document dat het systeem probeert op te bouwen om door te sturen naar de client.

Vervolgens wordt gekeken welke klasse dit part heeft. Op basis van die klasse wordt in de gebruikte vocabulary opgezocht welke properties er opvraagbaar zijn voor dit widget.

Dan wordt voor alle beschikbare properties die niets te maken hebben met de locatie van het part de waarde opgevraagd en deze worden in het juiste formaat toegevoegd aan het UIML-document dat wordt doorgestuurd. De locatie kan niet zomaar overgenomen worden voor het distribueren. Wanneer deze opgevraagd wordt zal het systeem de locatie geven, relatief ten opzichte van de parent van dit part. Waar we echter aan geïnteresseerd zijn is de locatie binnen de geselecteerde rechthoek. Hiervoor worden de locatie van het gerenderde widget voor het part en de geselecteerde rechthoek omgezet naar absolute coördinaten op het scherm. Het verschil tussen de X en Y coördinaten van beiden geeft dan de relatieve locatie van het concrete widget binnen de geselecteerde rechthoek. Deze locatie wordt meegegeven als property voor het juiste UIML-part toegevoegd aan het document dat doorgestuurd zal worden.

Bij deze aanpak wordt de grootte van de controls overgenomen en wordt uiteindelijk een user interface doorgestuurd die dezelfde afmetingen heeft als de rechthoek die de gebruiker geselecteerd heeft. Indien de gebruiker een gedeelte van de interface wil selecteren om door te sturen naar een PDA, dan geeft dit natuurlijk een probleem wanneer de geselecteerde rechthoek groter is dan het scherm van het doelapparaat. In deze thesis zal de interface gewoon op de PDA worden weergegeven op de aangegeven grootte, en zal er op het apparaat gebruik gemaakt worden van schuifbalken zodat de gebruiker toch de volledige interface kan gebruiken. Er wordt dus vanuit gegaan dat de gebruiker eigen inzicht gebruikt en aan zulke apparaten geen interface toewijst die veel te groot is om nog echt gebruiksvriendelijk te zijn. Deze verantwoordelijkheid volledig aan de gebruiker overlaten is natuurlijk geen ideale oplossing. Naar de toekomst toe zouden er andere oplossingen hiervoor geïmplementeerd kunnen worden zoals het weigeren van de distributie in dit geval, of het schaleren van de interface. De eerste oplossing zou dan gewoon een foutmelding geven wanneer de gebruiker een interface doorstuurt die groter is dan het scherm. Hiervoor hoeven we enkel de resolutie van het PDA-scherm te weten. De distributie gewoon weigeren is misschien echter een te restrictieve oplossing aangezien het kan voorvallen dat de gebruiker er het kleine ongemak van het scrollen bij moet nemen omdat hij op basis van de gebruikte apparaten geen andere keuze heeft. Een tweede mogelijk oplossing om het scrollen te voorkomen zou dan zijn de grootte van de gebruikte controls te verkleinen. Door de afmetingen van de interface naar beneden te schaleren zou deze wel volledig kunnen weergegeven worden. Het probleem met

deze oplossing is dan weer dat er grenzen zijn aan de mate waarin we de interface kunnen verkleinen met behoud van gebruiksvriendelijkheid. Voor een toekomstige versie van het systeem zouden we ons verder kunnen verdiepen in dit probleem en eventueel de voorgestelde oplossingen evalueren op basis van gebruikerstesten om te zien welke manier van werken de voorkeur heeft.

Wanneer het UIML-document om door te sturen is samengesteld, zal het systeem kijken op welke afbeelding de gebruiker juist geklikt heeft en de W2P-peer opvragen van de client via een hashtabel. Naar deze peer wordt het document dan verzonden via W2P in een message. De volgende listing geeft het zonet beschreven proces weer in pseudo-code.

```
IF selectie gemaakt
    Maak lijst van widgets in de interface;
    FOR EACH widget
        IF bounds van widget liggen binnen selectie
            Zoek UIML-part van widget;
            Voeg UIML-part toe aan UIML-document voor verdeling;
            Zoek klasse van UIML-part;
            Zoek de opvraagbare properties voor deze klasse;
            FOR EACH property
                IF property is niet locatie
                    Vraag waarde van property op voor huidig part;
                ELSE
                    Bereken waarde van locatie;
                END IF
                Voeg waarde toe aan UIML-document voor verdeling;
            END FOR EACH
        END IF
    END FOR EACH
    Identificeer doel-client;
    Stuur document door naar doel-client;
END IF
```

Listing 9: Pseudo-code van het distributie-algoritme

5.6 Functionaliteit van de gedistribueerde interface

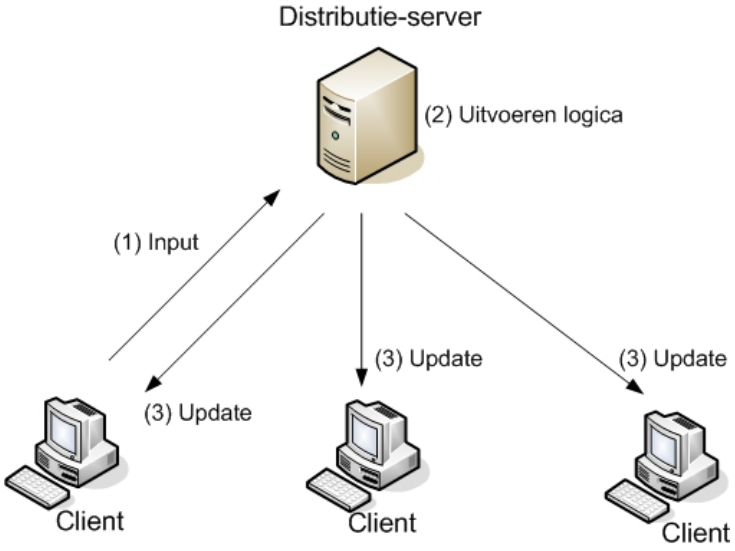
Om een nuttige gedistribueerde interface te verkrijgen is het uiteraard niet genoeg dat de delen ervan weergegeven worden op de verschillende apparaten. De interactieve functionaliteit van de interface zou ook behouden moeten blijven. In deze thesis is het niet gelukt om een gedistribueerde user interface volledig werkend te krijgen, maar er is wel reeds geëxperimenteerd met een aanpak die in de toekomst gehanteerd zou kunnen worden om dit mogelijk te maken. Er zit dus een beperkte "proof of concept"-implementatie in het gerealiseerde systeem.

Bij de gebruikte aanpak zal een client die een deel van de interface krijgt om te renderen het concrete widget van elk interface-deel opvragen. Dan wordt er voor dit widget gekeken of het van een type is waar de gebruiker input aan kan geven of niet. Een label bijvoorbeeld kan geen input krijgen van de gebruiker, maar een knop of een tekstvak wel. Wanneer het een type betreft dat input kan krijgen, zal het systeem één of meerdere event-handlers koppelen aan de events die er optreden voor dit widget bij interactie door de gebruiker. Voor een knop zal er bijvoorbeeld een event-handler gekoppeld worden aan het click-event. Op deze manier wordt de input van de gebruiker aan de client-zijde opgevangen. De event-handlers zullen vervolgens zoeken bij welk UIML-part het concrete widget hoort dat input gekregen heeft van de gebruiker. Dan wordt er een bericht gestuurd naar de distributie-server waarin staat welk UIML-part input gekregen heeft, wat voor type input, en indien nodig de waarde (zoals bijvoorbeeld input in een tekstvak). Wanneer de server dit bericht krijgt zal hij kijken welk concreet widget er in zijn kopie van de interface gelinkt is aan het bewuste part. Vervolgens wordt er een input-event gegenereerd op dit widget. Voor een klik op een knop wordt bijvoorbeeld gebruik gemaakt van de "PerformClick()"-method[43] uit de Button[44]-klasse. Door dit event te genereren wordt de achterliggende functionaliteit uitgevoerd die hieraan gekoppeld is. Met deze manier van werken is het dus mogelijk om vanuit de gedistribueerde delen van de interface input te geven aan de versie op de distributie-server. De volgende stap is dan zorgen dat de output die er in de interface getoond wordt als resultaat van deze input, ook doorgegeven wordt aan de gedistribueerde delen. Hiervoor kan er handig gebruik gemaakt wordt van het systeem van groepen in W2P. Wanneer de distributie-server een interface rendert zal hij alvorens er delen van die interface gedistribueerd worden de waarde voor elke property van elk part opvragen en bijhouden in een geneste hashtabel. Nadat een event gegenereerd werd voor de input die er binnenkwam, gaat de server een method uitvoeren die ook weer de waarde van elke property gaat opvragen voor elk part uit de interface. Wanneer één of meerdere waarden veranderd zijn als gevolg van het uitvoeren van de logica, zullen deze properties ook geüpdate moeten worden in de interface op de clients waar zich een exemplaar van het bewuste part bevindt. De server maakt hiervoor een bericht aan waar alle gewijzigde properties inzitten voor dit part en stuurt het naar een W2P-groep die als naam de naam van het part heeft. Dit veronderstelt dat de clients, wanneer ze een interface-deel krijgen om weer te geven, de namen van de parts hieruit gebruiken om zich te abboneren op de juiste groepen. Wanneer de server dan de nieuwe properties doorstuurt naar de groep zullen de clients deze properties binnenkrijgen en kunnen toepassen op hun gerenderde instantie van de interface via gebruik van de IPropertySetter-interface. In de huidige implementatie is er een beperkte voorziening voor input vanuit de gedistribueerde delen waarbij het klikken op knoppen en het invoeren van tekst in een tekstvak wordt doorgegeven naar de server. Voor het terugkoppelen van properties naar de clients toe zijn er

[43] http://msdn2.microsoft.com/en-us/library/system.windows.forms.button.performclick.aspx
[44] http://msdn2.microsoft.com/en-us/library/system.windows.forms.button.aspx

geen beperkingen. Wanneer als resultaat van een klik op een knop bijvoorbeeld een label gewijzigd wordt op de server, kan dit in principe teruggekoppeld worden naar de clients toe.

Samengevat komt het principe er dus op neer dat input van de gebruiker vanop de clients naar de distributie-server doorgestuurd wordt. Op de server wordt dan de achterliggende logica uitgevoerd, en de veranderingen die hierdoor in de user interface veroorzaakt worden, worden doorgestuurd naar de clients om hun weergave te updaten.

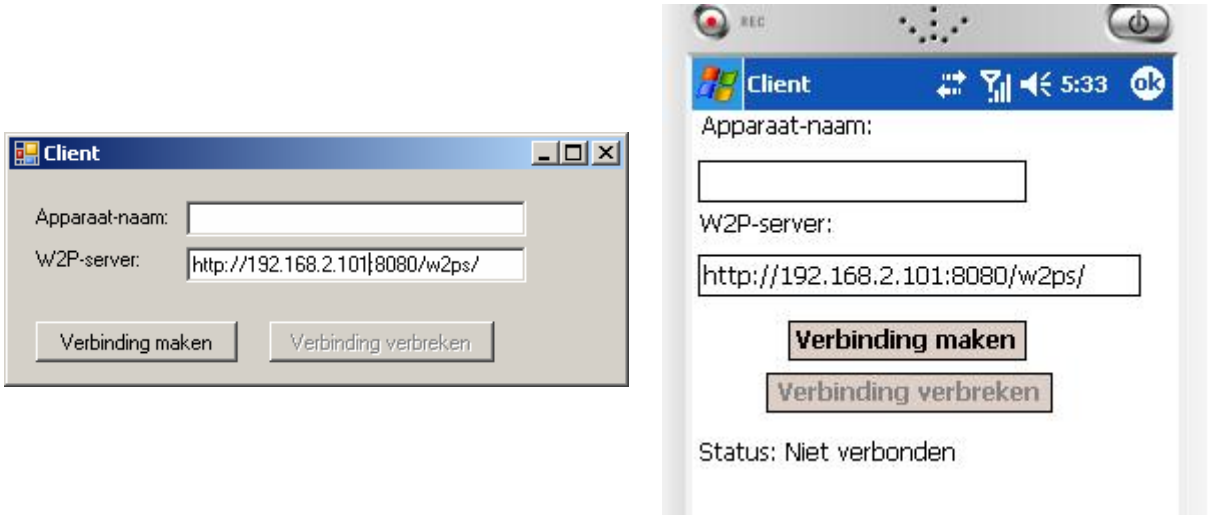


Figuur 28: Overzicht van de functionaliteit van de gedistribueerde interface

5.7 De client-software

De client-software wordt gedraaid op de apparaten die deel willen nemen aan de device federation en zich zo beschikbaar willen stellen om een deel van de interface te ontvangen. In ons voorbeeldscenario wordt deze software dus gedraaid op de PDA en de set-top box van de televisie.

Van het client-programma bestaat zowel een versie voor het gewone .Net framework als voor het .Net Compact Framework. Het is dus mogelijk om een deel van de interface te laten weergeven op elk apparaat dat één van deze twee frameworks ondersteunt en de mogelijkheid heeft tot communicatie over TCP/IP.



Figuur 29: De clients voor het gewone .Net framework (links) en .Net CF (rechts)

Het enige wat de gebruiker nog moet doen is het ingeven van de url waar de W2P communicatieservice zich bevindt, en een naam waarmee we het apparaat uniek kunnen identificeren binnen deze sessie van het gebruik van ons systeem. Deze naam zal op de server weergegeven worden zoals eerder uitgelegd.

Vervolgens wordt er op “Connect” geklikt. Er wordt in het venster van de distributie-server een afbeelding getoond voor de client en vanaf dan is deze beschikbaar voor binnenkomende verzoeken om een deel van de interface weer te geven tot op “Disconnect” geklikt wordt, het programma gesloten wordt of door een technisch probleem de verbinding wegvalt.

Wanneer de client een UIML-document met een beschrijving van (een deel van) de interface doorgestuurd krijgt, zal dit gerenderd worden via UIML.Net en gewoon weergegeven worden. Wanneer er een nieuw document binnenkomt om weer te geven, zal eerst het vorige worden afgesloten.

5.8 De communicatie-server (Web to Peer)

De W2P-server is de centrale communicatie-hub in het ontwikkelde systeem. Zowel de server als de clients moeten zich bij deze server aanmelden aangezien alle communicatie in het systeem via W2P verloopt. De W2P-server laat de apparaten binnen de federation toe om op

een uniforme manier met elkaar te kunnen communiceren, ongeacht het type apparaat of fysieke locatie.

Om de server te kunnen runnen moet er een servlet container beschikbaar zijn. Binnen deze thesis wordt er gebruik gemaakt van Jetty[45], maar deze kan evengoed vervangen worden door bijvoorbeeld Apache Tomcat[46], of JBoss[47].

Het service-adres waarop deze dienst beschikbaar is moet op voorhand gekend zijn door de distributie-server en alle clients die willen deelnemen aan de federation. Gebruik van het systeem zou waarschijnlijk vlotter en transparanter zijn wanneer de apparaten de communicatie-server zelf zouden kunnen ontdekken via een broadcast-mechanisme of iets dergelijks. Dit zou niet al te moeilijk moeten zijn om toe te voegen, maar in deze thesis ligt de focus vooral op de verdeling van de interface.

Eens de service gestart is en het service-adres ervan gekend is hoeft hier in principe niet meer naar omgekeken te worden. Er zijn geen verdere configuratiestappen meer nodig en het systeem zal gewoon beschikbaar blijven zolang de server “up and running” is. In het voorbeeldscenario uit deze thesis wordt ervan uitgegaan dat de W2P-communicatie-server in de omgeving draait zonder dat de gebruiker hier nog expliciet iets voor hoeft te doen en dat het adres van deze service gekend is door de gebruiker.

Dit maakt wel dat we met deze server een “single point of failure” hebben in het systeem. Wanneer deze server crasht of door hardwareproblemen onbereikbaar wordt, is dit een groot probleem. Een mogelijke oplossing hiervoor zou kunnen zijn dat wanneer de andere apparaten (distributie-server en clients) detecteren dat hun verbinding met de W2P-server is weggevallen ze via een systeem van broadcasting erachter proberen te komen welke apparaten zich allemaal in de device federation bevinden. Op basis van deze informatie zou er dan via een election-algoritme een vervanger aangeduid kunnen worden die de taak van de W2P-server overneemt.

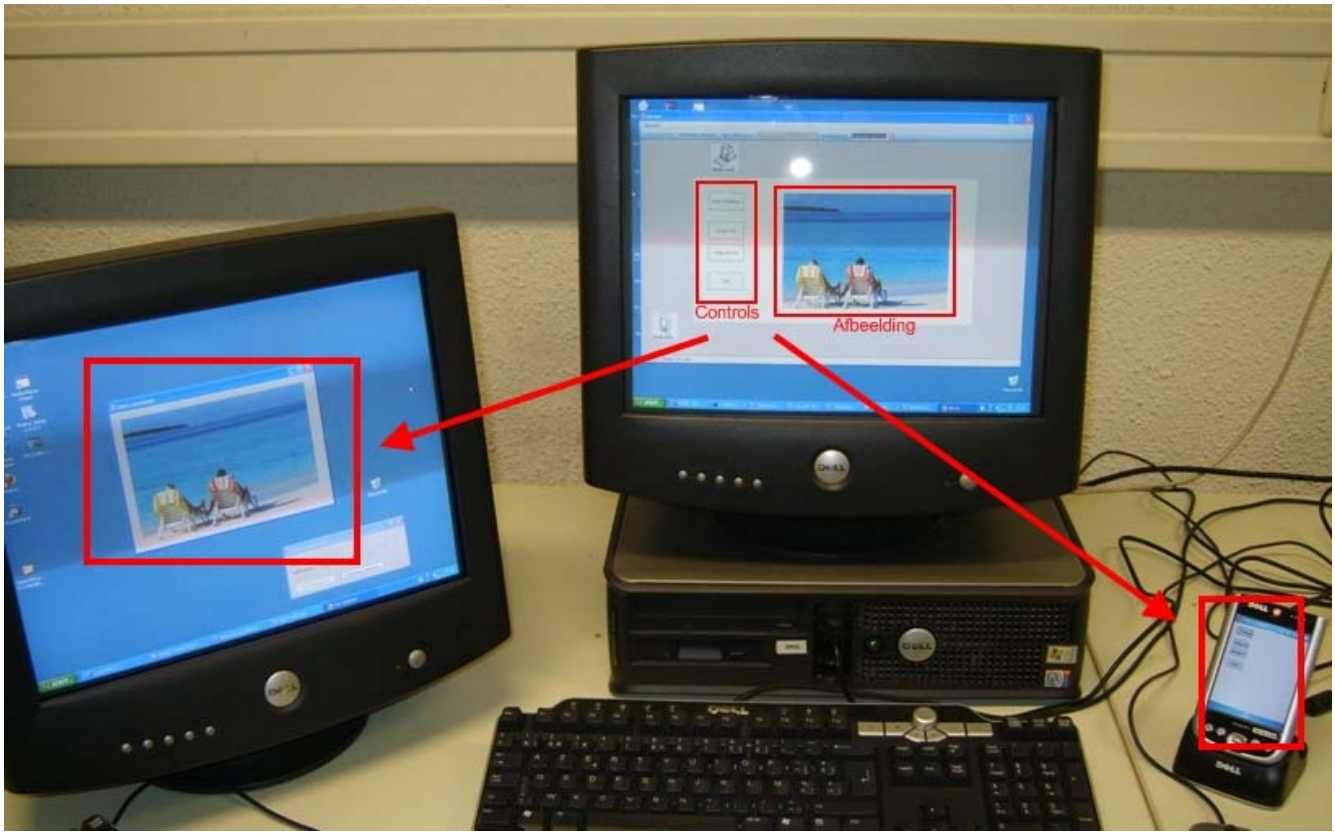
5.9 Conclusie

In dit hoofdstuk werd de software toegelicht die ontwikkeld werd om de probleemstelling uit deze thesis aan te pakken. De verschillende delen van het systeem werden besproken en designbeslissingen werden toegelicht. We hebben gezien hoe de gebruiker kan aangeven welk deel van de user interface naar welk apparaat gedistribueerd moet worden en hoe deze verdeling plaatsvindt.

[45] http://jetty.mortbay.org/
[46] http://tomcat.apache.org/
[47] http://www.jboss.org/products/jbossas

6 Discussie

6.1 Resultaat van deze thesis



Figuur 30: Een proefopstelling van het gerealiseerde systeem

Het systeem dat we in deze thesis hebben ontwikkeld probeert de gebruiker op een gebruiksvriendelijke manier te laten aangeven welke deel van een user interface hij op een bepaald apparaat wil laten weergeven. Het systeem probeert die verdeling dan uit te voeren op een voor de gebruiker transparante manier.

Een gebruiker kan een bepaald deel van de user interface ook gemakkelijk naar meerdere apparaten distribueren om zodanig op verschillende apparaten toegang te hebben tot dezelfde informatie, of input te kunnen geven. Er zijn geen speciale locking-mechanismen voorzien voor het bewaren van state-consistency aangezien we er in deze thesis niet van zijn uitgegaan dat er meerdere gebruikers deelnemen aan de interaction space. Mits de implementatie van een dergelijk systeem kan er gemakkelijk een collaborative interaction space worden opgezet met ons systeem zodra het laten functioneren van de gedistribueerde interface volledig gerealiseerd is.

De server geeft het overzicht van de verbonden clients vrij duidelijk weer door de afbeeldingen langs de rand. Deze afbeeldingen zijn klein genoeg om niet storend te zijn, maar toch groot genoeg om interactie toe te laten. Door de cursor te laten veranderen wordt er ook een hint gegeven naar het feit dat interactie mogelijk is. Dit systeem van widgets langs de

rand van het scherm plaatsen is in de context van het RelateGateways systeem reeds een eerste maal geëvalueerd met een kleine groep gebruikers [33]. De algemene indruk was dat dit systeem goed ontvangen werd door de gebruikers, waarbij er gesuggereerd werd dat de widgets voor de apparaten te onduidelijk waren omdat er meerdere grafische afbeeldingen op elk widget stonden. Dit werd in deze thesis vermeden doordat de widgets voor de clients slechts één afbeelding bevatten van voldoende grootte. Ook werd in het onderzoek door de gebruikers gesuggereerd dat de widgets dynamischer moesten zijn en groter zouden moeten worden wanneer de cursor ze nadert om zo de interactie te vergemakkelijken.

De manier om delen van de interface te selecteren door middel van een selectiekader is een directe manipulatie-techniek die de gebruiker op een simpele manier zijn keuze kenbaar laat maken. Deze techniek is eenvoudig genoeg en het is ook een standaardmanier van selecteren in andere veelgebruikte programma's, zoals bijvoorbeeld de windows explorer. Daardoor verwachten we dat weinig mensen problemen zullen hebben met deze manier van werken.

In deze thesis wordt een systeem voorgesteld om de user interface te kunnen distribueren naar meerdere clients. Enkel het distributiesysteem dat ontwikkeld werd kan dan uiteraard beoordeeld worden in het kader van deze thesis, en niet de gebruiksvriendelijkheid of het gemak waarmee de gebruiker uiteindelijk zal kunnen werken met de gedistribueerde interface die voortvloeit uit het gebruik van ons systeem. Het feit dat er met een gedistribueerde interface gewerkt wordt zorgt op zich voor een aantal usability issues zoals eerder aangehaald [15]. De gebruiksvriendelijkheid van de gedistribueerde interface op zich zal dus sterk afhankelijk zijn van de situatie, en van de gebruiker aangezien hij ook degene is die zelf beslist over de verdeling van de interface.

6.1.1 Beoordeling volgens het 4C referentiemodel

Zoals reeds eerder in deze thesis aangehaald kan een systeem voor gedistribueerde user interfaces besproken worden aan de hand van het 4C referentiemodel waarbij een systeem in vier dimensies bekeken wordt (Computation, Communication, Coordination & Configuration).
Het systeem dat voor deze thesis gerealiseerd werd kan dan in de vier dimensies als volgt bekeken worden:

Computation:

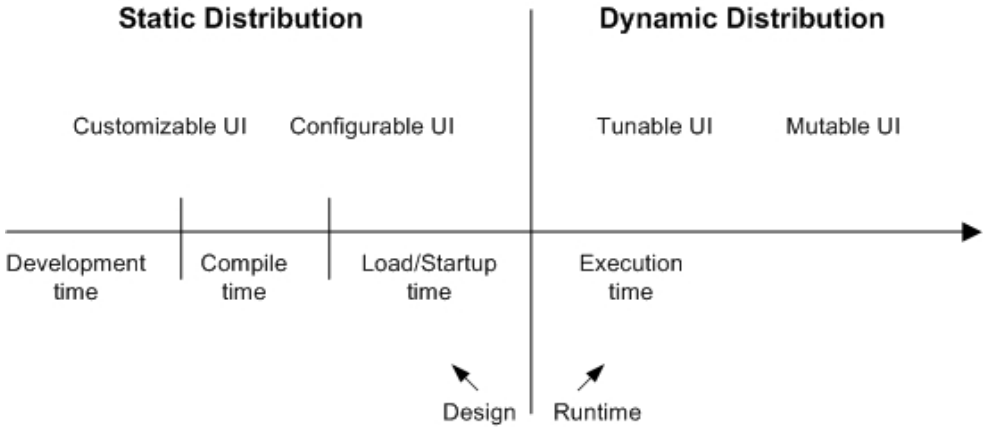
Deze dimensie bekijkt welke elementen er gedistribueerd worden. In de eerste plaats kunnen we al zeggen dat in ons systeem enkel gewerkt wordt met distributie van de user interface. Alle logica die er in de user interface of de achterliggende applicatie steekt, blijft op de server.

De elementen die we kunnen distribueren zijn delen van de abstracte user interface. Op de server wordt een selectie gemaakt uit een concrete instantie van de user interface. Van de delen die hier geselecteerd zijn gaan we de abstracte beschrijvingen ophalen (UIML-parts) en deze worden dan als UIML-document doorgestuurd naar een client, waar ze via rendering weer omgezet worden in een concrete user interface.
Het deel dat we distribueren kan zowel één enkel interface-element zijn als een groep interface-elementen, of zelfs de volledige interface. We kunnen dezelfde interface-elementen ook naar meerdere clients tegelijk distribueren in dit systeem. Er is geen enkele beperking in

ons systeem die zegt dat bepaalde containers niet meer verder opgedeeld kunnen worden. We kunnen zonder problemen splitsen tot op het niveau van een enkelvoudig UI-element.

Communication:

De volgende dimensie is een tijdsdimensie die bespreekt wanneer de distributie plaatsvindt. De twee grote onderverdelingen die hier gemaakt kunnen worden zijn statische distributie en dynamische distributie. Bij statische distributie wordt de verdeling berekend bij het ontwikkelen, compileren of laden. Dynamische distributie vindt plaats tijdens het runnen van de applicatie.



Figuur 18: Tijdslijn voor distributie van een UI (Figuur geïnspireerd door [15])

Bij dynamische distributie hebben we nog twee onderverdelingen. Een user interface wordt beschouwd als “tunable” wanneer ze tijdens het runnen wel aangepast kan worden, maar de onderliggende code niet veranderd kan worden. Bij een “mutable” user interface daarentegen wordt er zo ver gegaan dat zelfs de onderliggende code verandert kan worden. Zo wordt een decompositie van de interface-elementen mogelijk. Ons systeem bevindt zich helemaal aan de rechterkant van de bovenstaande tijdslijn, omdat delen van de achterliggende AUI opgevraagd worden, op de gewenste manier opnieuw samengesteld worden in een nieuwe deelinterface en op hun bestemming opnieuw gerenderd worden.

Coordination:

Hier kijken we wie verantwoordelijk is voor de distributie van het systeem. Bij ons gebeurt de distributie vanop de distributie-server door middel van een interface die enkel dient voor het beheren van de distributie. In het kader van het 4C referentiemodel wordt dit de meta-ui genoemd. Binnen het distributieproces zijn er een aantal verschillende operaties te onderscheiden, namelijk:

- Detectie van het feit dat distributie wenselijk/noodzakelijk is
- Berekening van de mogelijke alternatieven voor distributie
- Selectie van een geschikt alternatief
- Het effectief uitvoeren van de distributie

Al deze operaties kunnen beschouwd worden als de verantwoordelijkheid van ofwel de gebruiker ofwel het systeem. In ons systeem is het de gebruiker die beslist dat er delen van de interface gedistribueerd moeten worden. Hij kijkt dan zelf wat de mogelijkheden zijn, kiest er

hier een uit en geeft dit in via de interface van de server. Daarna is het de server die de distributie uitvoert.

Configuration:

De laatste dimensie kijkt waar de delen naartoe gedistribueerd worden. In ons systeem worden de delen gedistribueerd naar een willekeurig aantal heterogene clients die van twee verschillende types kunnen zijn: een client die beschikt over het gewone .Net framework of een client met het .Net compact framework.
Ook wordt er in deze dimensie gekeken naar de wijze waarop een interface gedistribueerd wordt. Deze kan zonder meer overgezet worden, of aangepast aan het doelplatform. In ons systeem wordt de interface in zeker mate aangepast aan het doelplatform omdat de abstracte user interface gerenderd wordt naar een concrete interface in de gebruikte widget-set van het doelplatform. Zo kunnen we de variaties aanpakken die er bestaan tussen de gewone SWF-toolkit en SWF op het .Net compact framework. Wanneer we compleet andere toolkits willen ondersteunen in de toekomst, zoals bijvoorbeeld GTK#, dan zullen we het UIML-bestand dat doorgestuurd wordt naar de clients nog verder moeten aanpassen aangezien de properties die door verschillende toolkits gebruikt worden ook kunnen verschillen.

6.2 Toekomstig werk

Het eerste punt waar in de toekomst aan gewerkt zou moeten worden in het systeem is de volledige functionaliteit van de gedistribueerde interface. Op dit moment zit er enkel een “proof of concept”-implementatie in het systeem die beperkte functionaliteit toelaat. Het nadeel van de huidige manier van implementeren is dat er weinig abstractie gebruikt wordt. Wanneer een client een deel van de interface krijgt zal er gekeken worden naar het concrete widget om te zien of dit een type is dat input kan ontvangen van de gebruiker. Daarop zal een event-handler gehangen worden aan de events die de gebruiker kan veroorzaken bij dit widget. Dit is momenteel specifiek gecodeerd voor de SWF-toolkit. In de toekomst is het de bedoeling om deze informatie uit de gebruikte vocabulary te halen, waardoor evolutie in de SWF-toolkit en eventueel gebruik van andere toolkits beter ondersteund wordt. Verder zal er ook rekening mee moeten worden gehouden dat resources die aangesproken worden eventueel meegestuurd moeten worden naar de client alvorens deze een interface-deel kan weergeven. Dit is bijvoorbeeld het geval wanneer een interface een afbeelding toont die lokaal op de distributie-server is opgeslagen, zoals in ons voorbeeldscenario. Wanneer de gebruiker daarin naar een volgende afbeelding navigeert zal deze beschikbaar gemaakt moeten worden op de client die de afbeelding moet weergeven.

Een volgend belangrijk punt aan het systeem dat naar de toekomst toe verbeterd zou kunnen worden is het feit dat de distributie-server uit ons systeem op dit moment enkel geschikt is voor een volwaardige desktop pc of laptop. De server is niet erg geschikt om te draaien op apparaten zoals bijvoorbeeld de PDA uit ons voorbeeldscenario. Hiervoor zijn verschillende redenen. Ten eerste is de server belast met het uitvoeren van een aantal berekeningen die het systeem op de meeste PDA’s waarschijnlijk te fel zouden vertragen. Dit probleem zou tot op zekere hoogte wel aangepakt kunnen worden met een aantal optimalisaties van de code en de gebruikte berekeningen. Een groter probleem bij gebruik op een PDA zou echter de beperkte schermruimte zijn, in combinatie met de huidige manier waarop de clients en de selectie van interface-delen gevisualiseerd wordt. De server moet op dit moment de volledige interface kunnen tonen die verdeeld wordt, en hier rondom nog afbeeldingen van de beschikbare

clients. Om de distributie-server ook op een PDA te kunnen laten werken zou er dus vooral gezocht moeten worden naar gebruiksvriendelijk alternatieve visualisaties voor het verdelen van de interface en het weergeven van de clients op een klein scherm. Het is in principe ook mogelijk om de user interface gewoon naar beneden te schaleren, maar deze aanpak garandeert niet dat de bruikbaarheid van de interface behouden blijft.
Met de mogelijkheid de distributie-server op een PDA te draaien zou het voor de gebruiker uit ons voorbeeldscenario nog makkelijker zijn. Hij zou dan eventueel gewoon het systeem vanuit de zetel hierop kunnen starten, vervolgens het gedeelte van de user interface dat de foto toont distribueren naar het grote scherm, en de controls op zijn PDA laten.

Een ander vrij groot nadeel dat in toekomstig werk aangepakt zou kunnen worden is het feit dat het systeem op dit moment afhankelijk is van een vaste, op voorhand geselecteerde distributie-server. Tijdens de werking van het systeem mag deze server dus eigenlijk niet wegvallen. Het zou nuttig zijn om het systeem uit te breiden zodat bij het wegvallen van de server deze functionaliteit overgedragen kan worden naar een ander apparaat, eventueel automatisch door gebruik van een election-algoritme gebruikt wordt om een vervanger aan te duiden. Wanneer in het voorbeeldscenario de distributie-server dan zou crashen of er zou zich een netwerkprobleem voordoen, dan zou er gewoon een andere distributie-server aangeduid kunnen worden.

Verder zouden we ook het systeem kunnen uitbreiden door in de toekomst een echt sensorsysteem te implementeren zoals dat in het systeem van RelateGateways werd voorgesteld. Dat zou het de gebruiker nog gemakkelijker maken om zich te oriënteren en in de interface te zien welke afbeelding welk apparaat representeert.

Daarnaast hebben we in ons huidig systeem de beperking dat er enkel gebruik gemaakt kan worden van de SWF-toolkit. In een toekomstige versie zouden we ook ondersteuning kunnen toevoegen voor andere toolkits zoals GTK# bijvoorbeeld. Eerder in deze thesis hebben we vermeld dat in de praktijk er vaak toch platformspecifieke informatie in een UIML-document zal zitten omdat er bijvoorbeeld in GTK# gewerkt wordt met lay-out managers en in SWF niet. Als we meerdere toolkits gaan ondersteunen moeten we een systeem hebben waarbij onze abstracte UIML-beschrijving in een correcte platformspecifieke beschrijving omgezet wordt voor het renderen. Een mooie oplossing zou kunnen zijn om gebruik te maken van het eerder in deze thesis aangehaalde Device-Independent UIML om onze hoofdinterface te beschrijven met platformonafhankelijke constraints die dan net voor het renderen omgezet worden naar een lay-out voor een specifiek platform.

Een grote uitdaging voor toekomstig onderzoek zou ook zijn om het distributieproces verder te automatiseren. In onze implementatie moet de gebruiker nog volledig zelf selecteren welke delen van de interface hij naar welk apparaat wil distribueren. Volledige automatisatie is waarschijnlijk niet erg realistisch omwille van het feit dat een optimale distributie zeer fel afhangt van persoonlijke mening, zoals eerder reeds aangehaald in deze thesis, maar in de toekomst zou de computer op basis van de kenmerken van de interface en van de deelnemende apparaten eventueel suggesties kunnen doen naar de gebruiker toe. Bij het geven van deze suggesties kan er dan bijvoorbeeld rekening gehouden worden met het feit of de geselecteerde interface niet te groot is voor het scherm van het doelapparaat.

We zouden eventueel ook nog kunnen zorgen dat de apparaten binnen het systeem zelf de W2P communicatie-server kunnen vinden, via een systeem van broadcasting of iets dergelijks, zodat het adres van de server niet meer met de hand hoeft ingegeven. Zoals eerder

vermeld zouden we ook kunnen voorzien dat een ander apparaat deze functie kan overnemen indien de W2P-server wegvalt.

Nog een aanpassing die we zouden kunnen suggereren naar de toekomst toe zou misschien zijn om in te spelen op een opmerking die gegeven werd bij de evaluatie van het RelateGateways systeem. Daarin werd gesuggereerd dat wanneer de cursor een afbeelding nadert die een apparaat representeert, deze afbeelding dynamisch van grootte zou kunnen veranderen om de interactie iets makkelijker te maken. We zouden in de toekomst kunnen nagaan of een dergelijke uitbreiding ook positieve effecten zou hebben op onze distributie-server.

Als laatste zouden we, zoals eerder aangehaald, via gebruikerstesten kunnen evalueren of we bij het afmelden van een client het beste kunnen werken met een melding in een messagebox, of het laten vervagen van de client-afbeelding over een korte periode.

7 Conclusie

We hebben in deze thesis onderzoek gedaan naar het concept gedistribueerde gebruikersinterfaces. Wat we vooral onthouden van deze thesis is dat er op dit moment in de bestaande literatuur opvallend weinig gepubliceerd is over het ontwikkelen van gedistribueerde gebruikersinterfaces. Dit toont ons dat ondanks het aanzienlijke potentieel van dit concept in de huidige evolutie van ubiquitous computing, het als onderzoeksgebied nog vrij nieuw is.

We hebben in deze thesis dan ook getracht een manier te tonen om op een simpele en gebruiksvriendelijke wijze een verdeling uit te voeren van een interface. De interface in kwestie hoeft hierbij enkel een beschrijving te hebben die opgesteld is in UIML. Voor de rest zijn er van de ontwikkelaar uit in principe geen extra vereisten om de interface distribueerbaar te maken.

We hopen dat het gerealiseerde systeem heeft aangetoond hoe een user interface op een gebruiksvriendelijke manier gedistribueerd kan worden, en hopen dat het kan bijdragen aan toekomstig werk in dit opkomende onderzoeksgebied.

Bibliografie

[1] *Web User Interface Migration through Different Modalities with Dynamic Device Discovery* (Renata Bandelloni, Giulio Mori, Fabio Paternò, Carmen Santoro & Antonio Scorcia) AEWSE'07: 2nd International Workshop on Adaptation and Evolution in Web Systems Engineering, Como, Italy, July 19, 2007

[2] *CAMELEON-RT: a Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces* (Lionel Balme, Alexandre Demeure, Nicolas Barralon, Joëlle Coutaz & Gaëlle Calvary) Proceedings EUSAI '04, LNCS 3295, Springer-Verlag, 2004, 291-302.

[3] *Flexible Interface Migration* (Renata Bandelloni & Fabio Paternò)
In Proceedings of Intelligent User Interface, 2004

[4] *Platform Awareness in Dynamic Web User Interfaces Migration* (Renata Bandelloni & Fabio Paternò) Proceedings Mobile HCI 2003, LNCS 2795, Springer Verlag, 2003, pp.440--445

[5] *The Impact of Migration of Data to Small Screens on Navigation* (Bonnie MacKay & Carolyn Watters) IT&Society, Volume 1 Issue 3 Winter 2003, Pages 90-101

[6] *Browser Session Preservation and Migration* (Henry Song, Hao-hua Chu & Shoji Kurakake) Poster Session of WWW 2002, Hawaii, USA. 7-11. May, 2002. pp. 2.

[7] *Tool Support for Designing Nomadic Applications* (Giulio Mori, Fabio Paternò & Carmen Santoro) Proceedings of 7 Int. Conf. on Intelligent User Interfaces IUI'03 (January 12-15, 2003)

[8] *The Computer for the 21st Century* (Mark Weiser) Scientific American, vol 265, number 3, pp 94-104 (1991)

[9] *Distributed User Interface Elements to support Smart Interaction Spaces* (Kris Luyten & Karin Coninx) IEEE Symposium on multimedia, Irvine, California, USA, December 12-14, 2005

[10] *Light-weight Distributed Web Interfaces Preparing the Web for Heterogeneous Environments* (Chris Vandervelpen, Geert Vanderhulst, Kris Luyten & Karin Coninx)
The 5th International Conference on Web Engineering (ICWE'2005), Sydney, Australia, July 25-29, 2005

[11] *Dialog Model Clustering for User Interface Adaptation* (G. Menkhaus & S. Fischmeister) In Web Engineering, Proceedings of ICWE 03, 2003

[12] *Augmented Surfaces: A Spatially Continuous Work Space for Hybrid Computing Environments* (Jun Rekimoto & Masanori Saitoh) CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 378-385, 1999

[13] *Using Handhelds and PCs Together* (Brad A. Myers) Communications of ACM, Vol. 44, No. 11. (2001), pp. 34-41.

[14] *Multiple-Computer User Interfaces: A Cooperative Environment Consisting of Multiple Digital Devices* (Jun Rekimoto) Lecture Notes in Computer Science ,volume 1370, 1998

[15] *The 4C Reference Model for Distributed User Interfaces* (Alexandre Demeure, Jean-Sébastien Sottet, Gaëlle Calvary, Joëlle Coutaz, Vincent Ganneau & Jean Vanderdonckt) ICAS 2008, 26th Congress of the International Council of Aeronautical Sciences, Alaska, USA 14-19 September 2008

[16] *Four Easy Pieces for Assessing the Usability of Multimodal Interaction : The CARE Properties* (Joëlle Coutaz, Laurence Nigay, Daniel Salber, Ann Blandford, Jon May & Richard M. Young) In InterAct, pages 115--120, 1995

[17] *One Model, Many Interfaces* (Fabio Paternò & Carmen Santoro) CADUI'2002 - 4th International Conference on Computer-Aided Design of User Interfaces, May 15-17, 2002

[18] *Applying Model-based Techniques to the Development of UIs for Mobile Computers* (Jacob Eisenstein, Jean Vanderdonckt & Angel Puerta) Intelligent User Interfaces, pp 69-76, 2001

[19] *An XML-based Runtime User Interface Description Language for Mobile Computing Devices* (Kris Luyten & Karin Coninx) DSV-IS'2001, Glasgow (Scotland), 2001

[20] *Profile-aware Multi-device Interfaces: An MPEG-21-based Approach for Accessible User Interfaces* (Kris Luyten, Kristof Thys & Karin Coninx) British Computer Society; Workshops in Computing (eWIC) Series Dundee, Scotland, August 23-25, 2005

[21] *A Review of XML-compliant User Interface Description Languages* (Nathalie Souchon & Jean Vanderdonckt),pp 377-391 Interactive System: Design, Specification, and Verification, 10th International Workshop, DSV-IS 2003, Funchal, Madeira Island, Portugal, June 11-13, 2003

[22] *UIML.Net: An Open UIML Renderer for the .Net Framework* (Kris Luyten & Karin Coninx) CADUI'2004, Funchal, Madeira Island (Portugal), 2004

[23] *A Generic Approach for Multi-device User Interface Rendering With UIML* (Kris Luyten, Kristof Thys, Jo Vermeulen & Karin Coninx) 6th International Conference on Computer-Aided Design of User Interfaces (CADUI'2006)

[24] *User Interface Markup Language (UIML) specification*, language version 3.0 (Marc Abrams & Jim Helms), Harmonia Inc., 2002

[25] *UIML: An Appliance-independent XML User Interface Language*, Ph. D. Dissertation, Virginia Polytechnic Institute and State University (Constantinos Phanouriou), 2000

[26] *A Graphical Design Tool for Multi-Device User Interfaces based on UIML* (Jan Meskens), Masters thesis, Uhasselt, 2007

[27] *Simplifying Construction of Multi-platform User Interfaces Using UIML* (Mir Farooq Ali, Marc Abrams) Proceedings of the User Interface Markup Language Conference, Paris, France, March 2001

[28] *Constraint Adaptability of Multi-Device User Interfaces* (Kris Luyten, Jo Vermeulen & Karin Coninx) Workshop on The Many Faces on Consistency, CHI'2006 workshop, Montreal, Quebec, Canada, April 22-23, 2006

[29] *Multi-device Layout Management for Mobile Computing Devices* (Kris Luyten, Bert Creemers & Karin Coninx) Technical Report TR-LUC-EDM-0301, 2003, EDM/LUC, Diepenbeek, Belgium

[30] *Middleware for Ubiquitous Service-oriented Spaces on the Web* (Geert Vanderhulst, Kris Luyten & Karin Coninx) International Workshop on Network-based Virtual Reality and Tele-existence (INVITE'2007) in conjunction with the 21st IEEE International Conference on Advanced Information Networking and Applications (AINA2007) Niagara Falls, Ontario, Canada, May 21 - 23, 2007

[31] *User Interface Façades: Towards Fully Adaptable User Interfaces* (Wolfgang Stuerzlinger, Olivier Chapuis, Dusty Phillips & Nicolas Roussel) In Proceedings of UIST'06, the 19th ACM Symposium on User Interface Software and Technology, pages 309-318, October 2006. ACM Press.

[32] *RelateGateways: Using Spatial Context to Identify and Interact with Pervasive Services* (Carl Fischer, Hans Gellersen, Dominique Guinard, Matt Oppenheim & Sara Streng) 9th International Conference on Ubiquitous Computing (UbiComp 2007), Austria, September 2007

[33] *RelateGateways: A User Interface for Spontaneous Mobile Interaction with Pervasive Services* (Dominique Guinard, Sara Streng & Hans Gellersen) In Mobile Spatial Interaction in conjunction with ACM International Conference on Human Factors in Computing Systems, San Jose CA, USA, CHI 2007

[34] *Extending Mobile Devices with Spatially Arranged Gateways to Pervasive Services* (Dominique Guinard, Sara Streng & Hans Gellersen) 3rd International Workshop on Pervasive Mobile Interaction Devices (PERMID 2007)