A User Interface Meta Data Layer to Support Intelligent Adaptation

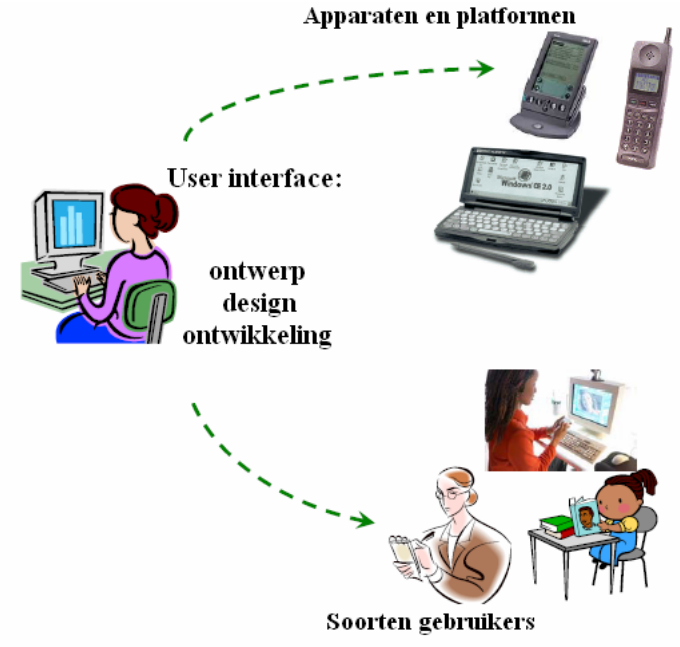
Tijl Lathouwers

promotor :
Prof. dr. Kris LUYTEN

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

universiteit hasselt

A USER INTERFACE META-DATA LAYER TO SUPPORT INTELLIGENT ADAPTATION



Master thesis Informatica – optie HCI
transnationale Universiteit Limburg - Diepenbeek
Tijl Lathouwers
0422492

Promotor: Kris Luyten
Co-promotor: Karin Coninx
Begeleiders: Carl Bruninx en Jo Vermeulen
Universiteit Hasselt
Juni 2008

Voorwoord

Dit eindwerk heb ik geschreven binnen de opdracht van mijn master thesis; hierin beschrijf ik uitgebreid hoe een user interface meta-data laag intelligente adaptatie ondersteunt.

Ik zou graag mijn begeleiders, promotor, vrienden en familie bedanken, die me geholpen, gesteund of begeleid hebben bij het schrijven van mijn thesis.

Inhoudstabel

Inleiding 7
1 Wat is adaptatie? 8
1.1 Adaptatie in de informatica 8
1.2 Waarom is adaptatie nodig? 8
1.3 Waarom is adaptatie meer nodig dan vroeger? 9
1.4 Adaptatie in user interfaces 9
2 Analyse 10
2.1 Formele definiëring van het begrip adaptatie 10
2.1.1 Doel van adaptatie 10
2.1.2 Wanneer is adaptatie juist uitgevoerd? 11
2.2 Analytisch onderzoek van adaptatie in user interfaces 11
2.2.1 Voorwoord: waarom een HLUIDL gebruiken 11
2.2.2 UIML 12
2.2.2.1 UIML en adaptatie 12
2.2.2.2 Welke onderdelen adapteren 12
2.2.3 Andere HLUIDs en frameworks 15
2.2.3.1 XIML 15
2.2.3.2 ISML 16
2.2.3.3 Seescoa XML 18
2.2.3.4 Teresa (Cameleon) 19
2.2.4 Conclusie 21
3 Vereisten voor intelligente en kwalitatieve adaptatie 22
3.1 Probleemsituering 22
3.2 Integratie van de context (door metadata) 23
3.2.1 Waarvoor wordt metadata gebruikt 23
3.2.2 Voorstelling van metadata 25
3.2.2.1 XML 25
3.2.2.2 RDF 25
3.2.3 Toepassing van metadata 28
3.2.3.1 MPEG-21 28
3.2.3.2 FOAF 29
3.2.3.3 GUMO 30
3.2.3.4 Andere soorten 31
3.2.4 Conclusie 31
3.3 Redeneren vanuit de context 31
3.3.1 Welke contextinformatie gebruiken 32
3.3.2 Context modelleringsregels 32
3.3.3 Adaptatieregels 33
3.3.4 Waar en wanneer contextinformatie gebruiken 33
3.3.5 Redeneringsmechanismen 34
3.3.6 Conclusie 34
3.4 Uitvoerbare transformatiemodellen 34
3.4.1 Gebruiksgerichte design patronen 34
3.4.2 Pattern mapping 34
3.4.3 Modelgebaseerde transformaties 37
3.4.4 Canonical Abstract Prototypes 38
3.4.5 Conclusie 39
4 Taxonomieën in adaptatietechnieken 41
4.1 Bepalen van een taxonomie 41
4.2 Tijdstip 42

4.3 Niveau 43
4.3.1 Doel 43
4.3.2 Taak 43
4.3.3 Semantiek 44
4.3.4 Syntactisch 44
4.3.5 Lexicaal 44
4.3.6 Conclusie 44
4.4 Techniek 44
4.4.1 External adaptivity 45
4.4.1.1 Reflectie 45
4.4.1.2 Introspectie 45
4.4.1.3 Polymorfisme 46
4.4.2 Self-contained adaptivity 46
4.4.3 Adaptatie-uitvoering 46
4.4.3.1 Switching 46
4.4.3.2 Reconfiguration 46
4.4.3.3 Enabling/disabling 46
4.5 Conclusie 47
5 Aanpak opstelling voor UIML.NET 48
5.1 Scenario's 48
5.2 UIML.NET 51
5.3 Adaptatie type bepalen 51
5.4 Adaptatie tijdsfase 52
5.5 Metadata om adaptatie te ondersteunen 53
5.6 Welke metadata gebruiken 54
5.7 Welke componenten aanpassen 55
6 The UIML.NET Adaptor 56
6.1 Adaptatie van de user interface 57
6.1.1 Indeling van de user interface 57
6.1.2 Adaptatieniveaus 58
6.1.3 Metadata laag 63
6.2 Modellering van de natuurlijke gebruikerscontext 70
6.2.1 Gebruiksomgeving 71
6.2.2 User interface 73
6.3 Mappings en beslissingsprocedures 75
6.3.1 Formalisatie van verbanden door mappings 75
6.3.2 Beslissingsprocedures en regels 78
6.3.3 Adaptatieproces 82
6.3.4 Voorbeeld 83
7 Discussie 85
7.1 Evaluatie 85
7.2 Future Work 85
7.2.1 Normaliseren van subjectieve attribuutwaarden 85
7.2.2 Methodes om de plasticiteitwaarde te berekenen van high level patronen 86
7.2.3 Generatie van mappings en adaptatieregels door artificiële intelligentietechnieken 88
7.3 Algemene conclusie 89
Referenties 91

Lijst met afbeeldingen

Figuur 1: User interface ontwikkeling voor meerdere gebruikersomgevingen. 1
Figuur 2: Gulf of execution en gulf of evaluation volgens Norman 9
Figuur 3: Model van de UIML specificatie 13
Figuur 4: Structuur van een UIML document 13
Figuur 5: XIML architectuur 15
Figuur 6: ISLM framework 17
Figuur 7: ISLM Meta-object abstractie 17
Figuur 8: ISML beschrijving. 18
Figuur 9: Ontwikkelingsstroom van user interfaces in Cameleon. 20
Figuur 10: The Context Management Framework 22
Figuur 11: Illustratie van metadata 23
Figuur 12: Abstractie niveaus van metadata. 24
Figuur 13: XML beschrijving. 25
Figuur 14: RDF principe. 26
Figuur 15: RDF graph. 27
Figuur 16: MPEG-21: Digital Item 28
Figuur 17: Digital Item Adaptation principe in MPEG-21 29
Figuur 18: Visualisatie van een FOAF beschrijving 30
Figuur 19: Fases in de adaptatiecyclus. 32
Figuur 20: Situering van verschillende modellen in de ontwikkelingscyclus 33
Figuur 21: Implementatie van het Quick Access patroon. 35
Figuur 22: Herdesigning van de user interface met patroon mapping 35
Figuur 24: Patroon geassisteerd re-engineering van de user interface 36
Figuur 25: Modelstroom in Model-based engineering of multiple interfaces with transformations 37
Figuur 26: Voorbeeld van een canonisch prototype 39
Figuur 27: Voorbeeld van een canonisch prototype 39
Figuur 29: Adaptatie aan een actieve taak 43
Figuur 30: Introspectief widget. 45
Figuur 31a: Weergave van resultaat van scenario 1. 48
Figuur 31b: Weergave van resultaat van scenario 1. 49
Figuur 31c: Weergave van resultaat van scenario 1. 49
Figuur 31d: Weergave van resultaat van scenario 1. 50
Figuur 32: Weergave van resultaat van scenario 2. 50
Figuur 33: Algemene werking van de adaptor. 56
Figuur 34: The Adaptor 57
Figuur 35: Samengestelde delen in een UIML gebaseerde user interface. 57
Figuur 36: Elementaire delen in een UIML gebaseerde user interface 58
Figuur 37: Situering van adaptatieniveau 1 59
Figuur 38: Situering van adaptatieniveau 2 60
Figuur 39: Situering van adaptatieniveau 3 61
Figuur 40: Situering van adaptatieniveau 4 62
Figuur 41: Situering van adaptatieniveau 5 63
Figuur 42: Stap 1 van beslissingsprocedure voor widget selecties. 78
Figuur 43: Stap 1 van beslissingsprocedure voor polymorfische delen 79
Figuur 44: Stap 2 van beslissingsprocedure. 79
Figuur 45: Stap 3 in beslissingsprocedure: opstelling beslissingsmodel. 80
Figuur 46: Adaptatiebeslissingen op niveau 2. 81
Figuur 47 : toewijzing van templates 82
Figuur 48: transformatie van widgets (niveau 3 en 4). 82
Figuur 49: Resultaat van adaptatie (1). 83
Figuur 50: Resultaat van adaptatie (2). 84

Figuur 51: Berekening van plasticiteitmatrices voor templates. 87

Lijst met tabellen

Tabel 1: Adaptatieniveau 1. 59
Tabel 2: Adaptatieniveau 2. 60
Tabel 3: Adaptatieniveau 3. 61
Tabel 4: Adaptatieniveau 4. 62
Tabel 5: Adaptatieniveau 5. 63
Tabel 6: Functionele metadata van een ‘class’ element. 64
Tabel 7: Overzicht van klassen in de SWF vocabulary. 65
Tabel 8: Metadata van een ‘reference’ element. 65
Tabel 9: Metadata van een ‘property’ element. 66
Tabel 10: Metdata van een ‘content’ element. 66
Tabel 11: Functionele metadata van een ‘part’ element. 67
Tabel 12: Metadata van een ‘part’ element met samengestelde delen. 67
Tabel 13: Overzicht van samengestelde klassen in de uitgebreide SWF vocabulary 68
Tabel 14: Metadata van een ‘part’ element dat een template importeert. 68
Tabel 15: Vergelijking tussen XSLT en transformatiemodules. 70
Tabel 16: Metadata van de gebruiker. 71
Tabel 17: Metadata van een apparaat. 72
Tabel 18: Semantische metadata van een ‘class’ of ‘template’ element. 73
Tabel 19: Uitleg bij de semantische parameters. 74
Tabel 20: Semantische metadata van een ‘constant’ of ‘template’ element. 74
Tabel 21: Metadata van ‘class’ attributen. 75
Tabel 22: Lijst van mappingfuncties. 77
Tabel 23: Lijst van mappingoperators. 77
Tabel 24: Lijst van Mappingoperators tussen contextattributen en ‘class’ attributen. 78
Tabel 25: Lijst van mappingoperators tussen ‘class’ attributen onderling. 78
Tabel 26: Voorbeeldprofiel 1. 83
Tabel 27: Voorbeeldprofiel 2. 83

Inleiding

Deze thesis is voornamelijk gericht op de vraag: “Welke soort metadata kunnen user interfaces bevatten en op welk niveau?” Dit met oog op het ondersteunen van intelligente adaptatie van user interfaces. De nood naar dit onderzoek is een gevolg van de toenemende opname van elektronische systemen en diensten door een meer heterogene groep van mensen. Gerelateerde trends die zich al langer voortzetten, zijn de toename van verschillende soorten toestellen waarmee deze systemen benaderd worden en de verschillende situaties waarin ze gebruikt worden. De rol van een user interface werd in het verleden onderschat en kan tegenwoordig als belangrijker dan ooit beschouwd worden. Een aangepaste user interface bepaalt in grote mate de efficiëntie waarmee een systeem gebruikt wordt. Waar in de eerste computersystemen de kern van het systeem en de technische efficiëntie centraal stonden, wordt de waarde van een modern systeem minstens evenveel bepaald door de user interface, zeker in die systemen waar de eindgebruiker rechtsreeks betrokken wordt.

Het onderzoek naar nieuwe user interfaces ligt de dag van vandaag nog lang niet stil en hangt veelal samen met de ontwikkeling van hedendaagse applicaties. Recent onderzoek tracht gebruik te maken van de volledige potentie van user interfaces. Op technisch vlak zijn we nog lang niet klaar met de ondersteuning van die nieuwe concepten in user interfaces; terwijl men reeds gedurende lange tijd bezig is met de non-humane aspecten van elektronische technieken, lopen we op gebied van gebruiksondersteuning eerder een achterstand op. Tegenwoordig zijn de begrippen “gulf of execution” en “gulf of evaluation” zeer zichtbaar aanwezig in de praktijk: een televisie programmeren, een gsm bedienen, tekstverwerker gebruiken, online bankverrichtingen uitvoeren, ... Allemaal zaken die niet zo vanzelfsprekend zijn voor iedereen. Er zijn (senioren-)opleidingen nodig om deze taken nog te kunnen uitvoeren, terwijl dit vroeger ook allemaal kon zonder opleiding of handleiding. Er is geen groot wetenschappelijk onderzoek nodig om vast te stellen dat dit te wijten is aan de vooruitgang van de techniek en aan het feit dat de gebruikersinterface niet in dezelfde mate mee evolueert. Het gebruik van een goed afgestelde user interface moet er voor zorgen dat de kloof tussen mens en machine niet groter wordt.

1 Wat is adaptatie?

Volgens de definitie uit het woordenboek [1] wil adaptatie het volgende zeggen: “het proces om zichzelf of iets anders aan te passen aan iets (bijvoorbeeld omgevingscondities).”. Adaptatie is iets wat de mens in zijn of haar dagelijks sociaal leven toepast, de één al wat beter dan de ander. Aan een kind van 4 jaar vragen we bij wijze van spreken hetzelfde dan aan een volwassene, maar dan wel op aangepaste manier. Iemand die verdrietig is, spreken we anders aan dan iemand die in extase verkeert. We bukken ons om een voorwerp door te geven aan een kleine persoon; we gaan onze handelingen en gedragingen aanpassen. Als we dit niet zouden doen, zouden heel wat boodschappen verkeerd overkomen en zou het moeilijk worden om activiteiten, met meer dan één betrokkene, tot een goed einde te brengen.

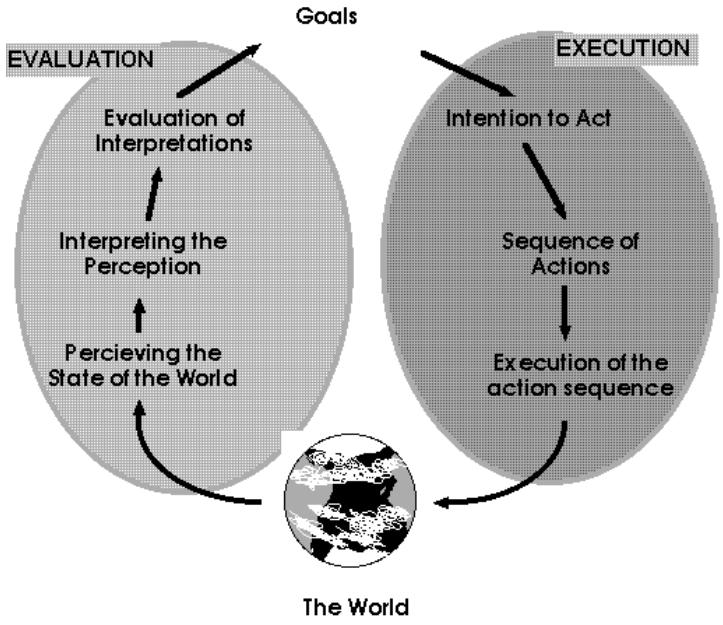
1.1 Adaptatie in de informatica

In de informatica betekent adaptatie: “Het aanpassen van elektronische toepassingen aan de gebruikssituatie.”. Adaptatie kan op verschillende plaatsen van een design en programma worden toegepast. Men kan het hebben over de uitvoering, presentatie, invoertechniek, het helpsysteem en de gebruikersrechten, en over de verschillende lagen van het systeem [3]. In gebruikersgerichte design methoden zal men in quasi elke fase van het ontwikkelingsproces rekening houden met de context waarin het eindproduct gebruikt zal worden. Taakmodel, presentatiemodel, applicatiemodel, gebruikersmodel en dialoogmodel [4] kunnen hier elk naar gericht worden en kunnen een weerslag hebben op de kwaliteit van de afstemming naar de context. Een systeem dat voor de ene gebruiker gegevens opslaat op een diskette en voor de andere, die dezelfde actie uitvoert, op een centrale database, is ook een vorm van adaptatie. In elk aspect van een softwaretoepassing zal men adaptatie op één of andere manier kunnen toepassen, en elk met zijn eigen technieken en elementen. Binnen deze studie gaan we adaptatie binnen HCI beschouwen.

1.2 Waarom is adaptatie nodig?

Bij elektronische toepassingen is de noodzaak aan adaptatie in principe hetzelfde als bij mensen. Als menselijke gebruiker weten we wel dat een computersysteem niet menselijk is en zullen we hier altijd rekening mee proberen te houden en ons hieraan proberen aan te passen. Doordat de aanpassing slechts van één kant komt, zal het op mentaal vlak moeilijker zijn een taak te laten uitvoeren door een computersysteem dan door een andere persoon. Ons adaptatievermogen is ook begrensd, waardoor sommige taken niet uitgevoerd kunnen worden. Men kan de kwaliteit van een designoplossing afmeten aan de ‘gulf of evaluation’ en ‘gulf of execution’ volgens Norman [5] (zie figuur 2).

Een tool of ‘werktuig’ (in ons geval een elektronische applicatie) is efficiënt in gebruik wanneer de gebruiker gemakkelijk deze ‘golven’ of ‘gulfs’ kan overbruggen. We kunnen de kwaliteit van de stappen in een dergelijke golf beschouwen in functie van de mentale inspanning die de gebruiker moet leveren om zich te adapteren naar het systeem. Er bestaan tal van richtlijnen om deze inspanning minimaal te houden, waarin de rode draad de nadruk legt op het schikken (of brengen) van het systeem naar de gebruiker. Daarom is het noodzakelijk dat ook het systeem zich aanpast aan de situatie.



Figuur 2: Gulf of execution en gulf of evaluation volgens Norman [5].

1.3 Waarom is adaptatie meer nodig dan vroeger?

In tegenstelling tot hetgeen in de vorige paragraaf beweerd wordt, kunnen we argumenteren dat adaptatie reeds voldoende wordt toegepast door de 'user centered design' methoden en de meeste toepassingen succesvol gebruikt worden. De situatie is echter veranderd ten opzichte van vroeger. Als we terugkeren naar de jaren ‘60 merken we dat bijna alle computergestuurde toepassingen (software) door computerdeskundigen gebruikt werden. Na verloop van tijd begon software echter meer door te dringen naar een publiek met slechts matige computerkennis. In het begin was deze software gericht naar één bepaalde groep gebruikers voor het verrichten van een bepaalde taak. Maar hoe meer we naar het heden toe gaan, des te meer software en computersystemen er gebruikt worden door bredere groepen van mensen. De meeste gebruikersgerichte toepassingen worden al lang niet meer ontworpen naar één profiel, maar naar meerdere. In het designproces is het meer regel dan uitzondering dat men meerdere 'personas' [6] gebruikt. Doordat deze personas tegenwoordig zo verschillend kunnen zijn, is een '1 size fits all' benadering zelden een haalbare uitweg. Er bestaat geen 'beste' interface waarbij de ‘gulf of evaluation’ en ‘gulf of execution’ voor alle gebruikers en situaties hetzelfde zal zijn. De resultaten van de kwaliteitstesten van Norman zullen telkens een ander resultaat geven. Hierdoor wordt de noodzaak tot aanpasbaarheid en personalisering een feit [2].

1.4 Adaptatie in user interfaces

In deze thesis richten we ons naar de adaptatie van user interfaces. Dit wil echter niet zeggen dat de modellen in andere lagen van het systeem niet in rekening gebracht hoeven te worden, integendeel. Doordat de user interface zich in de bovenste laag (die op zich uit meerdere sublagen bestaat) bevindt, is ze het resultaat van de onderliggende lagen en zullen beslissingen in deze onderste lagen een weerslag hebben op de bovenste, met name de user interface. Dit maakt adaptatie in user interfaces een uitdaging die groter is dan vele andere vormen van adaptatie in elektronische toepassingen.

2 Analyse

Voor we op zoek gaan naar (betere) adaptatiemethoden en resultaten, en alvorens we de moeilijkheden gaan blootleggen, zal er eerst een classificatie van adaptatie en de bijhorende aspecten uitgediept worden.

Opmerking: Deze thesis is gericht naar het adapteren van user interfaces. Veel adaptatieprincipes zijn echter van toepassing op adaptatie in het algemeen en dus niet beperkt tot het gebied van user interfaces. Het zou moeilijk zijn om elke keer te vermelden of het gaat om algemene wetenschap of wetenschap die specifiek van toepassing is op user interfaces. Uit de context zal duidelijk worden over welke van de twee het gaat; indien dit niet het geval is, zal het expliciet vermeld worden.

2.1 Formele definiëring van het begrip adaptatie

In het volgende deel gaan we meer formeel naar het begrip adaptatie kijken vanuit verschillende invalshoeken. De bedoeling is hieruit de basisvereisten op te stellen voor het samenstellen van adaptatiemodellen en -technieken.

2.1.1 Doel van adaptatie

Een belangrijke vraag die men zich moet stellen is de volgende: 'Welk resultaat wordt er van adaptatie verwacht?'. Deze vraag mag zeker niet onderschat worden. Adaptatie heeft niet slechts één concreet doel, maar kan voor verschillende redenen worden toegepast. Tot nu toe hebben we al vastgesteld dat adaptatie toegepast wordt om user interfaces flexibel te maken in verscheidene contexten.

Om het doel van adaptatie formeel te definiëren is het begrip 'plasticity' of 'plasticiteit' geïntroduceerd [7]. *Plasticiteit van interactieve systemen is het aankunnen van veranderingen aan de context terwijl de bruikbaarheid van het systeem behouden wordt (vergelijking met een elastiek)*. Omdat 'context' op verschillende manieren geïnterpreteerd kan worden, gaan we ook dit begrip formeel vastleggen.

Context is elke informatie die gebruikt kan worden om de situatie van een entiteit te karakteriseren. Een entiteit is een persoon, plaats of object dat relevant is voor de interactie tussen een gebruiker en een applicatie (inclusief de gebruiker en applicatie zelf). De situatie is een beschrijving van de staat van de entiteiten. [8].

In het onderzoek naar plasticiteit gebruikt men de entiteiten eindgebruiker, eindplatform en gebruikersomstandigheden. Formeel wordt de context hier uitgedrukt als een tripel: (gebruiker, platform, omgeving) [9].

De definitie van adaptatie kan als volgt opgesteld worden: Adaptatie betekent veranderingen in het systeem brengen om te accommoderen aan de verandering in haar omgeving. Meer specifiek: adaptatie van een software systeem (S) wordt veroorzaakt door een verandering (delta E) van een oude omgeving (E) naar een nieuwe omgeving (E') en resulteert in een nieuw systeem (S') dat uiteindelijk voldoet aan de noden van haar nieuwe omgeving (E'). Dit zou men ook kunnen opvatten als een functie: *Adaptatie: E x E' x S --> S'*, waarbij *voldoet_aan(S', nood (E'))* [11, 12].

Adaptatie van een user interface aan de context is dus een middel om user interfaces plastisch te maken zodat deze blijft voldoen aan de noden van de gebruiker in die context.

2.1.2 Wanneer is adaptatie juist uitgevoerd?

De entiteiten en factoren in het vorige deel (zie 2.1.1) zijn dezelfde als die (mogelijk) in de design fase van user interfaces en applicaties in het algemeen worden opgenomen om het design te gaan bepalen. We kunnen dus stellen dat na adaptatie van een user interface, deze het resultaat zou moeten zijn van een aangepast design, en dit als gevolg van veranderingen in de context.

Men kan dus spreken van een 'succesvolle' en 'goede' adaptatie wanneer de functionele en niet-functionele eigenschappen, die in de design fase gebruikt worden om de bruikbaarheid te meten, behouden worden [7]. We kunnen nu al voorspellen dat methoden toegepast zullen moeten worden om te evalueren wat de eisen hiervoor zijn en wanneer een user interface hieraan voldoet.

2.2 Analytisch onderzoek van adaptatie in user interfaces

Het zou fout zijn de user interface als één homogeen en elementair onderdeel te beschouwen. Als men een user interface gaat aanpassen, kan dit op verschillende manieren. Bij adaptatie is het van belang welk onderdeel men gaat aanpassen (verwerken). Er zijn verschillende onderverdelingmogelijkheden, granulariteiten en taxonomieën van de onderdelen en aspecten van user interfaces, afhankelijk van het standpunt van waaruit men deze bekijkt [10]. Dit kan vanuit technisch standpunt, gebruikersstandpunt, designstandpunt, ... zijn. Er zijn geen vaste regels of grenzen die bepalen uit welk standpunt men het best vertrekt of welke filosofie men het best volgt. Doordat het de bedoeling is adaptatie te gaan toepassen op UIML user interfaces, zal onze invalshoek op de verschillende onderdelen van een user interface afhangen van de manier waarop UIML ze beschrijft. Daarom gaan we in de volgende delen UIML van naderbij bekijken en situeren t.o.v. andere user interface beschrijvingstalen.

2.2.1 Voorwoord: waarom een HLUIDL gebruiken

UIML is een HLUIDL of High Level User Interface Description Language. Een UIDL is een beschrijvende user interface taal (HTML, WML, Xforms, ...). Dit zijn meestal XML-gebaseerde talen en kunnen in zekere mate platformonafhankelijk zijn. De user interface wordt door middel van tags beschreven in een simpele en door mensen leesbare taal. De user interface wordt at runtime geïnterpreteerd, dus vanaf het moment dat ze uitgevoerd (en benaderd) wordt. Dit wordt ook het renderen van de interface genoemd. Het aantal platformen die een bepaalde taal ondersteunt hangt enkel af van de beschikbaarheid van een compiler of renderer voor deze taal. Doordat een UIDL de user interface op een zeer expliciete manier beschrijft, zal de interface in de praktijk meestal voor slechts 1 platform bruikbaar zijn, hierin ligt het verschil met HLUIDs.

HLUIDLs (UIML, AUIML, XIML, SeescoaXML, ...) beschrijven user interfaces op een abstracte manier. De verschillende abstracte componenten die beschreven worden in deze taal kunnen gemapt worden op widgets en componenten uit een bepaalde toolkit of op elementen van een UIDL. Het voordeel van het werken met deze talen is de vereiste kennis van slechts één syntax, namelijk die van de HLUIDL, voor het ontwikkelen van user interfaces voor verschillende platformen. Dit verhoogt de consistentie, flexibiliteit en aanpasbaarheid van de user interfaces. Doordat deze talen de user interface op een meer abstracte manier beschrijven, waarbij men minder rekening hoeft te houden met implementatiedetails, zijn ze het meest geschikt voor de ontwikkeling van adapteerbare user interfaces. Ze laten het meest ruimte voor implementatie van de concrete user interface. Sommige HLUIDLs zijn slechts een onderdeel van een groter framework waarin adaptatie reeds wordt toegepast. Het loont dus de moeite om ook andere HLUIDLs te bekijken.

2.2.2 UIML

UIML staat voor User Interface Markup Language en is een uitgewerkte OASIS standaard voor de beschrijving van user interfaces. UIML op zich is een meta-taal die de user interface opdeelt in 4 generische aspecten:

- Structuur : De opbouw en verschillende componenten van de user interface door middel van abstracte 'parts'.
- Inhoud : Expliciete lexicale en multimedia inhoud die weergeven wordt in de user interface.
- Stijl : Specificatie van eigenschappen van de generische parts of presentatiestijl.
- Gedrag : Functionaliteit en gedrag van de verschillende elementen voor interactie met de user interface.

Om de uitvoerbare user interface te genereren op een bepaald platform wordt er voor dat platform een vocabulary opgesteld, die de abstracte parts mapt op een specifiek widget van een user interface toolkit of andere beschrijvende taal (Glade, Windows Forms, Java Swing, HTML, ...).

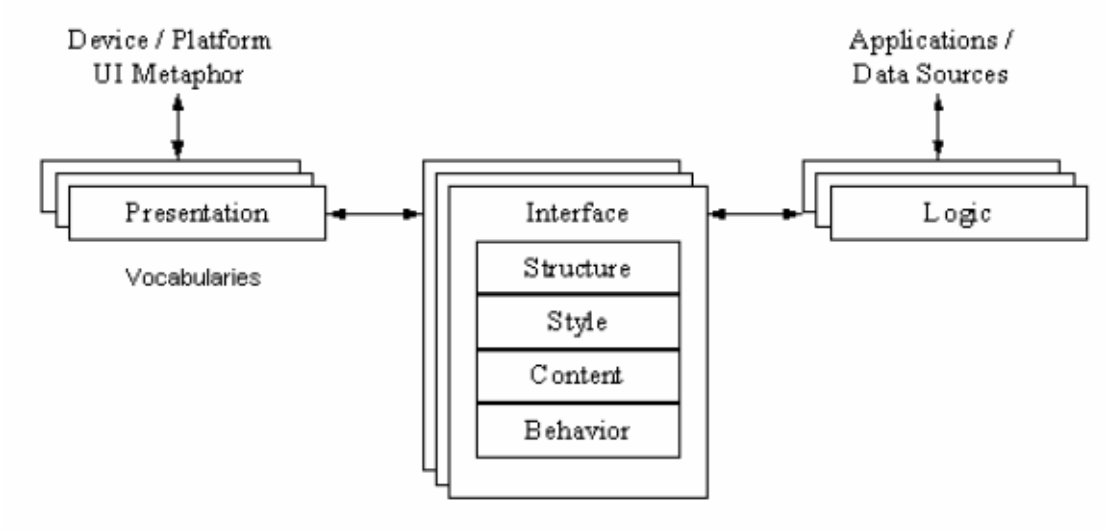
De communicatie met de achterliggende applicatie wordt beschreven in de 'logic' sectie. Voor elk doelplatform hoeft slechts 1 keer een vocabulary te worden geschreven om user interfaces hiervoor te kunnen schrijven, wat UIML een platformonafhankelijke taal maakt. Echter moet er hiervoor een UIML compiler, renderer genoemd, beschikbaar zijn die deze taal ondersteunt of toegang heeft tot de toolkit. De manier waarop vocabularies beschreven moeten worden, en de uiteindelijke user interface gaan bepalen, hangt gedeeltelijk af van de UIML compiler op een specifiek platform. [13, 76]

2.2.2.1 UIML en adaptatie

In theorie zou men met UIML eenmalig een user interface kunnen beschrijven en deze met één of meerdere vocabularies mappen op verschillende platformen. Hierdoor kan men adaptatie naar de gebruiker en omgeving toepassen op 1 user interface beschrijving terwijl de adaptatie naar platform (apparaat, software) automatisch verder wordt afgehandeld. Een UIML-beschrijving is hiervoor echter niet abstract genoeg [14]. Vooral in de stijl sectie worden al apparaat- en taalafhankelijke beslissingen genomen. Voor elke taal en apparaat zal men dus opnieuw een user interface moeten specificeren, waardoor deze in wezen apparaat- en platformafhankelijk zijn. Het gebruik van applicatiespecifieke interactie is beperkt door de mapping methode die wordt toegepast, waardoor UIML niet volledig apparaatonafhankelijk lijkt te zijn en slechts geschikt is voor een beperkt aantal categorieën van eindplatformen. Adaptatie op UIML-gebaseerde user interfaces is hiermee een grote uitdaging, waarin ook rekening moet gehouden worden met de vocabularies, aangezien deze niet gedefinieerd zijn in de UIML specificatie maar voor elk eindplatform moeten opgesteld worden. Dit komt omdat UIML een meta-taal is en op zich niet gebruikt wordt om een user interface te beschrijven. Er zijn echter wel een reeks algemeen aanvaarde vocabularia beschikbaar die als standaard kunnen gebruikt worden, onderandere voor HTML, Java AWT, Java Swing, Windows Forms, Compact Windows Forms, GTK+, CSS, JavaScript, VXML en WML.

2.2.2.2 Welke onderdelen adapteren

In UIML wordt een scheiding gemaakt van de eigenschappen van de verschillende onderdelen, de structuur van de onderdelen en de gedragingen ervan. Op deze manier kan men zich richten tot 1 aspect of parameter van een onderdeel, zonder te moeten sleutelen aan de specificatie van het onderdeel zelf of andere eigenschappen ervan.



Figuur 3: Model van de UIML specificatie [82]

Een andere eigenschap van de UIML taal zijn de scheiding van content (inhoud) en gebruik van (herbruikbare) templates. Templates kunnen gebruikt worden in de verschillende secties en kan dus structuur, stijl of behavior informatie bevatten.

Een derde interessante eigenschap is de universele manier waarop onderdelen en eigenschappen gedefinieerd worden. Dit gebeurt onafhankelijk van het soort onderdeel, de klasse, of de soort eigenschap. Dit is een noodzakelijke vereiste om solide, universele, en flexibele adaptatie te kunnen ondersteunen.

```
<?xml version="1.0">
<!DOCTYPE uiml PUBLIC
"-//Harmonia//DTD UIML 3.0 Draft//EN"
"http://uiml.org/dtds/UIML3_0a.dtd">
<uiml xmlns='http://uiml.org/dtds/UIML3_0a.dtd'>
<head>        ... </head>
<template>    ... </template>
<interface>
     <structure>   ... </structure>
     <style>       ... </style>
     <content>     ... </content>
     <behavior>    ... </behavior>
</interface>
<peers>
     <logic>       ...</logic>
     <presentation>...</presentation>
</peers>
</uiml>
```

Figuur 4: Structuur van een UIML document [82].

UIML lijkt abstract genoeg te zijn om een zeker niveau van adaptatie te ondersteunen, met name op aspecten die in vele modellen voorkomen : Interactie, presentatie, gedrag en dialoog.

Interactie

We kunnen interactie omschrijven als de manier waarop gebruikers met een user interface communiceren. De acties, die nodig zijn om een taak uit te voeren, bepalen mede de gebruiksvriendelijkheid van een interface en kunnen per context verschillen. Daarom is het nuttig om interactie te adapteren aan de context. Bij het selecteren van een geschikte interactiemethode en dus bij het bepalen van welke acties vereist zijn om een bepaalde taak uit te voeren, zal men rekening moeten houden met lichaamsbewegingen die nodig zijn voor een actie (vingers, armverplaatsingen, mond), voorkeuren, begrijpelijkheid, efficiëntie, etc. Hiervoor bestaan tal van richtlijnen en methoden. Interactie kan worden aangepast in UIML door: 1) het "class" attribuut van een part aan te passen, 2) een andere template te gebruiken, 3) de vocabulary in de "peers" sectie aan te passen.

De logic sectie kan ook een invloed hebben op de interactie maar is zo expliciet dat ze niet zo interessant is om adaptaties op uit te voeren.

Presentatie

De vorm waarin informatie aan de gebruiker wordt overgebracht, noemt men de presentatie van de user interface. Het bepaalt het beeld dat een gebruiker over een systeem heeft. Presentatie is niet enkel het grafisch design van een interface; presentatie bepaalt ook welke informatie wanneer wordt weergegeven en op welke manier. Er zijn meestal meerdere mogelijkheden om een bepaald gegeven voor te stellen. Zo kan een bepaalde waarde symbolisch of literair weergeven worden. Hiervoor bestaan ook weer verschillende granulariteiten en standpunten over presentatie-categorieën. Een andere verdeling kan zijn: grafisch of textueel, impliciet of expliciet, etc.

Aspecten die in UIML binnen het presentatiegebied horen, zijn layout, grootte, kleur, densiteit, vorm, symmetrie, expressie, associatie, medium, etc. Het zoeken naar de geschikste presentatievorm is een vakgebied op zich, en kennis van informatievisualisatietechnieken is hier een onderdeel van. Aangezien de geschiktheid van presentatie contextafhankelijk is, is het nuttig om presentatie te adapteren aan de context. Dit kan met behulp van 1) Aanpassingen in de structure, style en content sectie binnen de interface sectie, 2) andere templates gebruiken, 3) de vocabulary in de peers sectie aanpassen, 4) "Constants" en "references" in de "content" sectie aanpassen.

Gedrag

Een ander aspect dat een user interface karakteriseert, is haar gedrag naar de acties van de gebruiker. Wanneer men bijvoorbeeld met de muis over een item beweegt, kan het systeem extra informatie over dit item weergeven. In andere gevallen zou dit onwenselijk kunnen zijn. Terwijl dit gedrag meer over de verwerking van invoer gaat, bedoelen we in deze context met gedrag feitelijk de reacties in de user interface zelf. Voorbeelden van gedragingen van user interfaces zijn: een geactiveerd gebied highlighten of vergroten, (auditieve) feedback geven bij een bepaalde handeling, hulp, predictieve invoer, en automatisch visualiseren van controles vanaf ze beschikbaar zijn, en ze weer laten verdwijnen wanneer ze niet langer van toepassing zijn.

Het gedrag van een user interface kan suggestief zijn en de gebruiksvriendelijkheid bevorderen. Maar zoals eerder vermeld, kan het gedrag van een user interface ook storend zijn en zijn efficiëntie verkleinen. Een 'preview' van een bewerking weergeven alvorens deze effectief wordt toegepast, zodat de gebruiker een idee krijgt over het resultaat van deze bewerking, is een zeer handige feature. Indien men daarentegen, dezelfde toepassing benadert met een apparaat met weinig processorkracht, zal dit de invoer vertragen of zelfs tijdelijk blokkeren.

Aangezien een bepaald gedrag in een bepaalde situatie meer kwaad kan doen dan goed, is gedrag zeker een kandidaat om in overweging te nemen voor adaptatie. In UIML kan dit verwezenlijkt worden door aanpassingen in de "behavior" sectie te maken.

Dialoog

De dialoog verschilt wat t.o.v. de andere geabstraheerde aspecten omdat deze op een hoger niveau toepasbaar is. Men kan niet altijd in één oogopslag zien hoe de dialoog tussen de user interface en de gebruiker eruit ziet. Een dialoog gaat over de interactie tussen gebruiker en user interface over

een bepaalde tijdsperiode. Om een taak te ondersteunen, gaat men dialoogmodellen opstellen die de informatie-uitwisseling tussen gebruiker en (bovenste laag van de) user interface beschrijven. Deze dialoogmodellen beschrijven de invoer, de acties van de gebruiker en de reactie van het systeem hierop. De volgorde van acties kan hier van belang zijn. De beschrijving van een dialoog omvat dus ook interactie, maar dan op een hoger niveau. De interactiemetafoor die men gebruikt, heeft onvermijdelijk gevolgen voor het dialoogmodel. Aangezien er veel dialoogmodellen voor de ondersteuning van een bepaalde taak mogelijk zijn en dit zeer sterk afhangt van gebruiker en eindplatform, is het zeker niet onbelangrijk om de dialoog op te nemen in het onderzoek naar adaptatiemethoden. Zo is bijvoorbeeld een metafoor met directe manipulatie of WIMP-interface is op sommige apparaten onmogelijk.

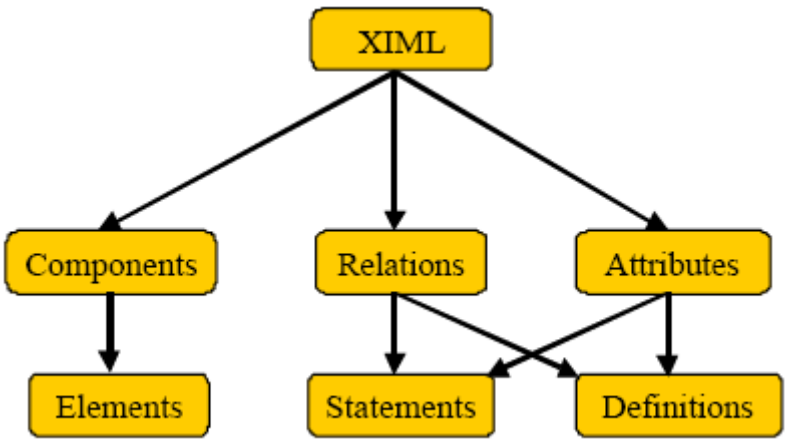
Aangezien dialoogmodellen redelijk vroeg in het designproces worden opgesteld, zal het niet eenvoudig zijn om in UIML deze rechtstreeks aan te passen. Doordat er in een UIML document geen informatie beschreven staat over het dialoog waartoe ‘parts’ of andere componenten behoren zullen we ofwel 1) met templates moeten werken of 2) de parts in de “structure” sectie herverdelen over meerdere UIML-documenten of samenvoegen tot 1 UIML-document. Op dit gebied schiet UIML dus te kort. Dit komt ondermeer doordat UIML gemaakt is om platformonafhankelijkheid van de taal op zich te bekomen en minder voor de ontwikkeling van apparaat- of contextonafhankelijke user interfaces.

2.2.3 Andere HLUIDs en frameworks

2.2.3.1 XIML

XiML staat voor eXtensible Interface Markup Language. De specificatie wordt voornamelijk beheerd door RedWhale Software. XiML gaat uit van het principe alle informatie die te maken heeft met user interfaces, te centraliseren en te benoemen als 'interactie data' [16, 17, 18]. De specificatie is gebaseerd op studies van 2 domeinen:

- ontologieën en de representatie ervan
- modelgebaseerde interface ontwikkeling



Figuur 5: XIML architectuur

De representatie principes van XiML zijn afgeleid van de eerste domeinstudie. De karakteristieken en de verschillende soorten van interactie data zijn gebaseerd op de studies van het tweede domein. Verder is de taal ook vaak geïnspireerd op ontwikkelde user interface management systemen.

XiML voorziet een collectie interface elementen die zijn onderverdeeld in verschillende hoofd interface componenten. Er kunnen een onbeperkt aantal interface componenten worden gedefinieerd. XiML voorziet 5 basis interface componenten:

- task (taak) component : Bevat (gebruikers) taken die de interface ondersteunt. De component bestaat uit een hiërarchische samenstelling van subtaken en de stroom of flow tussen deze taken. Hierdoor ligt de granulariteit niet vast en kunnen taken op elk niveau beschreven worden.
- domain (domein) component : Bestaat uit een gestructureerde georganiseerde collectie van objecten en classen. Het is een hiërarchie van objecten die door de gebruiker kunnen bekeken of gemanipuleerd worden. Ook deze kunnen zich op elk niveau bevinden.
- user (gebruiker) component : Definieert een hiërarchie van gebruikers en hun attributen die het design beïnvloeden. De attributen definiëren dus de karakteristieken van de gebruiker. Het gaat hier weer om een hiërarchie waardoor zowel gebruikersgroepen en individuele gebruikers beschreven kunnen worden.
- dialog (dialoog) component : Definieert een gestructureerde collectie van dialoog elementen die de acties bepalen die voor een gebruiker van de user interface beschikbaar zijn. Ook de stroom (flow) tussen de interactie acties, die de navigatie in de user interface bepalen, worden beschreven. Deze component lijkt op de task component maar werkt op een zeer laag en concreet niveau.
- presentation (presentatie) component : Definieert een hiërarchie van interactie elementen die communiceren met de gebruiker in een user interface. Om de user interface context- en platformonafhankelijk te maken worden deze best op een hoog niveau beschreven (hoge granulariteit), zodat de code en procedures van het interactie element gescheiden is van hun definitie. Hoe het element wordt gevisualiseerd wordt volledig overgelaten aan het doel systeem.

Eigenschappen van component worden gedefinieerd aan de hand van attributen waar een waarde aan toegekend wordt. Naast deze componenten zijn er nog andere componenten geïdentificeerd maar deze gaan we niet bespreken.

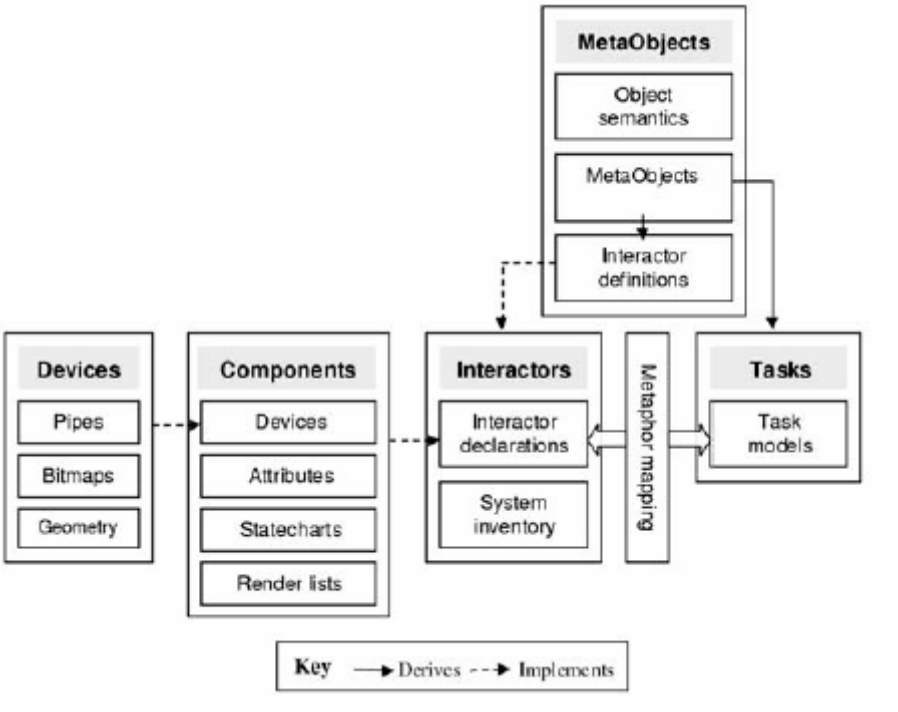
XiML definieert ook relaties tussen deze componenten die 2 elementen al dan niet in dezelfde component linkt. Deze relaties vormen de designkennis van de user interface en bepalen design beslissingen.

Conclusie
XiML is één van de meest uitgebreide HLUID's met een duidelijke splitsing tussen abstracte en concrete user interface. Zo beschrijven taak-, domein- en gebruikerscomponenten abstracte aspecten en presentatie- en dialoogcomponenten concretere aspecten van de user interface, wat niet voorkomt in UIML. XiML is gemaakt op een manier zodat deze componenten gebruikt kunnen worden doorheen de hele ontwikkelingscyclus, en maken deze taak zeer geschikt om uit te breiden of te gebruiken voor de specificatie van adapteerbare user interfaces.

2.2.3.2 ISML

ISML (Interface Specification Meta-Language) [19] is een onderdeel van het model-gebaseerde ISML-framework. ISML gaat uit van gedeelde concepten tussen gebruiker en computer, metaforen genaamd, en koppelt deze los van enige implementatie. Om dit metafoormodel te ondersteunen gebruikt het framework mappings tussen zowel gebruikersgeoriënteerde modellen (zoals taakbeschrijvingen), en software architecturale kwesties (interactor definities). Deze concepten worden samengesteld met 5 lagen en een reeks mappings die deze aan elkaar linken. De lagen in het

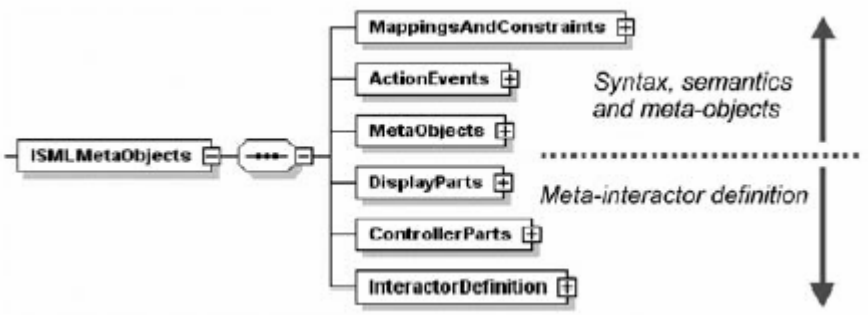
model erven of implementeren abstracties van elkaar dankzij gedeelde berekenbare formalisaties die lijken op event modellering, taakmodellen, abstract-to-concrete mappings, etc.



Figuur 6: ISLM framework [19].

We bespreken elke laag met de bedoeling de rol van de meta-informatie te achterhalen en hoe deze kan gebruikt worden voor automatische intelligente adaptatie.

- Component laag : In deze laag worden presentatie eigenschappen van een concreet interface object gespecificeerd, zoals hoogte en breedte. Statecharts, beschreven met een topologie van nodes en transities, worden gebruikt om devices te peilen op nieuwe in- en uitvoerinformatie. De effectieve rendering gebeurt door renderinglijsten waarin functionele aanroepen naar de, met de component geassocieerde, 'devices' zijn opgeslaan.
- Device laag : Bevat eenvoudige abstracties van basis attributen en low-level functies van in- en uitvoerapparaten (pipes, bitmaps en geometry). Een device is hier dus geen abstractie van computer hardware maar voorzien I/O (input- en output) operaties voor API's zoals bijvoorbeeld DirectX van Microsoft.
- Meta-Object laag : Deze laag is het interessantste met betrekking tot de meta-informatie die het fundament vormt van het framework. We gaan deze laag dan ook iets meer in detail bespreken dan de anderen. De metaforische aspecten van de user interface waar het framework van uitgaat, worden in deze laag vastgelegd door syntactische en semantische definities.



Figuur 7: ISLM Meta-object abstractie [19].

- Syntax en semantieken : Voor elke meta-object abstractie worden eerst relaties en communicatie tussen de meta-objecten beschreven met syntactische en semantische regels. 'Action events' zijn syntactische beschrijvingen tussen alle mogelijke communicaties tussen objecten.
  Een voorbeeld: een object vraagt gebruikersrecht van een ander object door een action-event op te roepen.

```
<AE Name="RequestOwnershipAction">
   <Parameters>
      <Param Name="eventSender" Type="SET"/>
      <Param Name="x" Type="INTEGER"/>
      <Param Name="y" Type="INTEGER"/>
   </Parameters>
</AE>
```

Figuur 8: ISML beschrijving.

Opmerking: Het ISML framework gebruikt een Backus-Naur Form gebaseerde grammatica (http://en.wikipedia.org/wiki/Backus-Naur_Form) om objecten te specificeren, maar voor de duidelijkheid beschrijven we dit met XML.

De eerste parameter (eventSender) identificeert het aanroepende object. De parameters x en y specificeren de methode voor het selecteren van het object. Het concept 'slepen' bijvoorbeeld wordt uitgedrukt door een mapping-constraint expressie, gebaseerd op mathematische en logische argumenten die bron- en doelargumenten specificeren waarop ze opereren.

Meta-objecttypen omschrijven de metaforische objecten die gespecificeerd zijn in het ISML framework. Dit zijn de abstracte delen die samengesteld zijn uit attributen en statemodels. Meta-interactortypen omschrijven de interactors (zie verder), gebaseerd op de reeds gedefinieerde metaobjecten.

- Interactor-laag : Deze laag realiseert de metaforen door verfijning en mapping naar voorgedefinieerde componenten.
- Taak-laag : Deze laag hergebruikt meta-object abstracties (mapping-constraints, action-events en metaobjecten) om taakgerelateerde entiteiten en hun rollen te beschrijven in een hiërarchische beschrijving van taken.
- Metafoor mapping : Dit deel van de ISML specificatie maakt mappings van objecten en acties, die gedefinieerd zijn in het taakmodel, naar interactors en interacties aan de user interface.

Conclusie

ISML ontkoppelt het metafoormodel van zijn specifieke implementatie, en drukt de mappings uit tussen concepten die gedeeld worden tussen de gebruiker en het systeem. Doordat dit mechanisme absoluut geen manifestatie heeft tov. zijn geïmplementeerde presentatie in de user interface, kunnen we dit metafoormodel overnemen voor adaptatiedoeleinden. Hierbij beschrijft men a.d.h.v. dit model eenmalig de user interface, en past men contextafhankelijke mappings toe om tot de concrete user interface te komen.

2.2.3.3 Seescoa XML

Het hoofddoel van Seescoa (Software Engineering for Embedded Systems using a Component

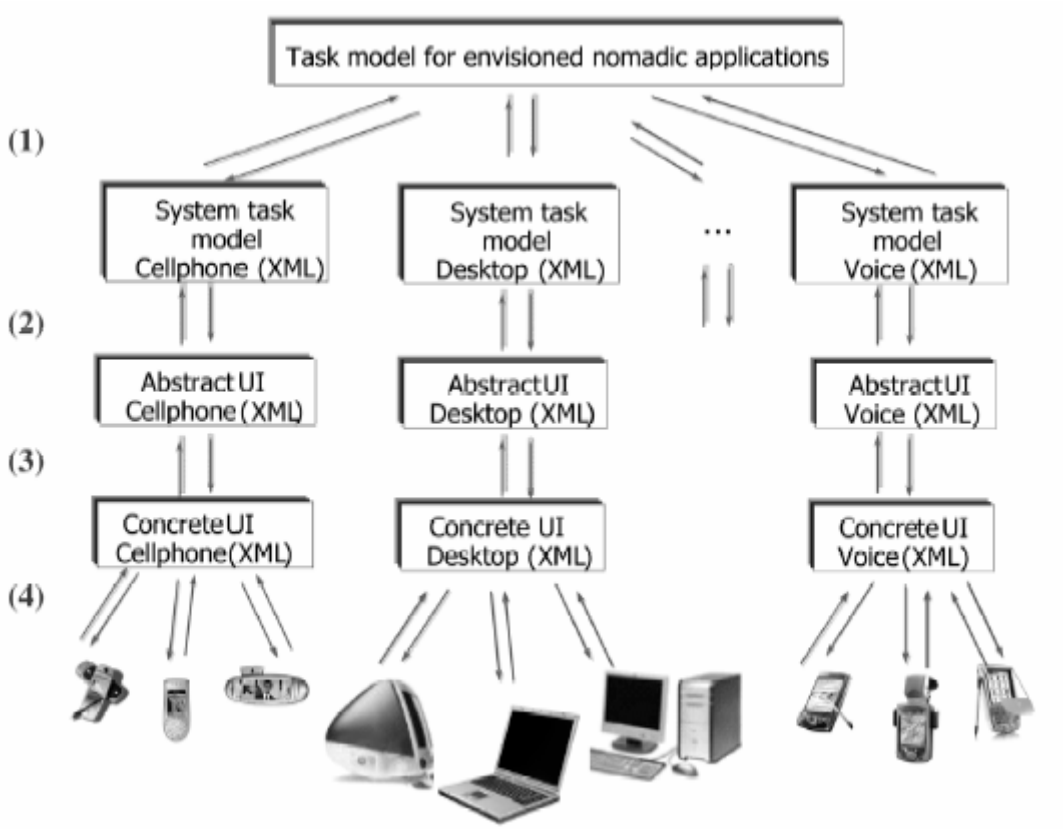
Oriented Approach) dat ontwikkeld werd in een samenwerkingsproject tussen de KUL, VUB, LUC en UGent, is om software engineering technologieën te adapteren naar de noden van embedded software [21]. Zo is er een architectuur voor at runtime serialisatie van Java user interfaces in XML beschrijvingen. Deze XML beschrijving beschrijft door middel van 'Abstract Interaction Objects' (AIO's) een abstractie van de user interface. De renderer (op het target platform) is vrij om andere manieren te kiezen voor de presentatie van de user interface met dezelfde functionaliteit.
Voor elk systeem is een XSLT gedefinieerd die de AIO van de abstracte user interface beschrijving mapt naar 'Concrete Interaction Objects' (CIO's) op de fundamenten en beperkingen van elk platform.

De componenten kunnen ook de 'look and feel' van de user interface beschrijven. Naast presentatie tags (waarvan er op dit moment 6 verschillende interactors beschikbaar zijn), zijn er de actie tags, die de actie specificeren die moet uitgevoerd worden wanneer de interactor gemanipuleerd wordt.

Conclusie
Seescoa laat toe de user interface op verschillende apparaten te weergeven en te gebruiken. Maar net doordat Seescoa gericht is op het genereren van een user interface waarbij de implementatietechniek aangepast is aan een vast toestel, en dus voor elk toestel een transformatie gedefinieerd dient te worden, is Seescoa niet ideaal voor ondersteuning van adaptaties aan een veranderlijke context. De adaptaties zouden dan toegepast moeten worden op de transformaties (XSLTs). Echter interessant is de gelaagdheid tussen AIO's, CIO's en de concrete user interface [22]. De renderer kan nog steeds een andere presentatie geven aan de user interface en de configuratie bepalen. Als deze beslissingen genomen worden op basis van de context hebben we aan dit uitgebreid model voldoende voor de ondersteuning van generatie van gepersonaliseerde user interfaces.

2.2.3.4 Teresa (Cameleon)

Het doel van het Cameleon project (Context Aware Modelling for Enabeling and Leveraging Effective interactiON) is om methoden en omgevingen op te bouwen die design en ontwikkeling ondersteunen van bruikbare contextafhankelijke interactieve software systemen [38]. Een belangrijke tool die hiervoor ontwikkeld werd is TERESA (Transformation Environment for inteRactivE Systems representAtions) [39, 40]. Hierin wordt vertrokken van een platform- en deviceonafhankelijk taakmodel van de applicatie. Vervolgens wordt deze gemapt op één of meerdere systeem taakmodellen waarvoor de applicatie ontwikkeld wordt. Hierna wordt van dit systeemmodel de abstracte user interface afgeleid, om er vervolgens een apparaat specifieke concrete user interface van af te leiden. De tool maakt gebruikt van XML-talen om de verschillende modellen voor te stellen.



Figuur 9: Ontwikkelingsstroom van user interfaces in Cameleon.

De ontwikkeling van TERESA is gebaseerd op de volgende vereisten:

- Model-gebaseerd
- Top-down aanpak (eerst abstracte beschrijvingen, en vervolgens meer concretere representaties)
- Variabel initiatief (designers bepalen zelf de automatisatie van de tool)
- XML-gebaseerd
- Verschillende vertrekpunten (designers moeten niet per se van het taakmodel beginnen)

Conclusie

Teresa bewijst dat creatie van bruikbare user interfaces in verschillende contexten mogelijk is door te vertrekken van 1 model [40]. Maar dit is niet voldoende voor volautomatische adaptatie. Teresa helpt enkel designers bij het nemen van beslissingen door oplossingen voor te stellen. Het geeft aan de hand van de modellen tips aan de ontwikkelaar voor bijvoorbeeld de selectie van een concreet interactie object in een concreter model. Teresa maakt hierbij gebruik van attributen van vaste eindplatformen. Dit zijn momenteel Web, Web voor mobiele apparaten en voice interfaces (generaties van VoiceXML). De attributen van deze eindplatformen kunnen uiteraard verfijnd worden, maar voorlopig worden geen andere platformen ondersteund en wordt de context beschouwd als het apparaat, en niet de gebruiker of natuurlijke omgeving. De overgang van concrete user interface beschrijving naar implementatie (code) kan wel volledig automatisch gebeuren.

Het concept van te vertrekken vanuit een taakmodel kan gebruikt worden voor adaptatie van user interfaces, maar vereist meer (generieke) data van het eindplatform opdat de tool volledig autonome beslissingen en transformaties kan berekenen.

2.2.4 Conclusie

Vele user interface beschrijvingstalen (XiML, TADEUS XML, SeescoaXML) baseren zich op modellen die in design- en engineeringmethoden gebruikt worden. Men tracht hierbij de overgang van verschillende modellen naar implementatie te overbruggen en de interface als een uitvoerbare modelvorm te beschrijven. Hierdoor kan men de user interface op een uniforme manier beschrijven en hoeven ontwikkelaars zich niet bezig te houden met implementaties. Om specificaties van user interfaces te ondersteunen die automatisch aangepast worden aan de context, heeft men een onderscheid gemaakt tussen abstracte en concrete aspecten in modellen. Abstracte elementen zoals gebruikerstaken in taakmodellen zijn altijd van toepassing in elke omgeving. Deze abstracte elementen of 'Abstract Interaction Objects' (AIO's) zullen niet veranderen naargelang de context.

Aan de andere kant heeft men vastgesteld dat er in deze modellen ook concrete elementen gebruikt worden die afhankelijk zijn van een technische omgeving waarin de user interface gebruikt wordt. Deze moeten op een ondubbelzinnige manier gespecificeerd worden, waardoor het design minder flexibel wordt.

Dialoog- en presentatiemodellen zijn meestal zeer expliciet maar noodzakelijk. Daarom tracht men de abstracte elementen te scheiden van de concretere elementen (XiML, SeescoaXML) en de abstracte modellen te mappen op concrete modellen. Dit blijkt het meest kritische aspect te zijn bij deze abstracte talen en is nooit bevredigend op elk vlak. Dit heeft in eerste instantie al te maken met het feit dat men modellen gebruikt die zowel concrete als abstracte elementen bevatten. Hierdoor gaan bepaalde concrete implementatie afhankelijke beslissingen en attributen bepaald moeten worden in abstracte modellen alvorens men de concrete modellen kan afleiden. Dit wordt het mapping probleem genoemd [23, 54]. Een goede abstracte user interface beschrijvingstaal zal andere aangepaste modellen moeten integreren dan de reeds gekende modellen om flexibel te zijn.

We merken op dat modelgerichte oplossingen die een hoge expressiviteit hebben en veel controle bieden over het uitzicht en andere eigenschappen van de user interface, meerdere lagen van abstractie en mappings hanteren (XiML, Seescoa). Ze vereisen complexe beschrijvingen voor uitgebreidere user interfaces (Seescoa) of betrekken een groot aantal modellen (XiML). Modeloplossingen met een simpele uitdrukkingswijze die een geautomatiseerde configuratie en implementatie ondersteunen (UIML), hebben 1 abstractieniveau en beperkte mappingsmogelijkheden maar zijn hierdoor efficiënt in gebruik.

Een andere eigenschap van automatisch gegenereerde user interfaces is dat ze meestal minder creatief ogen en dezelfde 'look-and-feel' hebben [24]. Er blijkt een onvermijdelijke afweging gemaakt te worden tussen complexiteit en expressiviteit bij beschrijvingstalen. Tot nu toe is er nog geen enkel model gekend met flexibele, praktische en voor designers begrijpbare mappings. Het ziet er naar uit dat een user interface beschrijvingstaal de ruimte zou moeten bieden aan designers om zelf de keuze te kunnen maken in welke mate men controle over het uitzicht en presentatie wil hebben. Hiermee is het toch nog mogelijk om creativiteit in de user interface te steken en de complexiteit zelf te bepalen (Seescoa). XiML lijkt op gebied van zowel expressiviteit en toepassingsdomein het meest uitgewerkt. Het betrekt veel maar relatief eenvoudige modellen die we herkennen in veel klassieke onuitvoerbare designmodellen.

UIML verschilt van andere HLUIDs en frameworks doordat het een meta-taal is om user interfaces te beschrijven, en doordat de UIML architectuur de elementen niet scheidt van hun presentatie, iets wat de voorgaande besproken talen wel doen. XiML, Teresa en Seescoa zijn meer taakmodel-geöriënteerd terwijl UIML eerder een patroongedreven oriëntatie (pattern-driven) heeft [24]. De adaptatiemodellen uit deze talen kunnen dus niet zomaar overgenomen worden in UIML als men het UIML framework niet drastisch wil aanpassen.

3 Vereisten voor intelligente en kwalitatieve adaptatie

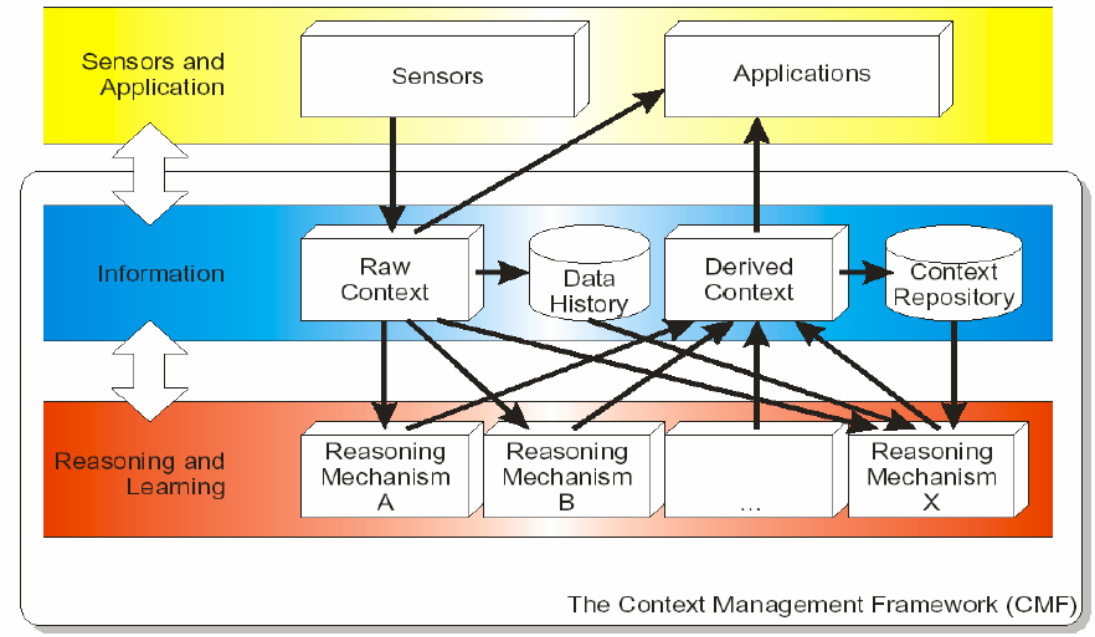
In dit onderdeel onderzoeken we de ‘gereedschappen’ die nodig zijn om de adaptaties die we willen uitvoeren op UIML user interfaces, te ondersteunen. Deze ‘gereedschappen’ bestaan uit *context sensors*, een *redenerings- of reasoningcomponent*, en een *uitvoerbaar transformatieproces* waarmee de adaptatie effectief uitgevoerd wordt [25]. De vereisten waar deze aan moeten voldoen om ons doel te bereiken worden vastgelegd. We gaan hierbij werken met het oog op automatische adaptatie.

3.1 Probleemsituering

Zoals we tot nu toe aangenomen hebben, zijn (aanpassingen in) design en implementatie geïnspireerd door (vereisten of requirements van) de context waarin de toepassing gebruikt wordt. Om het design automatisch te adapteren, is deze informatie eveneens noodzakelijk, alsook de informatie die we bij handmatige (adaptatie in) designs nooit opschrijven maar wel gebruiken.

Dergelijke informatie wordt beschreven aan de hand van gebruikersmodellen, een requirementslist, een applicatiemodel, etc., en wordt gemaakt om efficiënt en ondubbelzinnig geïnterpreteerd te worden door de mens. Men kan nu al opmerken dat deze onbruikbaar zijn om een automatisch gegenereerde adaptatie/transformatie te ondersteunen, zonder dat het adaptatiesysteem toegang heeft tot deze modellen. De modellen zullen dus op uitvoerbaar niveau gebracht moeten worden, zodat ze door het systeem geïnterpreteerd kunnen worden.

Een voor de handliggende vraag is: Hoe gaan we deze informatie voorstellen op een manier, die voor de machine begrijpbaar en ondubbelzinnig is? Hoe gaat het systeem deze verwerken? Uiteraard zijn er nog vele andere vragen die men zich kan stellen. Is het praktisch haalbaar om adaptatie volledig automatisch te laten uitvoeren? In de ideale oplossing zal het systeem zelf de context ‘meten’ (met sensors) en hieruit autonoom redeneren (‘reasoning’), waarop deze volledig autonoom een geschikte hieraan aangepaste user interface genereert (transformatie). Dit framework zou volgens onze opgestelde benodigde gereedschappen er als volgt uit zien (zie figuur 10).



Figuur 10: The Context Management Framework

Doordat het handmatig design al een ingewikkelde materie kan zijn met problemen die daarmee gepaard gaan, en waarbij het resultaat soms nog te wensen over laat, lijkt het zeer onwaarschijnlijk dat een dergelijk systeem even kwalitatieve resultaten kan bereiken.

Een kosten- batenafweging zal de voorkeur geven aan een handmatig design en transformatiemechanisme indien we met deze aanpak enige waardevolle resultaten willen behalen. Om te onderzoeken tot welk niveau we efficiënt adaptatie kunnen uitvoeren gaan we in de volgende 3 delen ook een kleine studie doen naar de huidige stand van zaken omtrent deze technieken..

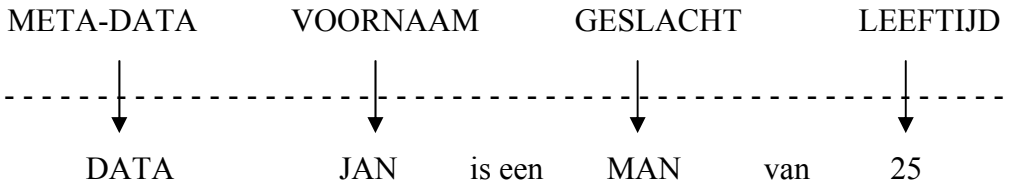
3.2 Integratie van de context (door metadata)

In voorgaande delen hebben we geconcludeerd dat adaptatie in functie van de context dient te gebeuren om zinvol te zijn. Om doeltreffende en intelligente adaptatie te ondersteunen dient het systeem toegang te hebben tot deze context. Er bestaan architecturen zoals [41] en [42] om contextbewuste applicaties te ontwikkelen en tools [43] om veranderingen in de context te detecteren. Het probleem hiermee is dat ze gemaakt zijn voor gebruik bij de ontwikkeling van een bepaalde applicatie, waarvan vele eigenschappen in de designfase gekend zijn. UIML is een meta-taal, we weten niet welke (soort) user interfaces ermee gaan ontwikkeld worden en welke contextinformatie invloed zal hebben. Informatie van een lichtsensor kan gebruikt worden om informatie te bekomen over de helderheid van de omgeving, maar het verband met een bepaald onderdeel of klasse van de UIML specificatie (als deze al bestaat) ligt niet op voorhand vast. Ook zal de manier waarop men de context interpreteert verschillen (licht/donker , aantal lumen, dag/avond/nacht, context1/context2,…). Dit moet de auteur van het UIML document zelf kunnen bepalen. Toch zal de context op een uniforme manier moeten voorgesteld worden zodat UIML als een universele taal kan blijven gebruikt worden en men daarbij eender welk contextmodel kan gebruiken (gebruikersmodel, platformmodel,…). We gaan in dit onderzoek in op de manier waarop de context voorgesteld wordt en gaan er vanuit dat deze informatie voor handen is, omdat de manier waarop de contextinformatie verkregen is per geval verschilt. In sectie 2.1.1 hebben we gesteld dat de context voor een groot deel informatie bevat over de buitenwereld, maar ook over de user interface of applicatie zelf. Om adaptaties op UIML gebaseerde user interfaces toe te passen is het ook nodig om het systeem te verschaffen van relevante contextgevoelige informatie over de onderdelen die in het UIML document beschreven worden. In dit deel bespreken we manieren om deze meta-informatie toegankelijk te maken voor het systeem en er andere informatie uit af te leiden.

3.2.1 Waarvoor wordt metadata gebruikt

Definitie van data: gegevens die (meestal) informatie uitdrukken, bevatten of waar informatie uit afgeleid kan worden. Data wordt gebruikt om informatie op te slaan/te noteren. “Paris Wed - PM - 26°C” is data, dat in Parijs in de woensdagnamiddag een temperatuur van 26 graden Celsius heerst of wordt verwacht is de informatie die het uitdrukt. Als er naast deze data ook nog “Belgium Wed – PM - 13°C” als data beschikbaar is, bevatten deze gegevens ook de informatie dat het woensdag warmer gaat zijn in Parijs dan België.

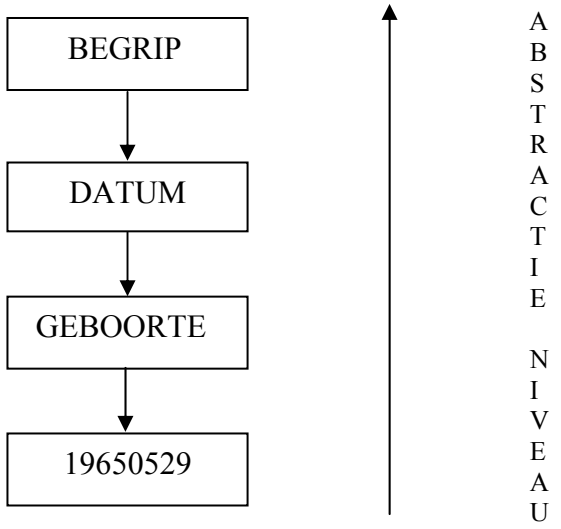
Metadata wil zeggen: data over data. Beschrijvende informatie dat iets zegt over andere informatie.



Figuur 11: Illustratie van metadata

Data over een persoonsbeschrijving zou kunnen zijn: Jan is een man van 25. De metadata hiervan beschrijft dat ‘Jan’ een voornaam is, dat ‘man’ gaat over het geslacht en ‘25’ slaat op de leeftijd. Deze (meta-)informatie is onmisbaar om de informatie juist te interpreteren. Men kan metadata bekijken op alle niveaus en met metadata andere metadata beschrijven. Als men over de gegevens: ‘19650529’ en ‘20070222’ beschikt, is het essentieel te weten dat de eerste en tweede reeks van 8 cijfers een datum weergeven (waarbij de eerste 4 het jaar voorstellen, de volgende 2 de maand en de laatste 2 de dag). Verder dienen we ook nog te weten dat de eerste datum een geboortedatum betreft en de 2e datum de sterfdatum, en niet de trouwdatum. Anders is deze data waardeloos en zijn ze niets meer dan een reeks cijfers.

De abstractielagen zijn voor de duidelijkheid in het voorbeeld niet in volgorde uitgelegd. Het metamodel dat hierbij hoort is:



Figuur 12: Abstractie niveaus van metadata.

Dezelfde data kan als metadata beschouwd worden in een andere abstractielaag en context. Bijvoorbeeld: De persoon kwam zich gisteren aanmelden. In dit geval zijn de gegevens: ‘Jan’, ‘man’ en ‘25’ de metadata over 'de persoon' en kunnen van essentieel belang zijn om de informatie zinvol te gebruiken.

In de informatica wordt metadata gebruikt om bijkomende informatie te beschrijven over de data die verwerkt wordt en waar het uiteindelijk om te doen is. Meestal wordt deze metadata gebruikt om te vertellen wat de eigenlijke data beschrijft en om welke soort data het gaat. Bij het streamen van een live concert op een computer wordt er metadata geleverd die vertelt welke stream het geluid bevat en welke de videobeelden bevat, samen met de codec die gebruikt is en nodig is om de stream te kunnen afspelen.

Zoals in de inleiding van dit hoofdstuk is vermeld, hebben we voor intelligente adaptatie ook bijkomende informatie nodig over data (modellen, context, toolkits,...). Om te onderzoeken welke data we gaan opslaan en gebruiken en tot op welk niveau we deze kunnen gebruiken, doen we in dit hoofdstuk een kleine studie over technieken om deze informatie te beschrijven.

3.2.2 Voorstelling van metadata

Modellen worden opgesteld aan de hand van informatie die we vergaren tijdens een onderzoek of domeinstudie alvorens deze worden opgesteld. De keuze van opstelling en invulling van attributen van deze modellen vereisen ook kennis over de definitie en methodologie van deze modellen. Ook implementatie beslissingen hierop vereisen zoals eerder besproken transformatiemodellen of in het geval van handmatige transformatie, kennis over niet-functionele attributen van een implementatie. Bij automatische adaptatie zal deze informatie ook nodig zijn omdat de modellen niet constant blijven en in de tijd gewijzigd moeten worden. Hierdoor zijn een reeks aanpassingen noodzakelijk in de lagere lagen. Er dient bijkomende informatie te worden voorzien aan het systeem om de adaptatie tot een goed einde te brengen. In deze sectie wordt besproken op welke manier (meta-) data kan worden opgeslagen en de verschillende standaarden die gebruikt worden om andere data of objecten te beschrijven.

3.2.2.1 XML

eXtensible Markup Language (XML) [26] is een standaard voor het definiëren van formele markup-talen voor de representatie van gestructureerde gegevens in de vorm van platte tekst. Deze representatie is zowel machineleesbaar als leesbaar voor de mens. Hiervoor wordt gebruik gemaakt van 'tags' die in een hiërarchische manier worden genest in andere tags.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<playlist name="mylist">
 <song>
  <title>Little Fluffy Clouds</title>
  <artist>the Orb</artist>
 </song>
 <song>
  <title>Goodbye mother Earth</title>
  <artist>Underworld</artist>
 </song>
</playlist>
```

Figuur 13: XML beschrijving.

Met andere woorden: XML is een bepaalde manier om gegevens gestructureerd vast te leggen. Deze manier is gedefinieerd en mag iedereen gebruiken. Het is ontworpen om zowel door een programma als door een mens leesbaar te zijn. XML is niet alleen geschikt om gegevens in op te slaan, maar wordt veel gebruikt om gegevens via het internet te versturen.

XML is niet meer weg te denken in veel applicaties en wordt zelfs gebruikt om transformaties uit te drukken (XSLT) waarbij men van het ene XML-gebaseerde formaat converteert naar een ander XML-gebaseerd formaat, bijvoorbeeld van HTML naar WML. Ook commando's kunnen worden verzonden in XML-formaat. Er zijn ontelbare definities voor xml-afgeleide talen die soms worden gebruikt door 1 specifieke applicatie. Het principe is echter altijd hetzelfde.

3.2.2.2 RDF

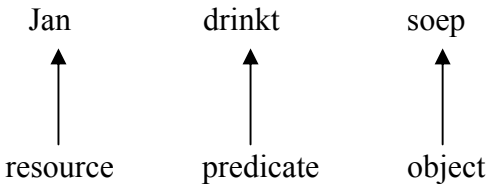
Algemeen

RDF of Resource Description Framework [28] is een model om gegevens in het algemeen voor te stellen en uit te wisselen. Met RDF beschrijft men eigenlijk een hele reeks beweringen (statements).

Een bewering bestaat uit een:

- onderwerp (subject)
- predicaat (predicate)
- object (object)

Het onderwerp wordt door middel van het predicaat in verband gebracht met het object. Een statement beschrijft dus een eigenschap van het object. Als naam van een onderwerp, predicaat of object wordt meestal een verwijzing of URI (Uniform Resource Identifier) gebruikt naar het onderwerp of document dat gaat over dat onderwerp. Een object kan echter ook een waarde bevatten.

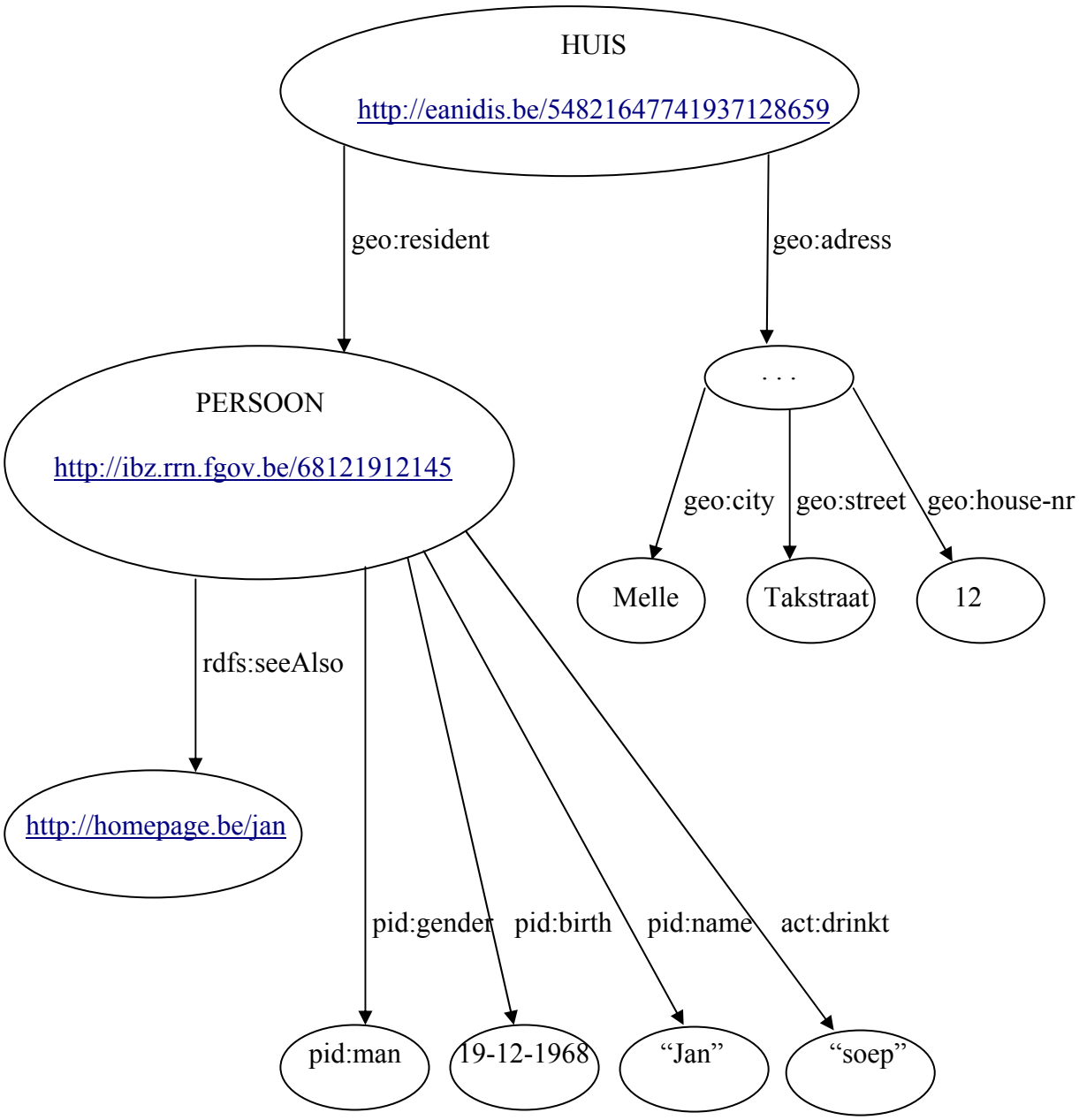


Figuur 14: RDF principe.

Dankzij deze verwijzingen kan men met RDF relaties tussen onderwerpen uitdrukken die zich op een andere plaats bevinden en totaal geen fysieke link met elkaar hebben. Op deze manier kan men (onderdelen van) webpagina's met elkaar linken die inhoudelijk geen hyperlink naar elkaar bevatten. Het Semantic Web [29] is een uitbreiding op het wereld wijde web waardoor (zoek)machines weten waarover (de inhoud van) webpagina's gaan en hiermee intelligentere zoekopdrachten kunnen verwerken. Zoals we dit in de Nederlandse taal tegenkomen kan een onderwerp in een RDF statement een object zijn in een ander statement. Op deze manier vormen verschillende statements een gerichte graaf (zie figuur 15). Hierdoor kunnen we expliciete beweringen opstellen over eender welk onderwerp en de machine duidelijk maken

- dat dit beweringen zijn
- hoe ze gerelateerd zijn aan elkaar

Nu rest nog de vraag hoe we deze onderwerpen, predicaten en objecten gaan benoemen. Gaat men de term 'land' of 'staat' gebruiken? Hiervoor gebruikt men RDF-schema's waar men een vocabularium kan definiëren die eveneens met RDF wordt beschreven. Ook de relaties tussen de gedefinieerde termen worden met RDF-schema beschreven. Er zijn verschillende standaard RDF-schema's beschikbaar zoals bv. Dublin Core [30], dat gebruikt wordt om informatie over boeken te beschrijven. Men kan het vergelijken met een woordenboek, men is vrij te kiezen welke termen men gebruikt. Er zijn meerdere syntaxen om 'dingen' of bronnen te beschrijven in RDF zoals RDF/XML, N3, Turtle, Ntripels, Trig, Trix. In feite kan men eender welk formaat gebruiken of uitvinden en kan men eenvoudig conversies maken tussen 2 syntaxen.



Figuur 15: RDF graph.

OWL

Tot nu toe kunnen we met RDF iets beschrijven op een door een machine begrijpbare manier en op eender welk niveau van abstractie. De informatie voor de machine is in dit geval beperkt tot hetgeen we beschrijven, zoals bij XML het geval is. Met OWL (Web Ontology Language) maken we beweringen d.m.v. axioma's (eveneens in RDF) over de termen en relaties die in RDF-schema beschreven staan. Deze axioma's kunnen onder andere transities, gelijkheden, negaties, booleaanse expressies en beperkingen voorstellen.

Dankzij OWL , dat voortgevloeid is uit de ontologische talen DAML en OIL [31], kan een systeem redeneringen maken uit RDF beschrijvingen. Uit het gegeven dat Veerle een dochter is van Jan, kan het systeem meer informatie afleiden dan het gegeven zelf, bijvoorbeeld dat Jan de vader is van Veerle; en dat Juul een voorouder is van Veerle als het over het gegeven beschikt dat Juul de vader is van Jan (transitie). Door een uitgebreide ontologie in OWL op te stellen kan men een schat aan informatie bekomen die niet expliciet zijn ingevoerd in het systeem. Wanneer we met RDF een context beschrijven kan men met behulp van OWL bijvoorbeeld eisen afleiden en bepalen of er aan

deze eisen voldaan is. Om deze informatie er uit halen en het systeem duidelijk maken wat we willen weten, kan men gebruik maken van ondervragingstalen zoals SERQL, RDQL, en SPARQL [32].

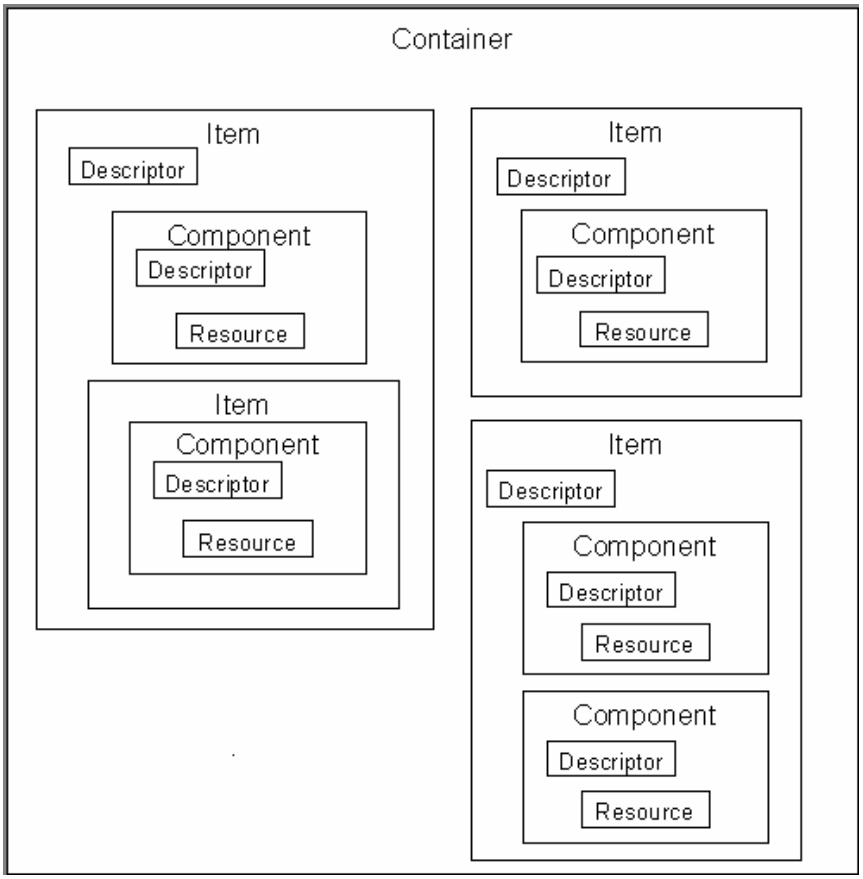
Implementaties van het RDF-framework zijn o.a.: Jena (java) die het meest geavanceerd is door RDF, RDFS, OWL en SPARQL ondersteuning [33], SESAME voor java [34], Raptor, een library in C [35], en SemWeb, geprogrammeerd in C# [36].

3.2.3 Toepassing van metadata

In dit deel worden enkele belangrijke toepassingen van metadata beschreven vanuit het oogpunt van bruikbaarheid voor intelligente adaptatie van user interfaces.

3.2.3.1 MPEG-21

MPEG-21 [37] is een standaard (framework) ontwikkeld door de Moving Picture Expert Group die de verzending van een breed scala aan multimediaformaten over netwerken en de weergave op verschillende apparaten moet vergemakkelijken. Het framework bestaat uit 9 onderdelen waarvan deel 2: Digital Item Declaration en deel 7: Digital Item Adaptation het meest interessant is met betrekking tot deze thesis. In deze onderdelen gaat men multimedia voorstellen als 'Digital Items' en de inhoud ervan beschrijven doormiddel van de XML-gebaseerde Digital Item Declaration Language.

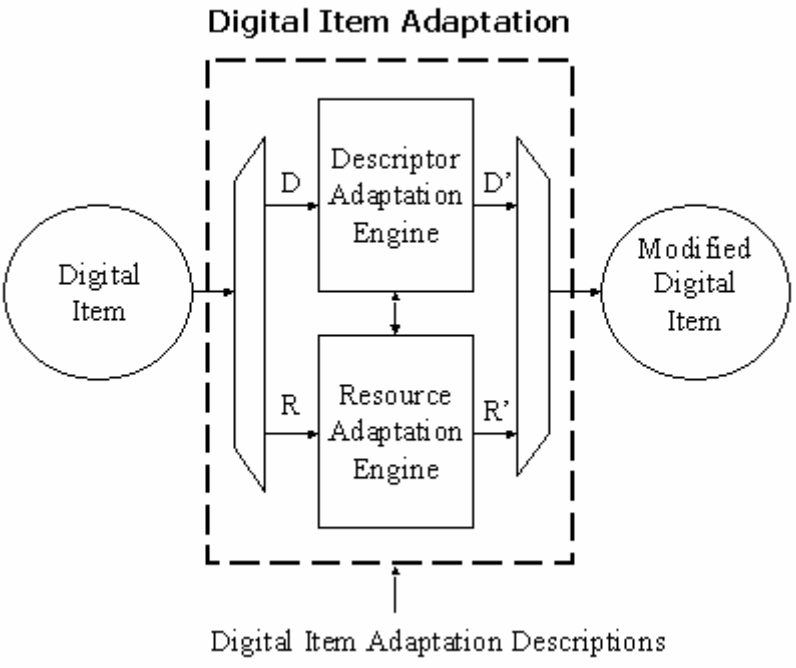


Figuur 16: MPEG-21: Digital Item [37].

Hiermee kunnen verschillende aspecten van (multimedia)bestanden, onderdelen van bestanden, en verzamelingen van bestanden beschreven worden. Dit concept zorgt ervoor dat alle multimediacontent op een universele manier kan benaderd worden (als 'Digital Items'), zonder dat

de gebruiker hoeft rekening te houden met het formaat, gebruikte codecs en context. Het Digital Item Adaption onderdeel van het MPEG-21 framework voorziet een aantal tools die gebruikt kunnen worden om een digital item (de content die het beschrijft) te adapteren aan de hand van Digital Item Adaption beschrijvingen (zie figuur 17). Deze beschrijvingen drukken de relatie uit tussen gebruikerscontext en inhoudskarakteristieken. Applicaties die deze beschrijvingsstandaard gebruiken kunnen zo de multimedia inhoud uitwisselen en optimaliseren naar gelang de context. Om deze context te beschrijven voorziet MPEG-21 Usage Environment Descriptors die metadata bevatten van netwerk- en apparaatkarakteristieken, voorkeuren van de gebruiker en natuurlijke omgevingskarakteristieken. Een verlaging van bandbreedte in het netwerk, beperkte schermgrootte en afwezigheid van audio weergave van een apparaat kan in rekening worden gebracht met de adaptatie. De verzendende applicatie zal bijvoorbeeld een hogere compressie toepassen of een lagere framerate gebruiken en geen geluidstroom bijvoegen wanneer de multimedia wordt verzonden. Een toestel zonder videomogelijkheid die hetzelfde 'Digital Item' opvraagt zal bijvoorbeeld een afbeelding of screenshot te zien krijgen.

Digital Item Adaption van MPEG-21 is gemaakt op modulair niveau en is protocol- en applicatieonafhankelijk, waardoor het niet beperkt is tot multimediale inhoud. Dit maakt deze specificatie interessant omdat het ook in andere toepassingsdomeinen zou kunnen gebruikt worden, bijvoorbeeld in user interfaces [58]. Door user interfaces en hun componenten als digital items te beschouwen kunnen we met een gelijkaardige aanpak technische maar ook semantische aspecten ervan beschrijven, en in verband brengen met de aanwezige karakteristieken van de context.



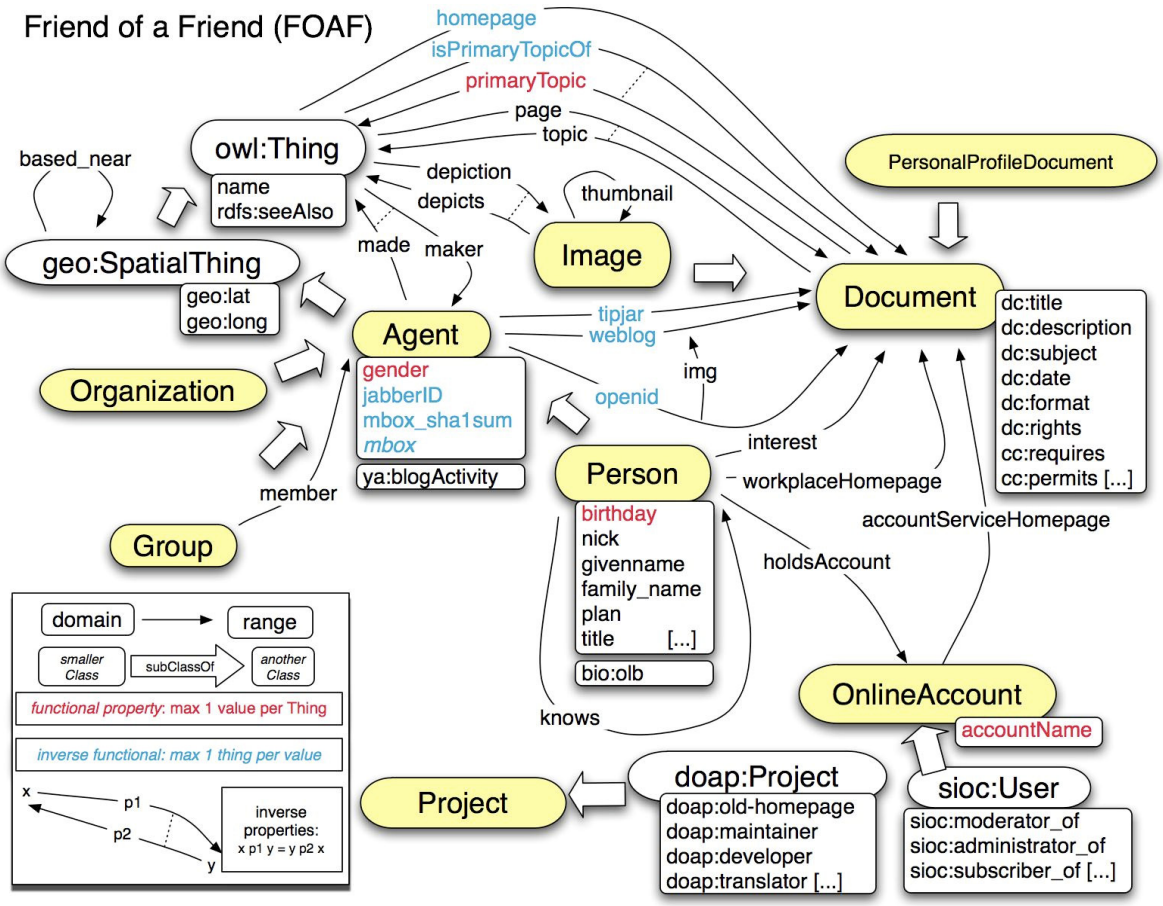
Figuur 17: Digital Item Adaptation principe in MPEG-21 [20].

3.2.3.2 FOAF

Friend of a Friend (FOAF) [59] is technisch gezien een RDF-vocabularium (RDF Schema) dat gebruikt wordt om relaties tussen mensen te beschrijven.

Door FOAF-beschrijvingen van jezelf (foaf-profiel) op je website te zetten, kunnen zoekmachines een sociaal netwerk afleiden tussen je website en andere websites die anders totaal los staand van elkaar lijken te zijn (zie figuur 18). Op gebied van adaptaties in user interfaces kan het foaf-principe enkel een meerwaarde betekenen op het gebied van afleiden van eisen en

voorkeuren. Dit gebeurt op basis van informatie die niet rechtstreeks in het profiel van de gebruiker vervat zit. Een eenvoudige toepassing is bijvoorbeeld het instellen van de standaardtaal op de landstaal van het land waar de gebruiker woont, zonder dat de gebruiker een voorkeurstaal heeft opgegeven. Dit staat niet expliciet in zijn of haar profiel vermeld. De volledige foaf namespace specificatie kan men vinden op http://xmlns.com/foaf/spec/ .



Figuur 18: Visualisatie van een FOAF beschrijving [27].

3.2.3.3 GUMO

GUMO (General User Modelling (and Context) Ontology) [60] is een ontologie beschreven in OWL en gaat over classen, predicaten en instanties betreffende de staat (situatie) en modellen van de gebruikers, systemen, apparaten en hun omgeving. Deze ontologie laat toe deze in verschillende dimensies te beschrijven. Zo kunnen personen aan de hand van oa hun sociale-, fysieke- en geografische omgeving beschreven worden maar ook hun fysische eigenschappen, gemoedstoestand, interesses, kennis, vaardigheden, relaties... enz. Kortom, op elke manier waarop men dit in gewone spreektaal doet. GUMO wordt gebruikt in het UbisWorld project [61] dat nog steeds in ontwikkeling is en verschillende ontologiëen combineert om zo alle objecten en begrippen die men in het echte leven tegenkomt, voor te stellen. Naast GUMO gebruik het UbisWorld project ontologiëen over fysische objecten, ruimtelijke elementen, temporele elementen, gebeurtenissen en interferenties, en is hiermee een zeer uitgebreide verzameling van ontologieën beschreven in OWL. Ook bij de ontwikkeling van user interfaces dienen we rekening te houden met karakteristieken van de gebruiker. Gebruikersmodellen beschrijven op een universele manier, zoals in GUMO, kan een oplossing bieden voor de automatisering van adaptatie.

3.2.3.4 Andere soorten

Metadata wordt gebruikt in ontelbare toepassing en op verschillende niveau's. Op een cd-rom kunnen we al volgende toepassingen van metadata terugvinden, vertrekkend van hoogste abstractieniveau: data van sectoren op cd's, blockdata van bestandsystemen, eigenschappen van bestanden, informatie over documenten enz.
Andere toepassingen zijn:

- ID3-tags in mp3-bestanden : Beschrijven titel, uitvoerder, genre enz. van een muziekfragment
- MPEG-7 [62]: beschrijft informatie over de inhoud van een multimediafragment. Dit kan ook berekende informatie zijn zoals veranderingen in toon bij muziek, bewegingen in videofragmenten, kleurenspectrum bij afbeeldingen, ... Op deze manier kan gelijkaardige multimedia worden opgezocht en tekstuele zoekopdrachten uitgevoerd worden met betrekking tot de inhoud van multimedia.
- Aan de hand van RFID tags en barcodes wordt extra informatie aan een fysiek object gelinkt (zoals de beschrijving en prijs van een product in warenhuizen).

3.2.4 Conclusie

Er lijken geen beperkingen te zijn op het soort objecten die we kunnen beschrijven met metadata op een toegankelijke manier voor een computersysteem, en het niveau waarop we dit doen. Door objecten van metadata te voorzien kan een systeem over meer informatie beschikken, die het niet kan afleiden uit het object zelf. Dit heeft 2 grote voordelen: 1) een digitaal object krijgt meer betekenis en is geen 'black box' meer voor het systeem. 2) Het systeem kan nu ook objecten benaderen (metaobjecten) waartoe het in de realiteit geen toegang heeft zoals personen of fysische objecten en aspecten in de buitenwereld. Metadata kan de kloof tussen de machinelogica en humane aspecten van user interfaces verkleinen en adaptaties mogelijk maken waarbij rekening wordt gehouden met deze humane aspecten.

Concreet in het domein van adaptatie in user interfaces kunnen we stellen dat gebruik van metadata een doorbraak betekent op 2 punten:

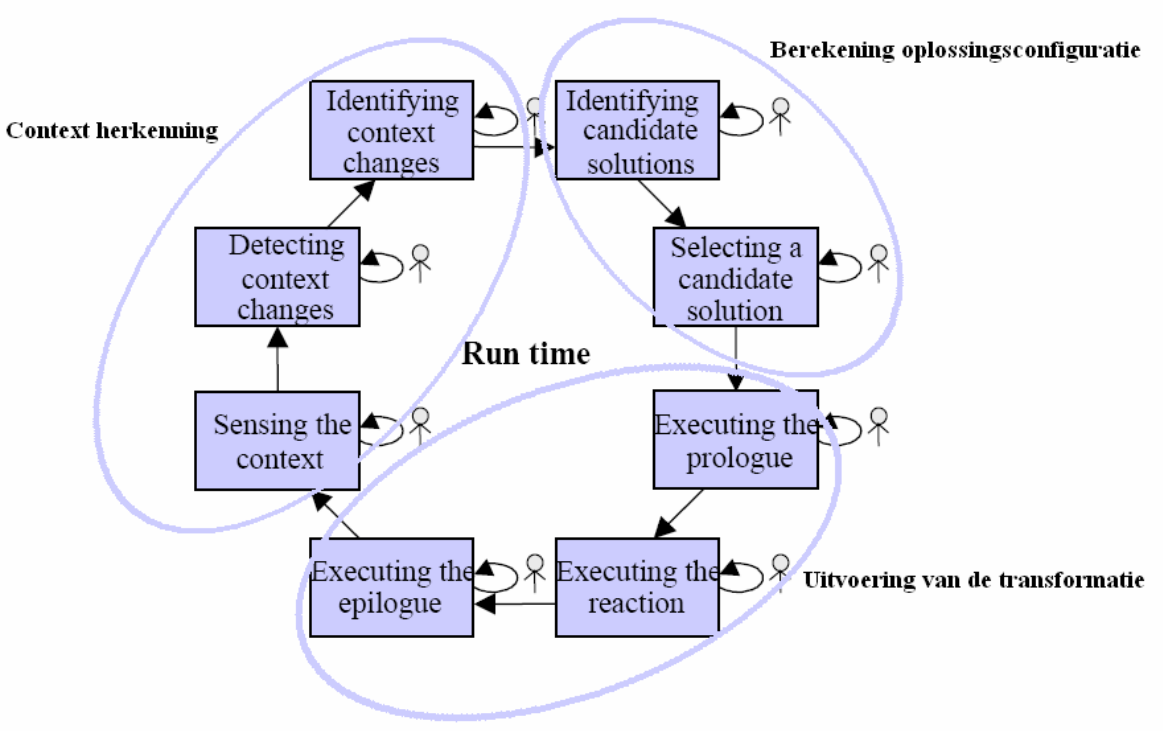
- Door metadata op te slaan van user interfaces en hun componenten, beschikt het systeem naast de implementatie ook over semantische informatie over deze componenten zoals hun functie, uiterlijke kenmerken, doel, gebruik,... en deze gaan herkennen op de manier zoals een mens dit doet. Dankzij deze informatie kan adaptatie meer algemeen worden toegepast.
- De voorstelling van personen, context en andere begrippen uit onze natuurlijke omgeving kan de kloof tussen domeinstudie en het maken van modellen dichten. Systemen hebben als het ware toegang tot de buitenwereld en kunnen aspecten uit de huidige natuurlijke omgeving als input gebruiken voor de modellen, iets wat vroeger door designers werd gedaan. Dit zorgt ervoor dat adaptatie automatisch kan worden toegepast.

3.3 Redeneren vanuit de context

We hebben aangetoond dat context kan benaderd worden vanuit de applicatie en mogelijkheden om deze te beschrijven besproken. Een volgende stap bestaat er uit iets nuttigs te doen met deze contextinformatie. Tenslotte is het de bedoeling dat deze bepaalt hoe en welke adaptatie wordt toegepast.

3.3.1 Welke contextinformatie gebruiken

Tijdens de hele ontwerpcyclus van user interfaces wordt in elke fase contextinformatie uit verschillende domeinen gebruikt om beslissingen te ondersteunen. Welke informatie precies relevant is hangt af van de fase waarin men zich bevindt in de ontwikkelingscyclus en het model of aspect waaraan men op dat moment werkt.



Figuur 19: Fases in de adaptatiecyclus.

Attributen van invoerapparaten zullen het interactiemodel in de designfase en de interactietechniek in de implementatiefase beïnvloeden, terwijl ze geen invloed hebben op lexicale- of stijlattributen van de presentatie. Als designer hebben we hier onze eigen regels voor of gebruiken deze van de vele al dan niet geformaliseerde designprincipes en frameworks. Het is dus belangrijk voor het systeem om te weten waarvoor een bepaald attribuut van de context gebruikt kan worden. Om dit te kunnen dient het systeem over het type kennis te beschikken waarmee die bepaalde attribuutwaarde kan gebruikt worden. We kunnen ons hierbij baseren op handmatige adaptatie in design: algemeen gaat men eerst de vereisten (requirements) opstellen van de user interface, vervolgens een modeloplossing uitwerken, en deze tenslotte realiseren. Bij automatische adaptatie kan men in het algemeen ook deze stappen herkennen: het systeem verkrijgt de vereisten uit de context, leidt een of meerdere kandidaatconfiguraties af die aan deze eisen voldoen, en voert tenslotte (eventueel) de transformatie uit om de meest geschikte configuratie te bereiken [48]. Voor elk van deze 3 stappen is andere contextinformatie nodig en wordt in de volgende delen besproken.

3.3.2 Context modelleringsregels

De eerste stap van elk intelligent adaptatieproces bestaat erin de vereisten uit de context af te leiden. Het systeem dient voorzien te worden van afleidingsregels, context modelleringsregels genaamd [50]. Met deze regels wordt er informatie afgeleid uit de contextinformatie, en dus kennis verkregen uit andere kennis. Dit wil meestal zeggen dat men de vereisten en andere parameters uit de context afleidt, bijvoorbeeld: gebruiker met visuele beperking → grote items nodig. Deze afleidingsregels zijn voornamelijk op kennis of ontologieën van gebruikersmodellen gebaseerd. De informatie die gebruikt wordt door deze afleidingsregels is veranderlijk per sessie en zullen uit een gebruikersprofiel kunnen afgeleid worden.

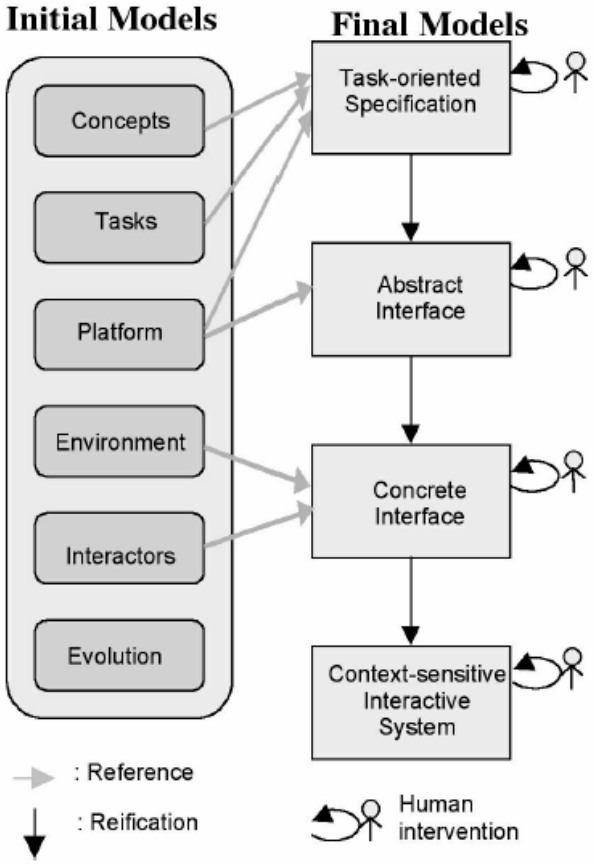
3.3.3 Adaptatieregels

Om uit de vereisten een oplossing te genereren dient een systeem te beschikken over: 1) de vereisten 2) relevante functionele en niet-functionele attributen van de user interface onderdelen 3) kennis om uit de eerste 2 gegevens te bepalen welke oplossing het meest geschikt is. Adaptatie- of reactieregels definiëren de strategieën van adaptatie en zijn gebaseerd op ontologieën over domeineigenschappen, adaptatiedoelen, gebruikersacties, context, en in- en uitvoerapparaten [50]. Dit hangt natuurlijk af van het soort adaptatie dat men toepast. Om adaptatie op alle aspecten van een user interface toe te passen zou men zich op al deze ontologieën moeten baseren.

Om een geschikte oplossing te gaan bepalen, gaan adaptatieregels attributen van de context mappen op attributen van de user interface. Beide maken onderdeel uit van de context. Welke contextinformatie gebruikt wordt zal naast het soort adaptatie dat men wil ondersteunen afhangen van het niveau waarop men wil adapteren: op hoog niveau waar algemene concepten worden gebruikt of op lager niveau waar men specifieke concepten gebruikt.

3.3.4 Waar en wanneer contextinformatie gebruiken

In frameworks en benaderingen voor adaptatieondersteuning wordt de context en de verschillende modellen (die voortgevloeid zijn uit de context) betrokken doorheen de hele ontwikkelingscyclus. Waar en wanneer men contextinformatie gaat betrekken in geautomatiseerde adaptatie hangt af van het abstractieniveau waarop men gaat adapteren. Zoals eerder gezegd is UIML een patroon gedreven taal (zie sectie 2.2.4) en hebben we niet veel keuze op dit gebied.



Figuur 20: Situering van verschillende modellen in de ontwikkelingscyclus [9].

Een user interface specificatie in UIML is eigenlijk al een vrij concreet model. Het is hierbij belangrijk te bepalen of de adaptatie op hetzelfde niveau blijft (AIO[1] → AIO) of een transformatie inhoudt van een hoger naar een lager niveau (AIO → CIO[2]). UIML suggereert een vertrekpunt op een eerder concreet niveau. We komen hier later op terug.

[1]AIO = Abstract Interactie Object , [2] CIO = Concreet Interactie Object

3.3.5 Redeneringsmechanismen

We hebben ook gezien dat kennis kan beschreven worden, bijvoorbeeld met OWL. In de toekomst zou dit kunnen gebruikt worden om kennis over design en modellen voor te stellen en deze te laten gebruiken door user agents. Hiernaast betekent dit dat we het design, de modellen en user interface specificaties in eenzelfde formaat kunnen gaan beschrijven. Zo zou men een UIML document ook met RDF kunnen beschrijven en kunnen adaptatiemodellen dynamisch worden ingeladen zonder de user interface agent te moeten aanpassen en te hercompileren. Dit concept zal nog verder onderzocht en getest worden voor dit in de praktijk te brengen met complexe kennis waar reeds moeilijkheden ervaren worden in zijn toepassing. De eerder rudimentaire prototypes in dit gebied [44, 45, 46] wijzen erop dat dit een krachtig maar complex concept is dat nog in zijn kinderschoenen staat. Adequate automatische redeneringen uit kennis voor contextbewuste applicaties vereist zeer uitgebreide ontologieën [47] terwijl men er nog niet in geslaagd is een reasoner te ontwikkelen die op een robuuste manier om kan met grote en complexe ontologieën [46].

3.3.6 Conclusie

Context moet in alle stappen van het adaptatieproces betrokken worden om intelligente en automatische transformatie van user interfaces te ondersteunen. Deze stappen zijn: modelleren van de context, oplossing bepalen door te redeneren uit de context, uitvoeren van de transformatie die de oplossing als resultaat heeft. In UIML verschilt de contextinformatie en soort adaptatie die men wenst te gebruiken per user interface doordat het een declaratieve syntax betreft. Doordat een reasoner die adaptatieregels afleidt uit kennis (ontologieën) complex is en veel moeite vraagt t.o.v. het resultaat, zal de auteur van een UIML document zelf adaptatieregels opstellen (of hergebruiken). Dit laat de auteur van een user interface toe de concepten die hij of zij wenst te gebruiken, vrij te kiezen door mappings op te stellen. Om deze toe te passen is het noodzakelijk de user interface te verrijken met meta-informatie over de context in diverse opzichten.

3.4 Uitvoerbare transformatiemodellen

Een laatste stap in een adaptatieproces is de uitvoering van de transformatie waarbij het resultaat een user interface is die voldoet aan de noden van de context. In het laatste deel van dit hoofdstuk bespreken we een benadering die we kunnen toepassen op UIML user interfaces.

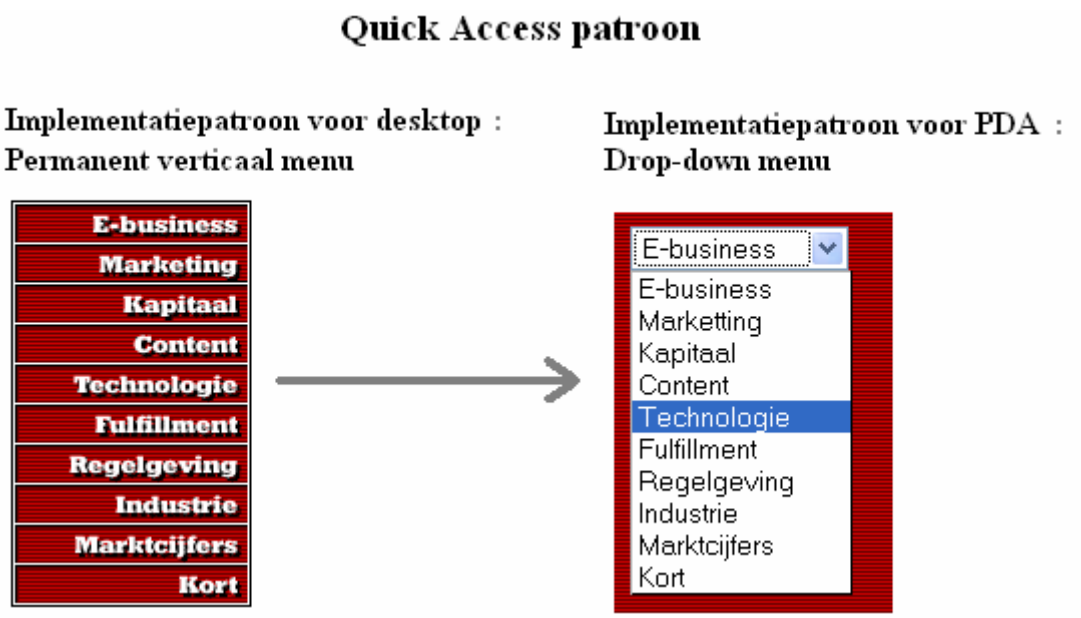
3.4.1 Gebruiksgerichte design patronen

Eerder hebben we impliciet al vermeld dat adaptatie en het beoogde resultaat gebaseerd is op gebruikersgerichte design kennis en principes. Het correct (laten) toepassen van deze principes is cruciaal voor zinvolle adaptatie. Het komt er op neer dat een user interface die vorm dient te krijgen die het zou hebben als er in het design proces werd uitgegaan van de huidige context. De logische daaruit volgende vraag: ‘Welke oplossingen zijn er om aan de gebruikersgerichte eisen te voldoen?’. We kunnen dit niet zomaar aan ‘creativiteit’ of ‘magie’ overlaten als het over automatische adaptatie gaat. We kunnen ons hierbij wel inspireren op HCI patronen [49, 51]. Een patroon is een algemeen herbruikbare oplossing voor een bepaald veelvoorkomend type ontwerpprobleem [52]. Patronen kunnen op diverse niveaus worden toegepast, ook op gebied van adaptatie. In de volgende delen onderzoeken we de bruikbaarheid van bestaande patronen voor adaptatie binnen UIML.NET.

3.4.2 Pattern mapping

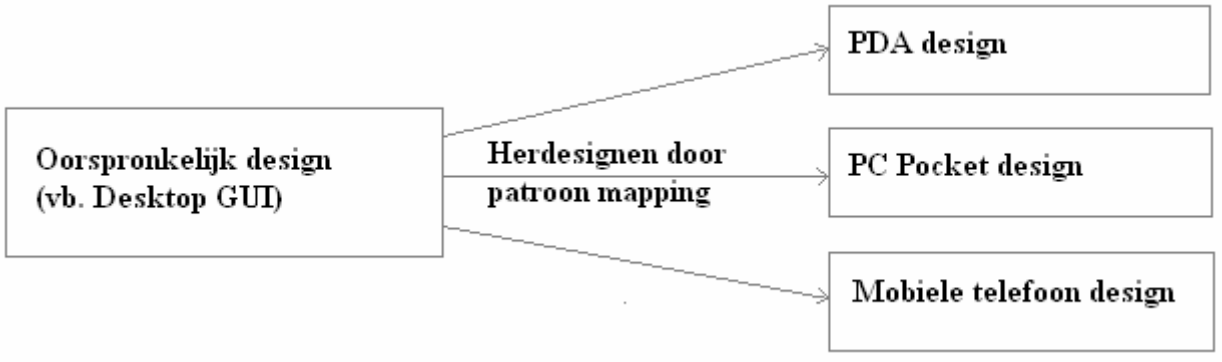
Op elk niveau kunnen er HCI patronen toegepast worden die geschikt zijn voor een eindplatform in een bepaalde context. In een UIML document zijn de patronen op hoog niveau hoogstwaarschijnlijk

al toegepast bij de creatie van het UIML document. We benadrukken nogmaals dat UIML als taal platform- en apparaatonafhankelijk is, maar niet per definitie de user interfaces. Tot nu toe is er ook geen gekende vocabulary voor UIML beschikbaar voor de creatie van implementatie-onafhankelijke user interfaces en het is ook niet de bedoeling van UIML om dit te doen. Bij de transformatie van een user interface naar een andere context worden de designpatronen op hoog niveau eveneens getransformeerd en dient een ander implementatie patroon op laag niveau gebruikt te worden die geschikt is voor de nieuwe context.



Figuur 21: Implementatie van het Quick Access patroon.

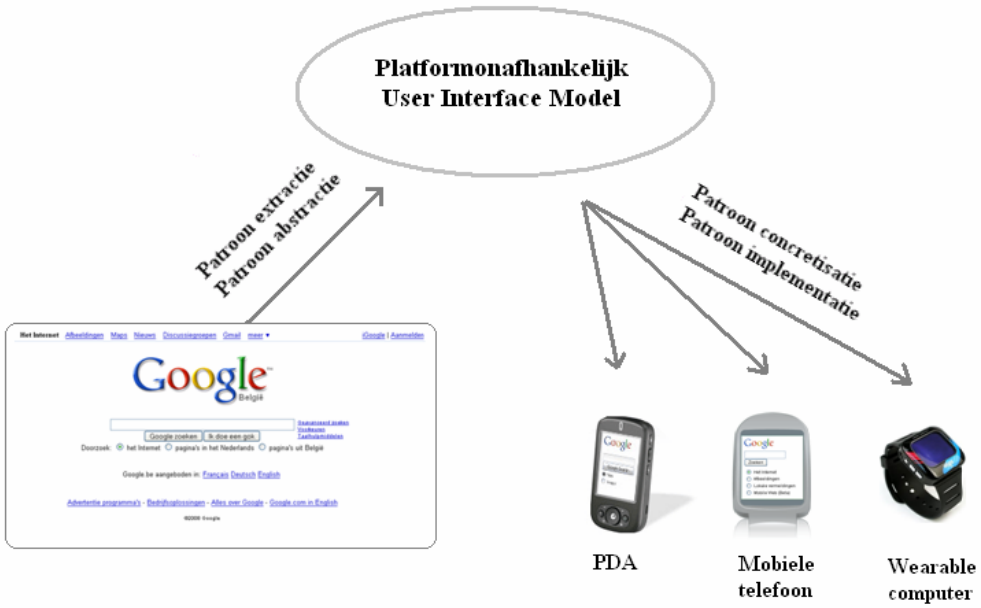
Ook voor de transformatie van een reeds opgesteld design bestaan modellen en patronen, 'pattern mapping' genaamd [64]. Deze kan men hanteren om het nieuwe design te laten voldoen aan de eisen van een andere context. Door deze adaptatie in het design verdwijnt de noodzaak om het design (geheel of gedeeltelijk) opnieuw op te stellen. Bij het opstellen van deze patroontransformaties is een grondig begrip van HCI patronen nodig, zodat de kennis hierover automatisch wordt toegepast bij gebruik ervan.



Figuur 22: Herdesigning van de user interface met patroon mapping , gebaseerd op [64].

Voordat we deze patroon gebaseerde aanpak gaan adopteren in onze toepassing, moeten we rekening houden dat het mappen van patronen tot nu toe werd toegepast om user interfaces te migreren naar verschillende eindplatformen (toestellen), en niet naar personen of gebruikssituaties. Stel dat we deze contexten gaan generaliseren en 5 archetypes gebruikers onderscheiden en 3

gebruikssituaties, zou het aantal benodigde transformatiemodellen met een factor 15 vermeerderen indien men elke context wil ondersteunen. In het geval er een design werd opgesteld voor een specifiek eindplatform, zal men over een daarvoor ontwikkeld transformatiemodel moeten beschikken om het design van dat platform te transformeren naar het gewenste eindplatform. Voor elke combinatie huidig platform –> nieuw platform is een apart transformatiemodel nodig dat de patronen - die gebruikt zijn voor het oorspronkelijke platform - transformeert naar patronen voor het nieuwe platform. De introductie van een platformonafhankelijk model kan heel wat kosten besparen. Hiermee hebben we per eindplatform (in ons geval: context) slechts één transformatiemodel nodig. In ons geval met UIML user interfaces vertrekken we van een platformspecifiek model, dus zouden we eerst dit model abstraheren als we vanaf een platformonafhankelijk model willen vertrekken [55]. Een niet zo praktische oplossing dus.



Figuur 24: Patroon geassisteerd re-engineering van de user interface, gebaseerd op [64].

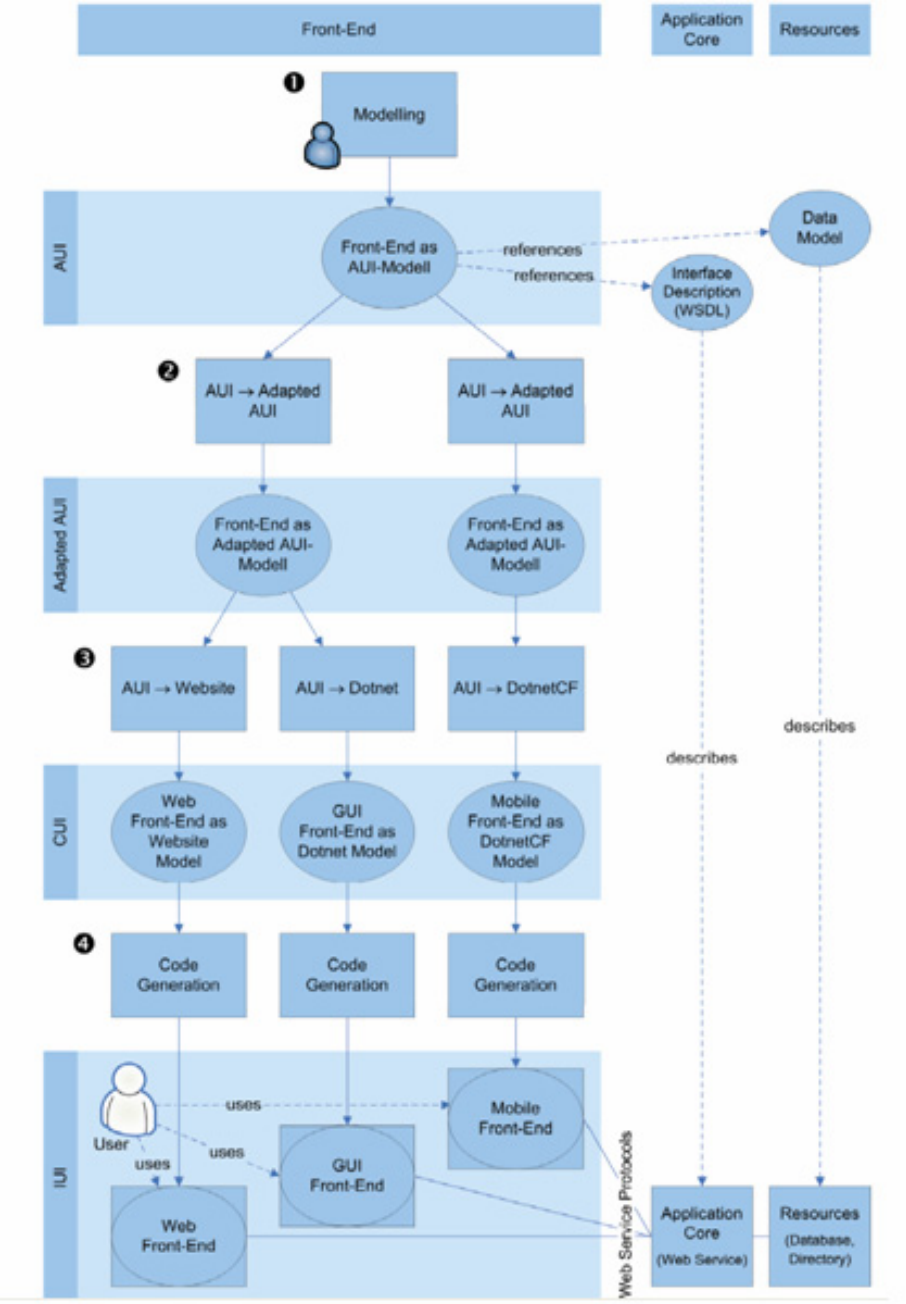
Conclusie

Om een patroongebaseerde aanpak te hanteren in adaptatie binnen UIML, zullen we te kampen krijgen met de volgende problemen:

- Patronen beschrijven een werkwijze van een bepaalde oplossing voor een specifiek probleem, en zijn geen uitvoerbare stukken software of algoritmen die in elke situatie werken. Dit is ook het geval voor transformatiepatronen. Voor geautomatiseerde adaptaties hebben we door programmatie uitvoerbare transformaties nodig.
- In UIML.NET en andere UIML implementaties worden de generische objecten at runtime gemapt op concrete widgets. Deze generische objecten kunnen we beschouwen als abstracte interactie objecten en widgets als concrete interactie objecten. Een widget kan ook beschouwd worden als een low-level implementatiepatroon. Patronen op hoger niveau zijn al toegepast door de auteur van het UIML-document. Om bevredigende adaptatie naar de context uit te voeren is de kans groot dat men ook abstractere patronen in rekening moet brengen. In een nieuwe context kan het immers aangewezen zijn een ander patroon te gebruiken en waarbij het huidige patroon niet geschikt is.
- De context in onze betekenis heeft geen discreet karakter zoals ‘eindplatformen’. De attributen van context kunnen ontelbare combinaties van waarden aannemen. Er is een techniek nodig om te bepalen wanneer een patroon geschikt is in een bepaalde context.

3.4.3 Modelgebaseerde transformaties

In dit deel gaan we op zoek naar strategieën om vooral de eerste 2 problemen in de conclusie van de vorige paragraaf aan te pakken, namelijk de algemene uitvoering van patronen, en transformaties op abstract niveau. Hierbij is het interessant om op te merken dat in ons geval van adaptatie de context niet op alle vlakken verandert: de taak van de user interface blijft hetzelfde, enkel de gebruiker- en apparaateigenschappen,.. kunnen veranderen. Abstracte high-level HCI patronen (vb. het shopping card patroon, navigatie patroon, view patroon,…) zullen in dezelfde toepassing waarschijnlijk de meest bruikbare patronen blijven, ongeacht de eindgebruiker en platform. Het implementatiepatroon voor het realiseren van een abstract patroon zal wel eerder afhangen van het eindplatform en de gebruikerskarakteristieken. Dit kunnen we afleiden uit tal van frameworks [65, 66, 67, 68, 69, 70] die zijn opgesteld voor de ontwikkeling van user interfaces die op meerdere platformen dienen gebruikt te worden, en veelal taakmodel gebaseerd zijn. Een rode draad die men terugvindt in deze frameworks is de abstracte modellering.



Figuur 25: Modelstroom in Model-based engineering of multiple interfaces with transformations [70].

Men vertrekt van een abstract user interface model dat wordt gebruikt voor alle eindplatformen. Men transformeert stap voor stap naar een meer concreet model. Platformonafhankelijke adaptaties worden op dit niveau (abstract user interface level) uitgevoerd. Met platformonafhankelijk bedoelt men aanpassingen in presentatie en dialoogstructuren die niet beperkt worden door de software, zoals bijvoorbeeld schermgrootte, ervaring van de gebruiker enz. Dit wordt gedaan door user interface patronen voor een bepaald type eindapparaat toe te passen. Zo komt men meestal tot een typische afleiding van een abstract model voor grote schermen, één voor compacte toestellen en eventueel een derde voor een voice interface. Het adaptatieproces heeft meestal de volgende vorm:

- Identificatie van user interface onderdelen die samen horen: bijvoorbeeld elementen in een scherm die tegelijk worden weergegeven op een bepaald tijdstip in de dialoog.
- Genereren van navigatie-elementen, die nodig zijn om tussen de verschillende groepen - geïdentificeerd in de vorige stap - te navigeren.
- Selecteren van inhoud (content) die weergegeven wordt: bijvoorbeeld bij beperkte schermgrootte wordt aanvullende informatie in een 'meer informatie' scherm gestoken.

Uiteraard bestaan hier meerdere varianten op [70].

Hierna transformeert men de geadapteerde abstracte modellen in meerdere concrete modellen met behulp van gespecialiseerde modeltransformaties zoals ATL [71] voor elk eindplatform (vb. .NET, Web,…). Deze transformaties passen in feite een patroon toe dat kennis bevat over hoe abstracte interactie objecten het best getransformeerd worden naar platformspecifieke concepten of concrete interactie objecten, waaruit uiteindelijk de code kan worden gegenereerd.

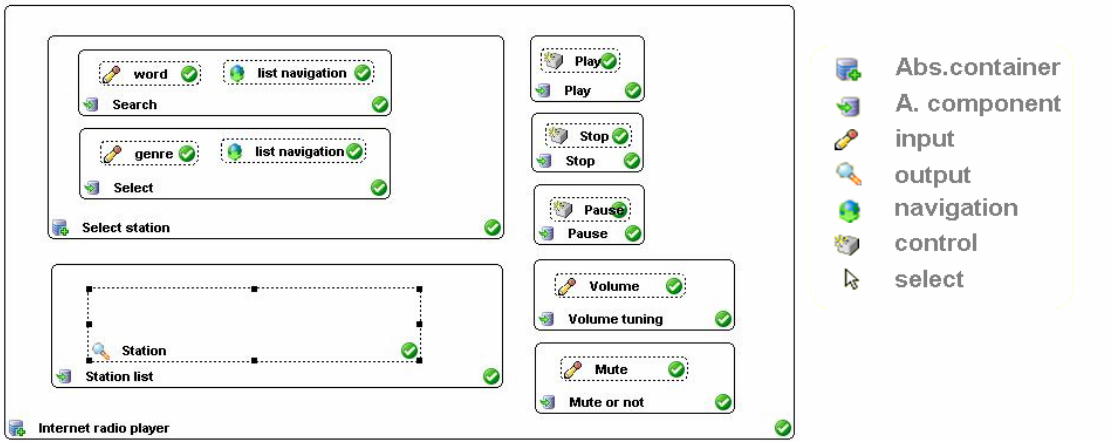
Conclusie

Het gebruik van ATL suggereert de mogelijkheid om adaptaties waarbij patronen worden toegepast automatisch uit te voeren. Het is echter zo dat dit enkel haalbaar is voor transformatie naar een specifiek software platform, en dus op laag niveau. Voor transformaties op hoger niveau blijkt een abstract model nodig zijn, wat nogmaals bevestigt dat dit type adaptatie niet kan toegepast worden op UIML gebaseerde user interfaces, tenzij men een abstract model kan afleiden uit het UIML document. Daarbij komt nog dat in geen van de vermelde aanpakmethoden men er in slaagt een volledig automatische adaptatie op abstract niveau uit te voeren. Ze kunnen wel ondersteuning bieden voor de uitvoering van de transformatie (halfautomatisch).

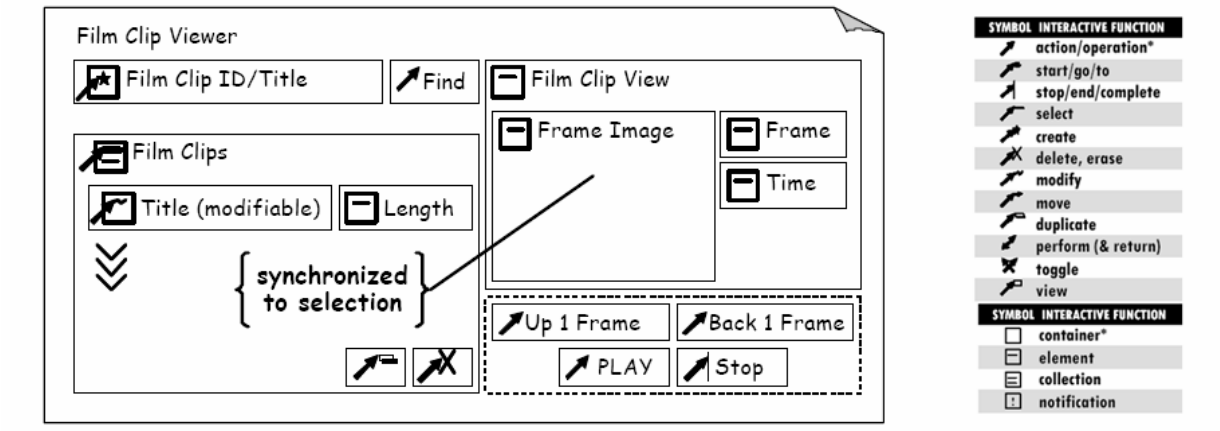
Het probleem van het continue karakter in onze definitie van context is nog steeds niet opgelost. In het domein van multiplatform ondersteuning kan men vooraf een ATL transformatie opstellen voor één bepaalde context in de zin van softwareplatform. Deze zijn relatief beperkt en kan men op voorhand vastleggen. Dit is hier niet het geval. We komen hier later op terug.

3.4.4 Canonical Abstract Prototypes

Om rekening te houden met abstracte componenten van concrete user interfaces en om doeltreffende adaptatie te ondersteunen, zou men deze kunnen aanvullen met abstracte componentbeschrijvingen. Met canonical abstract prototypes [72] kunnen we de visuele en interactiedesigns uitdrukken door de verschillende onderdelen en inhoud van user interfaces te beschrijven, onafhankelijk van presentatiedetails en gedrag van de user interface. Dit model wordt gebruikt om een vlotte overgang van abstractie naar realisatie van het user interface design te ondersteunen en is daarom bruikbaar voor de creatie van adapteerbare user interfaces. Belangrijk om te weten is dat het doel waarvoor canonical abstract prototypes ontwikkeld zijn, de ondersteuning is van designbeslissingen op een hoger abstractieniveau dan gebruikelijke prototypes, en niet zozeer de ondersteuning van meerdere eindplatformen. Canonical abstract prototypes vertrekken van het taakmodel en maken de overgang naar design gemakkelijk door twee relatief eenvoudige translatieprocessen.



Figuur 26: Voorbeeld van een canonisch prototype [83].



Figuur 27: Voorbeeld van een canonisch prototype [72].

De prototypes worden opgesteld met behulp van 'Canonical Abstract Components' die de verschillende interactieve functies die nodig zijn in de user interface, vanuit het oogpunt van de gebruiker modelleren. Elke canonische abstracte component heeft een specifieke interactieve functie die voorgesteld wordt door een symbool en een naam.

Conclusie

Ook canonical abstract prototypes kunnen een patroongebaseerd design op een abstracte manier modelleren. Hierbij moet gezegd worden dat dit uiteraard louter gaat over high-level patronen omdat low-level implementatiedetails niet worden gemodelleerd. Indien we de canonische componenten van het prototype kunnen linken met haar geïmplementeerd component, kan men uit deze metadata abstracte, zoals semantisch gerichte, adaptaties sturen.

3.4.5 Conclusie

Hoewel deze methoden doen blijken dat deze transformaties automatisch gebeuren en een louter uitvoerende taak zijn, is er een menselijke tussenkomst nodig voor de uitvoering van deze transformaties. In feite zijn ze op een zodanige manier uitgewerkt dat designers zich kunnen focussen op creatieve en fundamentele beslissingen waarvoor vaardigheden vereist zijn die tot nu toe alleen de mens bezit. De routine procedures worden automatisch afgeleverd maar moeten nog steeds handmatig uitgevoerd worden. We merken ook op dat deze methodes meestal gericht zijn op platformonafhankelijkheid in de zin van hardware- en softwareplatform (besturingssysteem en device).

Om gepersonaliseerde adaptatie te ondersteunen moeten we verder gaan dan eindplatform, en dit aanvullen met karakteristieken van de gebruiker en zijn omgeving. Theoretisch is er weinig verschil met deze uitgebreidere definitie omdat het gemodelleerd kan worden als een combinatie van vereisten, voorkeuren en beperkingen. Doch in de praktijk zal men merken dat het begrip 'context', zoals wij deze gaan gebruiken, een continu karakter heeft betreffende de waardes die ze kan aannemen, en dat hard- en softwareplatform tot op heden nog onderling als een discrete context kan beschouwd worden. In veel modellen maakt men voor elk platform een transformatiemodel. In de toekomst zal dit steeds moeilijker worden door de diversiteit aan toestellen en (aangepaste) besturingssystemen. Hiermee zal rekening gehouden moeten worden. We zullen een alternatief moeten zoeken voor het gedeelte waar er wordt uitgegaan van vooropgestelde transformatiemodellen.

Het plasticiteitdomein van high level en medium level patronen worden voornamelijk beperkt door respectievelijk apparaatkarakteristieken en gebruikerskarakteristieken. Terwijl de geschiktheid van low level patronen voornamelijk afhangen van het softwareplatform. Een transformatie creëren op een hoger abstractieniveau vereist menselijke creativiteit en een model op dat abstractieniveau. Daarom is het in ons geval met UIML user interfaces noodzakelijk deze transformaties op voorhand vast te leggen. Voor de mogelijke nood aan een ander high level patroon gaan we de specificatie van dit patroon eveneens op voorhand vastleggen. Om te bepalen of het plasticiteitdomein van high en medium level patronen in de huidige context liggen, is abstracte meta-informatie vereist.

4 Taxonomieën in adaptatietechnieken

Adaptatie kan op verschillende niveaus en manieren worden toegepast op de user interface [3]. In hoofdstuk 2 hebben we reeds besproken welke onderdelen geadapteerd kunnen worden en welke bijdrage dit kan leveren voor personalisatie van user interfaces. In dit deel trachten we een duidelijke onderverdeling te maken van de verschillende adaptatietaxonomieën. Een taxonomie classificeert verschillende adaptatiemethoden. Dit gebeurt volgens een bepaald kenmerk van de adaptatietechniek, waardoor we dit ook als een typologie kunnen beschouwen die de verschillende methoden hiërarchisch onderverdeelt. Hieruit gaan we proberen te bepalen welke vorm van adaptatie het meest geschikt is om toe te passen binnen UIML.NET. Daarom gaan we types bespreken volgens een aspect dat van nut kan zijn voor adaptatie binnen onze toepassing.

4.1 Bepalen van een taxonomie

In de inleiding hebben we het doel van adaptatie abstract kunnen definiëren als het maken van aanpassingen naargelang de context, waarbij de bruikbaarheid behouden blijft. Adaptatie wordt in vele domeinen, toepassingen en niveaus onderzocht met telkens andere aspecten die cruciaal zijn. Hierbij hebben de begrippen entiteit en context iedere keer een andere betekenis. Zo bestaan er frameworks die specifiek gericht zijn op adaptatie van multimedia (vb. MPEG-21), anderen beschouwen de adaptatie van slechts 1 aspect of onderdeel van de user interface (vb. alleen op lexicaal niveau), terwijl nog anderen zich richten tot het apparaatonafhankelijk maken van de interfaces zonder de verschillende karakteristieken van gebruikersgroepen in acht te nemen.

Ook de draagwijdte van de oplossingsruimte varieert. Een aanpak heeft bijvoorbeeld betrekking op automatische adaptatieondersteuning tijdens vroege designfasen, anderen bieden een benadering voor de hele ontwerpcyclus van de interface, die dan weer niet volautomatisch kan toegepast worden. Verschillende combinaties hiervan zijn mogelijk. Er kan een gestructureerde oplossing bestaan voor de definiëring van at-runtime adapteerbare user interfaces, maar die geen adaptatiemodel beschrijft.

Er kunnen tal van taxonomieën opgesteld worden in verschillende combinaties van dimensies [50]. Bovendien zijn deze taxonomieën niet altijd specifiek opgesteld voor het domein van (intelligente) adaptatie van user interfaces. Het is daarmee zeer moeilijk, en bijna onmogelijk, om een bepaald aspect afzonderlijk te vergelijken met die van andere onderzoeksdomeinen. Elk aspect is nauw afgesteld op verschillende factoren uit de context en andere aspecten van het domein. Zelfs benaderingen uit hetzelfde domein in hun geheel zijn moeilijk te vergelijken omdat hun doel nog net iets anders is. Daarom gaan we gaan we de dimensies waarmee we een taxonomie kunnen opstellen afzonderlijk bespreken. Na elk aspect (dimensie) leiden we af hoe en welke categorie volgens die dimensie we kunnen toepassen in ons domein van adaptatie (generische automatische personalisatie van UIML gebaseerde user interfaces).

Tot op heden is er nog geen generieke totaaloplossing gemodelleerd voor adaptatie in een soortgelijke taal als UIML. We mogen niet uit het oog verliezen dat het onderzoek in deze thesis de nadruk legt op welke manier metadata gebruikt kan worden om intelligente adaptatie te ondersteunen.

Op technisch niveau kunnen we volgende adaptatiemechanismen onderscheiden: Component adaptatie, Connectie adaptatie, Patroon adaptatie, Constraint adaptatie, Stijl adaptatie, Rationale adaptaties (voorgedefinieerde adaptaties) en een combinatie van de voorgaande [73]. Te ver ingaan op adaptatieprocessen zelf zou buiten het bestek van deze thesis vallen, we hebben de soorten adaptatie op een hoger niveau immers al besproken in de vorige hoofdstukken. Toch is het onvermijdelijk om de werking van een transformatie te schetsen, om de rol van de gebruikte metadata duidelijk te maken. Ruw gezien proberen we het type metadata dat benodigd is bij elke categorie van adaptatie af te leiden. Echter zal dit niet in alle domeinen mogelijk zijn; hier kan dus

van worden afgeweken.

Wanneer we een concept van een bestaande adaptatietechniek adopteren in onze toepassing, is het nuttig terug te vallen op deze taxonomische verdelingen. Door een techniek te plaatsen in de taxonomische onderverdelingen, leiden we af onder welke voorwaarden een methode moet plaatsvinden en of deze methode wel geschikt is voor UIML.NET. We gebruiken 4 taxonomieën die adaptatiemethoden classificeren volgens:

- tijdstip of ontwikkelingsfase van de user interface waarin de adaptatieprocedure plaatsvindt
  → We gebruiken het best adaptatiemethoden die gebruikt worden tijdens een fase waarbinnen het UIML.NET framework werkt.
- niveau of abstractielaag in de user interface waarop geopereerd wordt
  → Dit niveau dient binnen de abstractiegrens van de UIML specificatie te liggen. Deze zullen we later bepalen. We weten nu al dat uiml user interfaces vrij concreet gespecificeerd worden.
- techniek of technische benadering die toegepast wordt om de adaptatie te realiseren
  → Deze moet realistisch haalbaar zijn in het geprogrammeerde UIML.NET framework. Het karakter van UIML.NET zal meer geschikt zijn voor bepaalde technieken dan anderen.
- uitvoering of manier waarop men delen van de user interface verandert of aanpast
  → De opbouw van een UIML document suggereert een aanpassing op de delen die hierin onderscheiden worden (structure, parts, style,…). Aanpassingen die uitgaan van een andere onderverdeling zijn minder geschikt.

4.2 Tijdstip

Het tijdstip is de ontwikkelingsfase van de user interface, waar de adaptatie plaatsvindt.
Dit kan zijn tijdens (at):

- Design-time
- Compile-time
- Run-time

Afhankelijk van het tijdstip waar we de adaptatie gaan toepassen, komen er aspecten te pas die in rekening gebracht moeten worden. Als we in de design fase werken hebben we in principe het meeste bewegingsvrijheid, doch zal in veel gevallen informatie over de context nog niet beschikbaar zijn in deze fase. Ofwel zullen we het proces tussen design en implementatie volledig moeten automatiseren met de nodige complexiteit die hiermee gepaard gaat.

Hoe later we de adaptatie toepassen, hoe complexer het design zal gemaakt worden opdat deze flexibel genoeg zou zijn om een veranderlijke context te ondersteunen. Het compilatie tijdstip is meer geschikt om implementatie specifieke adaptaties uit te voeren.

Just-in-time adaptaties tenslotte kunnen het accuraatste in rekening gebracht zijn op de huidige context maar zijn beperkt tot de 'mate van adaptiviteit' of plasticiteit van de user interface. Om in elke mogelijke context ondersteuning te bieden, zal er al in de vroegste designfase een architectuur gebruikt moeten worden die het hele contextdomein beschouwt. Adaptatie doorheen sessies zal daarmee enkel praktisch haalbaar zijn bij special purpose adaptaties.

In de praktijk hebben we niet altijd de keuze in welke soort adaptatie we gaan toepassen. Sommige systemen zijn gemaakt op reeds gecompileerde user interfaces, waardoor aanpassingen in het design niet kunnen gebruikt worden. Andere user interface systemen zijn dan weer te beperkt om adaptaties tijdens compilatie of gebruik toe te laten omdat hier geen programmeermodules voor beschikbaar zijn. Hierdoor zullen adaptaties op het design moeten uitgevoerd worden. In

UIML.NET hangt de presentatie van de user interface doorheen de sessie volledig af van de applicatie ('logic') en het runtime systeem van het softwareplatform waarop de concrete user interface gerealiseerd wordt [53].

4.3 Niveau

Een andere taxonomie verdeelt de adaptatiecategorieën in volgens het niveau of de laag van de user interface die wordt geadapteerd. Dit is niet hetzelfde als de technische aspecten van een user interface die we kunnen adapteren zoals besproken in hoofdstuk 2 (gedrag, presentatie,...), maar het veranderlijke aspect waar wordt vanuit gegaan volgens het oogpunt van de gebruiker.

4.3.1 Doel

Adaptatievormen die op een hoog niveau werken, vertrekken meestal van een doelstelling die verandert. Dit kan bijvoorbeeld de snelheid (van interactie) of simpliciteit zijn die een hoge prioriteit aanneemt in een bepaalde context. Hier is de doelstelling een user interface opbouwen die de gebruiker toelaat één bepaalde basistaak snel en eenvoudig uit te voeren en waarbij verfijnde configuratie of presentatie een minder grote rol speelt. Deze technieken hebben meestal betrekking op de hele user interface en niet op specifieke onderdelen ervan. In de praktijk zijn deze adaptatiemethoden het beste te bekomen met voorgedefinieerde contexten en interface configuraties omdat het moeilijk is een systeem een user interface te laten evalueren op abstracte kenmerken zoals overzichtelijkheid en dergelijke. We kunnen dit ook niet verwachten voor UIML.NET.

4.3.2 Taak

Adaptatie op taak niveau is ook vrij abstract maar kan op specifieke onderdelen van de user interface worden uitgevoerd. Zoals de naam doet vermoeden gaan we hier uit van het taakmodel en de manier waarop deze taak ondersteund wordt, maar dan enkel voor elementaire taken. De taken die behandeld worden zijn meestal in de aard van: selectie, invoer, weergave van een item enz.. Het systeem gaat de taak herkennen ('plan recognition'), en de user interface schikken op een dergelijke manier zodat de taak het beste kan uitgevoerd worden. Zo kunnen er bepaalde widgets verschijnen en verdwijnen die respectievelijk wel en niet van belang zijn bij de uitvoering van een taak. Bij de selectie van een reeks items in een container kan men deze vergroot weergeven om zo de selectie te vergemakkelijken. De user interface zal dergelijke vorm aannemen die specifiek is afgesteld voor de ondersteuning van deze bepaalde taak.

Figuur 29: Adaptatie aan een actieve taak: *door programmagids navigeren (links), programma bekijken (rechts).*

Dit type van adaptatie is vooral nuttig tijdens het gebruik van een interface (runtime). Het systeem

dient immers te weten met welke taak een gebruiker bezig is op een bepaald tijdstip en heeft daarvoor applicatiespecifieke gegevens nodig. Dit kan moeilijk uitgevoerd worden vanuit UIML.NET maar eerder in de applicatie waarvoor een user interface gemaakt is.

4.3.3 Semantiek

Een ander gebruikersaspect waar men zich op kan richten is de semantiek. In tegenstelling tot wat men zou denken kan dit van toepassing zijn op zowel de presentatie, het gedrag en de interactie met de user interface. Deze soort van adaptatie handelt enkel over metaforen, presentatie en het mentaal model. Ze is gericht naar de begrijpbaarheid van de interface. Dit type van adaptatie kan dus in zekere mate toegepast worden in onze toepassing door bijvoorbeeld andere widgets en display elementen te gebruiken.

4.3.4 Syntactisch

Op syntactisch niveau gaan we al meer concrete aspecten aanpassen zoals de uitdrukkingswijze en samenstelling van de user interface. In ruwe termen gesproken is dit de 'notatie' van de user interface die men aanpast. De interface heeft dezelfde low-level functionaliteiten, maar worden op een andere manier of in een andere volgorde benaderd. De plaats van de menubalk of de opeenvolging van acties om een taak uit te voeren zijn eigenschappen die betrekking hebben op syntax van de user interface.

Ook dit is realiseerbaar in de presentatie van UIML user interfaces door de componenten in de ‘structure’ sectie te herschikken of gebruik te maken van een andere template.

4.3.5 Lexicaal

Deze categorie van adaptatie past de lexicale inhoud van de user interface aan. Het betreft hier de symbolische notaties, figuren en tekstuele labels. De structuur en elan blijft behouden, enkel expliciete gegevens worden aangepast. Deze vorm van adaptatie wordt toegepast om meertaligheid en regiogebonden eenheden te ondersteunen. Het kan gebruikt worden om user interfaces toegankelijk te maken voor personen met een beperkt zicht of analfabeten door bijvoorbeeld symbolische tekens te gebruiken in plaats van tekst.

Ook dit kan redelijk eenvoudig worden toegepast op presentatie-elementen van UIML beschrijvingen, door in de ‘content’ sectie aanpassingen te maken of andere waarden voor ‘references’ te gebruiken.

4.3.6 Conclusie

We merken dat adaptatietechnieken die op aspecten op een hoger niveau werken, ook invloed hebben op abstractere aspecten van de user interface en omgekeerd. UIML beschrijvingen laten, zoals we in hoofdstuk 3 reeds gesteld hebben, gemakkelijker aanpassingen van aspecten op laag niveau toe. Adaptaties op een lager niveau lijken dus meer geschikt te zijn voor onze toepassing.

4.4 Techniek

In dit deel bespreken we de technische benaderingen die gebruikt worden om adaptatie te ondersteunen en de manier waarop de eventuele metadata hiervoor gebruikt wordt. Deze technieken kunnen in principe voor elk type van adaptatie in het vorige deel worden gebruikt. Deze taxonomie deelt eigenlijk de verschillende adaptatietechnieken in volgens het designpatroon dat gebruikt wordt voor de implementatie ervan. In UIML user interfaces kunnen we adaptatie aan de context in principe op 2 manieren realiseren:

- Aanpassingen op eigenschappen van de user interface toepassen door te sleutelen aan de stijlattributen in de ‘style’ sectie [56].
- Aanpassingen in de opbouw van de user interface aanpassen door gebruik te maken van layout-templates, en de mappings van generische ‘parts’ op concrete widgets te wijzigen [57].

We beperken ons tot de technische benaderingen die hiervoor bruikbaar zijn.

4.4.1 External adaptivity

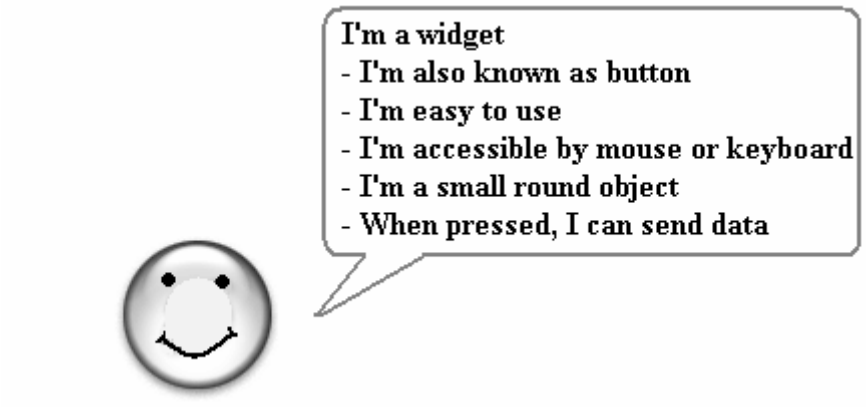
Bij external adaptivity of een open adaptieve aanpak [15] gaat men de (widgets van) user interfaces zo maken dat ze aangepast kunnen worden, maar dat de verantwoordelijkheid niet bij de widgets zelf ligt. Dit laatste wordt uitgevoerd door een al dan niet externe component.

4.4.1.1 Reflectie

Om een specifiek widget te selecteren voor de ondersteuning van een bepaalde taak in een bepaalde context, heeft het adaptatiesysteem de eigenschappen van de tools (widgets) uit de toolkits nodig die het ter beschikking heeft. Opdat het dit zou kunnen, gaat men de toolkit reflectief maken door functionele en niet-functionele informatie over elk widget beschikbaar te stellen. De metadata van een widget gaat louter over de eigenschappen van het widget zelf en beschrijft de functie van een widget zoals bv. het selecteren van 1 waarde uit een set van opties. Deze functionele eigenschappen gaan dus niet over de adaptatiemogelijkheden van de component. Niet-functionele eigenschappen zijn o.a. kosten, noodzakelijke acties voor gebruik, vereisten aan het eindplatform en variabele opties.

4.4.1.2 Introspectie

Introspectie wil zeggen: het kunnen waarnemen van eigenschappen over zichzelf. Bij een introspectieve aanpak gaat men user interfaces en/of componenten (widgets meestal) introspectief maken door ze zelf hun functionele en niet-functionele eigenschappen te laten publiceren. We laten hierdoor de widgets ‘spreken’ (figuur 30). Dit kunnen naast widgets uit de toolkit ook asbtracte widgets zijn of een subinterface die deel uitmaakt van de hele user interface. De component die verantwoordelijk is voor de adaptatie gaat op basis hiervan de user interface en de elementen waaruit het is opgebouwd aanpassen. Opdat een 'combo box' introspectief zou zijn moet deze zijn taken die het ondersteunt (single selection) en kosten (footprint, interactiepad) exporteren. De informatie kan (maar hoeft niet te) gaan over het widget binnen de context van een specifieke user interface waar het in gebruikt wordt. De adaptatielaag kan met deze informatie een evaluatie opstellen over het user interface element en een adaptatieprocedure samenstellen.



Figuur 30: Introspectief widget.

4.4.1.3 Polymorfisme

Polymorfische user interfaces publiceren meerdere versies van hun componenten waaruit ze zijn opgebouwd. Een polymorfisch widget publiceert meerdere implementaties van zichzelf. Het bevat dus geen mechanismen maar al uitgewerkte geadapteerde versies (alternatieven) van zichzelf. Het is aan de component die instaat voor de adaptatie, om de polymorfische widgets juist in te stellen en dus een geschikte implementatie te kiezen. Omdat het dit redenerend zou kunnen uitvoeren dienen de widgets best ook introspectief te zijn.

4.4.2 Self-contained adaptivity

Bij self-contained adaptivity of een close adaptive (gesloten adaptieve) aanpak [15] werken we met een systeem dat alle componenten bevat voor zelfstandige adaptatie. Er komt geen tussenlaag te pas die instaat voor adaptatie. Close adaptive user interfaces bevatten zelf mechanismen voor de uitvoering van de adaptatie.

Op widget-niveau vallen hieronder de self-adaptive widgets. Deze zijn in staat zichzelf te adapteren aan de context of use. Ze zijn close adaptive en bepalen dus zelf wanneer ze adapteren (en wordt niet gedaan door een externe component). Dit kan bereikt worden met polymorfische widgets die zichzelf transformeren door te switchen van de ene 'vorm' naar de andere. Bijvoorbeeld een widget dat een tijdstip weergeeft, kan een versie van zichzelf hebben met analoge klok metafoor en een versie met digitale weergave.

4.4.3 Adaptatie-uitvoering

Naast de adaptatietechnieken kunnen we ook een taxonomie van de uitvoeringstechniek van de adaptatie opstellen. M.a.w. de effectieve uitvoering van de opgestelde adaptatieformule.

4.4.3.1 Switching

Een vorm van aanpassing van de user interface kan men bekomen door een onderdeel ervan te vervangen door een ander met dezelfde taakondersteuning. Dit lijkt op het eerste zicht eenvoudig omdat het originele onderdeel niet aangepast dient te worden. Echter kan dit onderdeel uitmaken van een samengesteld onderdeel of afhankelijk zijn van andere onderdelen in de user interface. De verbinding met andere componenten moet afgesteld zijn op het nieuwe onderdeel. Zo kan de parameter anders zijn voor de weergave van een bepaalde waarde.

4.4.3.2 Reconfiguration

De verschillende onderdelen van de user interface behouden maar aanpassen door hun instellingen te wijzigen heet reconfiguration. Herconfiguratie heeft niet enkel betrekking op één widget of een verzameling samenhangende widgets maar kan de aanpassing van een reeks andere onderdelen van de user interface aangaan. Zo kan men niet enkel de scrollrichting van een deel van de user interface wijzigen van verticaal naar horizontaal, zonder hiervoor de items en aangrenzende widgets te gaan herpositioneren als gevolg van de breedte die zal moeten aangepast worden voor bruikbare verticale scrolling. Valkuilen waar geacht wordt rekening mee te houden zijn onmogelijke configuraties en incompatibele combinaties.

4.4.3.3 Enabling/disabling

Een andere manier om adaptatie uit te voeren bestaat uit het activeren en deactiveren van een onderdeel van de user interface. Dit kan nuttig zijn voor extra support voor beginnende gebruikers door bv. help secties te weergeven en tooltips te activeren. De weergave van snellinks is dan weer handig voor experts. Ook uitgebreidere transformaties kunnen bekomen worden door alle

onderdelen en configuraties voor eender welke context in de user interface plaatsen, en afhankelijk van de huidige context bepaalde secties en onderdelen zichtbaar maken (activeren), en anderen te verbergen (deactiveren).

Ook deze methode mag niet onderschat worden door de simplistische uitstraling ervan. Doordat alle onderdelen reeds geïmplementeerd en geconfigureerd zijn, lijkt activering en deactivering op basis van contextgegevens een eenvoudige primitieve procedure. Toch kan het verbergen of weergeven van een onderdeel bijkomende aanpassingen vereisen. Bij het weglaten van een knop dient zijn functie op een andere manier te worden aangeroepen of door een ander widget te worden overgenomen bv. door een 'event' die na selectie van een item uit een selectielijst 'getriggered' wordt (in het geval van een submit-functie).

4.5 Conclusie

Het begrip adaptatie kan volgens vele dimensies worden onderverdeeld. De keuze van het type adaptatie staat niet altijd vrij en ieder heeft zijn voor- en nadelen. De ideale aanpak hangt af van hetgeen men wil bereiken en de omgevingsfactoren zoals platform, taal en het type interface. Het concept van Digital Item Adaptation in MPEG-21 bijvoorbeeld, vindt plaats at compile- en run-time; op lexicaal niveau; en kan worden beschouwd als een open adaptieve aanpak waarbij introspectie en polymorfisme wordt gebruikt. Ondanks DIA van toepassing is op multimedia items kunnen we stellen dat DIA zowel switching (alternatief multimedia element), reconfiguration (wijzigen van bitrate, codec,…) als enabling/disabling (weglaten/invoegen van geluid of extra streams) als adaptatie-uitvoering toepast. Het type adaptatie dat men gaat toepassen dient zorgvuldig te worden afgewogen met andere methoden door hun voor- en nadelen te vergelijken.

5 Aanpak opstelling voor UIML.NET

Na het domein van adaptatie verkend, bestudeerd en afgebakend te hebben gaan we uit de resultaten ervan keuzen moeten maken voor het toepassen van theoretische modellen in onze UIML.NET toepassing. Hiervoor gaan we UIML en UIML.NET met betrekking tot dit onderzoek iets grondiger bestuderen om de concrete eisen en de architectuur van het adaptatiemechanisme op te stellen. Vervolgens bespreken we in het kort hoe we de resultaten uit de vorige hoofdstukken gaan toepassen om aan deze eisen te voldoen.

5.1 Scenario's

Om te bepalen welke gegevens we gebruiken en hoe we deze abstraheren stellen we mogelijke scenario's op. Als toepassing voor onze scenario's gebruiken we een publieke applicatie voor het reserveren van cinema- en/of theatertickets. Het is de bedoeling dat de volgende info moet verkregen worden van de gebruiker:

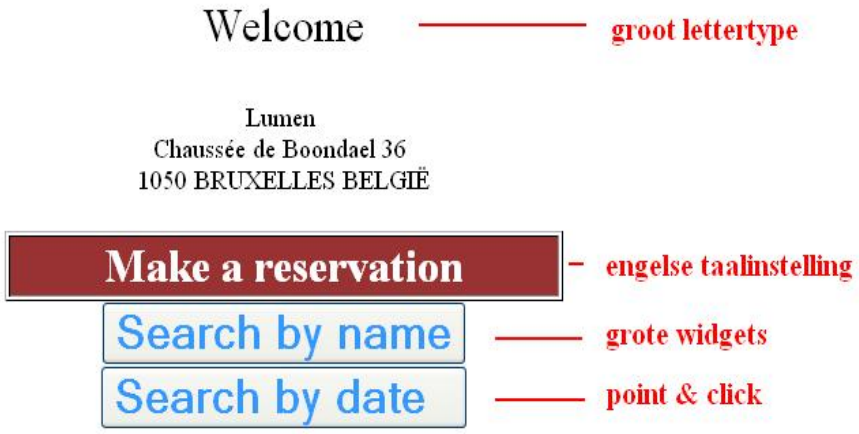
- *voorstellings id*
  (kan afgeleid worden uit: *titel* , *tijdstip* (kan afgeleid worden uit *dag* en *uur*))
- *plaats-categorie*
  (premium-, vip-, zit- , staanplaats, ...)
- *aantal*

Scenario 1
Schulz is een man van 42, heeft Engels als moedertaal, is slechtziend, en heeft weinig ervaring met elektronische toepassingen. Hij benadert de toepassing met een PC met alle standaarddevices zoals toetsenbord, muis, luidsprekers, en kleurenscherm. Hij geeft voorkeur aan duidelijke en simpele interactie boven versnelde, meer efficiënte interactie.

Deze informatie zit samen met de specificaties opgeslagen in zijn profiel. Dit profiel wordt doorgestuurd naar de applicatie.

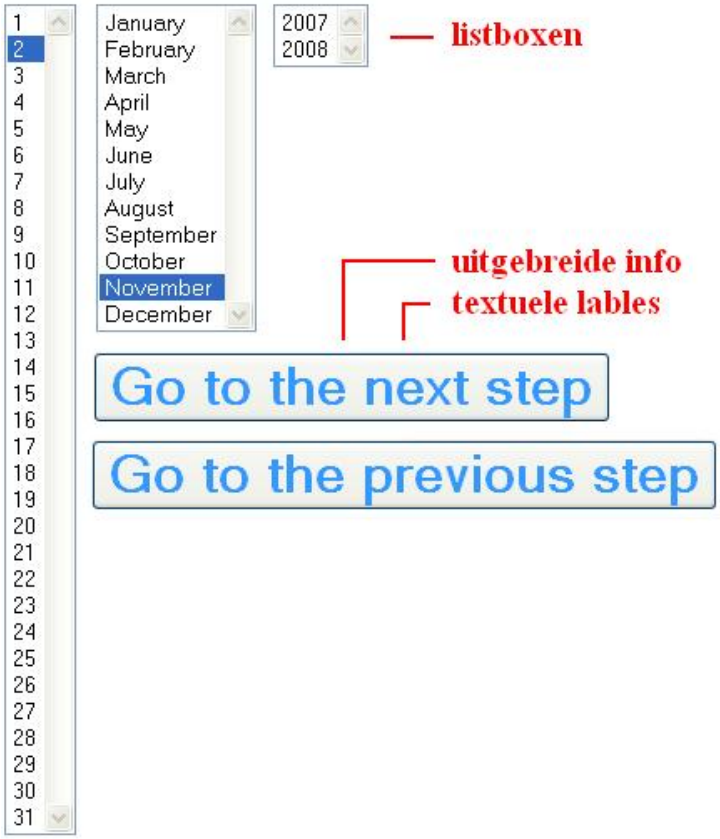
De interface wordt gegenereerd met berekende parameterinstellingen en heeft de volgende eigenschappen:

- Wizard interactiemetafoor
- Textuele labels in plaats van symbolen
- Grote buttons, widgets en lettertype.
- Point&Click interactie
- Maximale zichtbaarheid van selecteerbare waarden



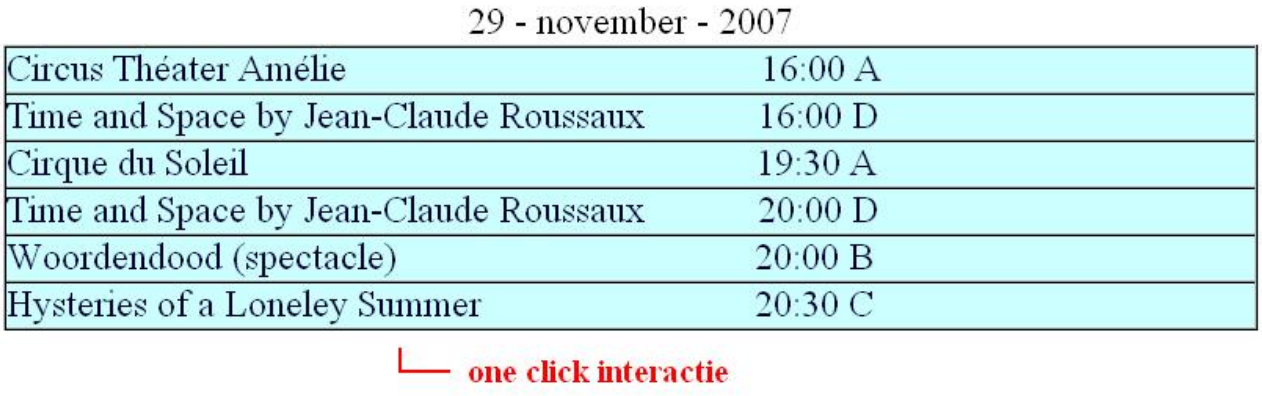
Figuur 31a: Weergave van resultaat van scenario 1.

Er wordt geklikt op 'Search by date'.



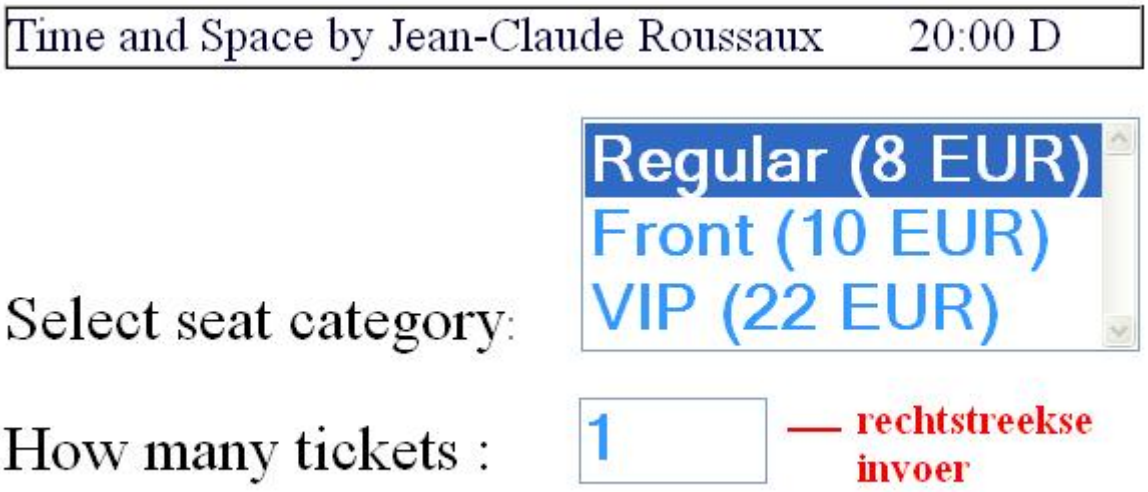
Figuur 31b: Weergave van resultaat van scenario 1.

'Go to the next step' met auditief feedbackgeluid.



Figuur 31c: Weergave van resultaat van scenario 1.

Voorstelling wordt aangeklikt met de muis.



Figuur 31d: Weergave van resultaat van scenario 1.

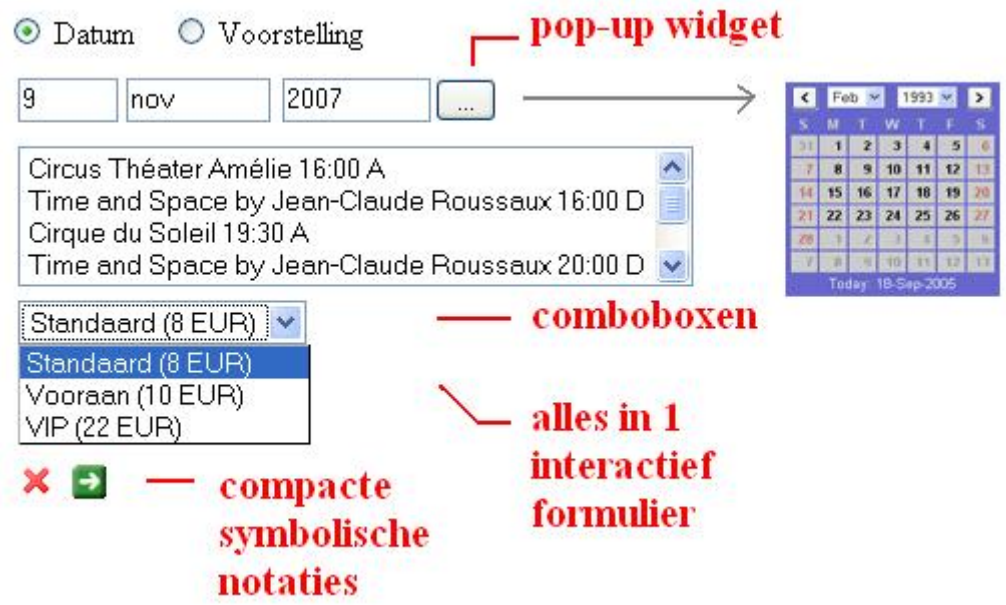
Scenario 2

Mario is een ervaren gebruiker, is Nederlandstalig en benadert de toepassing met een PDA met kleurenscherm, pen, en numeriek klavier als invoerfaciliteiten. Hij verkiest een snelle en efficiënte interactiestijl en houdt van multimedia.

Deze informatie zit samen met de specificaties opgeslagen in zijn profiel. Dit profiel wordt doorgestuurd naar de applicatie.

De interface wordt gegenereerd met berekende parameterinstellingen en heeft de volgende eigenschappen:

- Form interactiemetafoor
- Compacte symbolen
- Compacte widgets
- Point&Click interactie
- Dynamische zichtbaarheid van selecteerbare waarden



Figuur 32: Weergave van resultaat van scenario 2.

Scenario 3
Sofie is matig ervaren, gebruikt een WAP toestel met numeriek klavier, een scroller en een zwart-wit scherm. Ze geeft de voorkeur aan een overzichtelijke interactie.

Voor haar wordt een WAP-interface met wizard display gegenereerd met tekstuele notaties of basissymbolen.

Invoer gebeurt door scrolling en buttons. Acties worden toegewezen aan toetsen ipv widgets (buttons met de tekst 'naar volgende stap' worden weggelaten). Er wordt consequent gebruik gemaakt van de meest compacte widgets.

5.2 UIML.NET

UIML.NET is een open-source renderer voor UIML, geschreven in C#. Het werd ontwikkeld op het Expertisecentrum Digitale Media door o.a. Kris Luyten [74]. Zoals eerder vermeld is UIML een taal om user interfaces te beschrijven op een platformonafhankelijke manier, ook wel HLUIDL genoemd. Het gebruik van UIML voor het ontwikkelen van user interfaces biedt vele voordelen [75]:

- slechts kennis van 1 syntax nodig
- gemakkelijk uitbreidbaar
- overdraagbaar naar andere platformen
- apparaatonafhankelijk
- eenvoudig consistent te houden tussen verschillende eindplatformen

Tot nu toe bestaan er 4 vocabularies die gebruikt kunnen worden voor UIML.NET :

- gtk-sharp-1.0.uiml (GTK#)
- swf-1.1.uiml (Windows Forms)
- wx-net-1.0.uiml (WX.NET)
- cswf-1.0.uiml (Compact Windows Forms)

Het punt is dat we geen adaptatie gaan uitvoeren op UIML als meta-taal, maar op user interfaces die beschreven zijn met UIML. Indien we de uitvoering van één concreet UIML voorbeeld bekijken, weten we dat de gerealiseerde user interface gecompileerd wordt uit een combinatie van informatie uit het uiml document zelf, en informatie uit een uiml vocabulary die de generische classen in het uiml document op concrete widgets mapt. Men beschrijft in feite een volledig uitgewerkte concrete user interface. UIML.NET leest de uiml beschrijving in en genereert de user interface volgens de mappingsregels (peers) van de gebruikte vocabulary. Het framework maakt daarbij zelf geen beslissingen.

5.3 Adaptatie type bepalen

Om te bepalen tot op welk niveau we adaptatie kunnen toepassen in UIML, zijn we nagegaan waar de abstractie stopt in UIML (zie sectie 2.2). Abstractie is namelijk een sleutelvereiste voor intelligente adaptatie (zie sectie 2.2). We weten al dat UIML abstract is op platformniveau. Hoe abstract is UIML binnen 1 platform? Het bleek dat dit op widget-abstractieniveau is. Doordat de user interface wordt geconcretiseerd door mappings in de gebruikte vocabulary, zou een adaptatie uitvoeren op de vocabulary een uitkomst kunnen bieden. Maar dit is geen bruikbare oplossing omdat dit 1-op-1 mappings zijn en andere delen van het uiml document opgesteld zijn volgens deze mappings [55]. Doordat we er van uitgaan dat we vocabularies blijven gebruiken van deze 1-op-1 mappings op specifieke widgets (Windows Forms, GTK+,…), suggereert de aard van UIML duidelijk een open-adaptive gerichte aanpak (zie sectie 4.4.1) voor de ondersteuning van adaptaties.

Dit wil concreet zeggen dat de metadata laag die aangebracht wordt in de user interfaces niet de procedures en regels voor adaptatie zullen bevatten, maar dat deze uitgevoerd worden door een (externe) component. Voor een close-adaptive (zie sectie 4.4.2) gerichte aanpak is UIML niet abstract genoeg.

Als we de UIML specificatie nader bekijkten, merkten we dat de structuur van de user interface abstract wordt beschreven. De structuur van een user interface wordt gespecificeerd zonder het uiterlijk, de inhoud en functionaliteit te bepalen. Deze 3 aspecten worden pas bepaald in respectievelijk de style, content en behavior sectie (zie 2.2.2). De abstractie in de structure sectie kan nog iets verder worden verhoogd door het gebruik van templates en een variabel 'class' attribuut of rendering attribuut (nog niet ondersteund in de huidige versie van UIML.NET). Ook de 'restructure' sectie kan interessant zijn voor het gebruik van adaptatie (nog niet geïmplementeerd in de huidige versie van UIML.NET). Deze scheiding maakt UIML user interfaces aantrekkelijk voor aanpassingen op deze delen. Als we dit in beschouwing nemen in combinatie met eerdere conclusies uit sectie 2.2.4, besluiten we dat UIML user interfaces geschikt zijn voor adaptaties op hun interactie, presentatie en gedrag, op lexicaal, syntactisch en op zekere hoogte semantisch niveau. Deze aspecten kunnen in UIML afzonderlijk worden beschreven en zijn relatief weinig afhankelijk van elkaar. In theorie zijn adaptaties op andere en meer abstracte aspecten zoals dialoog ook mogelijk maar dan spreekt men niet meer van intelligente maar voorberekende adaptatie (zie sectie 4.1). Dit komt doordat UIML hier niet naar gericht is.

5.4 Adaptatie tijdsfase

Om efficiënte adaptatie te bekomen en omdat er best geen veranderingen aangebracht moesten worden in de kernmodules van de renderer, is het tijdstip van de adaptatie in het generatieproces van de user interface een belangrijke beslissing. Door geen veranderingen aan te brengen in de kernmodules van UIML.NET, kan het adaptatiemechanisme gemakkelijk worden overgenomen naar andere systemen en frameworks.

Om een gepaste fase in het compilatieproces te kiezen, gaan we na hoe dit proces verloopt in UIML.NET. Dit gebeurt als volgt:

- Het UIML document wordt ingelezen waarbij er een object georiënteerd model of UI-tree van wordt opgesteld. Tijdens deze fase worden attributen toegewezen aan parts, en de templates aangebracht.
- Aan de hand van de vocabulary wordt er bepaald voor welk platform het UIML document gemaakt is, en de juiste backend factory ingeladen. De user interface wordt stuk voor stuk via reflectie door de factory gecompileerd aan de hand van de mappings in de vocabulary.

In sectie 3.3 en 3.4 hebben we geconcludeerd dat adaptaties op hoger abstractieniveau vroeger in het ontwikkelingsproces dienen plaatst te vinden. Als we deze willen ondersteunen, dienen ze worden toegepast tijdens of na de eerste stap (het inlezen van het UIML document) en voor de tweede stap (het mappingsproces op de widgets).

Adaptaties op laag implementatie-niveau dienen toegepast te worden na de eerste stap of tijdens de tweede stap.

De adaptatie gebeurt daarom in 2 fasen:

De eerste fase vindt plaats tijdens de opbouw van het interne object georiënteerde model van het UIML-document (of UI-tree). In deze fase worden:

- de implementaties (AIO's op middelhoog niveau) en opbouw voor de subinterfaces gekozen, en
- de lexicale inhoud bepaald.

De tweede fase gebeurt na de opbouw van het model van het UIML-document, en voor het user

interface generatieproces van dit model. Hierin worden:

- AIO's op laag niveau en widgets (CIO's) bepaald, die zullen gebruikt worden voor een elementair component (part),
- de transformatie uitgevoerd om deze te bekomen,
- aanpassingen gedaan op de eigenschappen van de (op dit moment concrete) user interface onderdelen als gevolg van de adaptaties die in de eerste fase bepaald werden, en
- de waarden van deze en reeds aanwezige eigenschappen van de widgets aangepast.

Een bijkomend voordeel van deze aanpak waarbij geen veranderingen aan de vocabulary worden aangebracht, is dat de adaptatie server-sided kan gebeuren.

5.5 Metadata om adaptatie te ondersteunen

Na de procedure uit het vorige deel zou men een concrete en aan de context geconfigureerde user interface beschrijving moeten hebben. Tot nu toe weten we op welke plaatsen we kunnen sleutelen om de user interface aan te passen. Er is echter te weinig informatie opdat de adaptatiecomponent autonoom kan bepalen welke adaptaties mogelijk zijn en welke daarvan het meeste geschikt is. We hebben geconcludeerd dat metamodellen die afgeleid zijn uit metadata, ervoor moeten zorgen dat de juiste adaptatie wordt toegepast (zie sectie 3.2 , 3.3 en 3.4). Het bleek dat we geen specifieke contextparameters kunnen opstellen voor de UIML specificatie op zich, aangezien deze voor elk UIML document anders zijn en deze hiernaast afhangen van de gebruikte vocabulary (zie sectie 3.3). We gaan dus een generische metalaag modelleren die gebruikt kan worden tijdens het design van de user interface.

We hebben het begrip context gedefinieerd als alle informatie die de staat van een entiteit karakteriseert (zie 2.1.1). We dienen hierbij alle entiteiten te beschouwen die van belang zijn bij adaptatie beslissingen zoals in deel 7 (DIA) van het MPEG-21 principe (zie 3.2.3.1). Deze kunnen bijgevolg afgeleid worden uit de entiteiten die een designer gebruikt om het design te adapteren. Tot nu toe genereert de UIML.NET renderer blindelings wat er in het UIML-document beschreven staat d.m.v. vaste mappings in de vocabulary. Om aan intelligente adaptatie te kunnen doen, moet de deze:

- weet en kennis hebben over de widgets (classes in de vocabulary) en user interface elementen
- weet en kennis hebben over de gebruiksomgeving (= gebruiker, platform en situatie)
- het vermogen hebben om te redeneren op basis van deze kennis

Deze kennis zal het moeten halen uit de deelnemende entiteiten die relevant zijn voor een afgestemde user interface. Ze zal daarom voorzien worden van metadata. In ons geval leiden we de volgende entiteiten af:

- gebruiker
- beschikbare classes (widgets) in de vocabulary
- platform (device + besturingssysteem)
- gebruiksituatie
- (eigenschappen van) de user interface zelf

Vanaf nu gebruiken we de term context dus als een combinatie van (de staat van) deze 5 entiteiten.
De entiteiten worden gedeclareerd volgens het Digital Item Declaration principe van MPEG-21 (zie deel 3.2.3.1). De user interface, natuurlijke omgeving en vocabulary worden als containers beschouwd van 'digital items' (in onze toepassing gebruiken we hiervoor respectievelijk de termen 'delen', 'aspecten' en 'klassen'). Eerder hebben we geconcludeerd dat we de entiteiten die per

gebruik van 1 user interface variabel zijn, op dezelfde manier kunnen behandelen (zie 2.2.1). ‘Gebruiker’, ‘platform’, en ‘gebruiksituatie’ kunnen we dus samennemen in één ‘contextonderdeel’. Meta-informatie over de vocabulary scheiden we ook van deze van de user interface omdat deze kan hergebruikt worden.

5.6 Welke metadata gebruiken

Welke meta-informatie we nodig hebben over een entiteit, verschilt per toepassing en zal weer net zoals in design engineering van user interfaces, uit design principes kunnen gehaald worden. De uitdaging hier is deze gegevens net zoals de UIML standaard dit doet, deze op een uniforme, maar ondubbelzinnige en voor de machine begrijpbare manier uit te drukken. Hierbij horen ook zeer logische gegevens die het design bepalen en men als mens vanzelfsprekend vindt.

Ook al zal de designer bepalen voor welke contexten de user interface bruikbaar moet zijn en welke meta-informatie hij/zij hiervoor gaat gebruiken, gaan we abstract het doel beschrijven waarvoor deze informatie gebruikt kan worden. Afhankelijk van het aspect dat men gaat aanpassen is er andere metadata vereist. Hetzelfde geldt voor de fase waarin men zich bevindt in het adaptatieproces (zie 3.3.4). Mogelijke doelen van het redeneren uit meta-informatie zijn:

- identificeren van een Abstract Interactie Object (AIO) uit Concrete Interactie Objecten (CIO’s)
- selectie van een geschikt patroon voor de nieuwe context
- bepalen of een transformatie naar een AIO of implementatie van een widget mogelijk is
- bepalen welke onderdelen en eigenschappen aangepast kunnen worden en wat de gevolgen hiervan zijn
- verbanden tussen aspecten van de context identificeren
- samenstellen van een transformatie

Om te bepalen welke informatie we gaan opslaan en op welke manier, onderzoeken we eerst welke *soort* gegevens we gaan verzamelen en *waarover* we gegevens gaan verzamelen over een bepaald element. Dit kunnen de volgende zijn:

Gebruiker:

- wat wil de gebruiker (niet).
- wat kan de gebruiker (niet).
- over welke technische mogelijkheden beschikt de gebruiker (niet).

Widgets in de vocabulary:

- welke (soort) interactie is er vereist voor het widget.
- hoe ziet dit widget er uit.
- welke functies heeft het widget, waarvoor kan het gebruikt worden (uitvoer, manipulatie, beide, soort data).
- welke media gebruikt dit widget, welke zijn de technische vereisten.

User interface:

- Waarvoor wordt de user interface of onderdeel ervan gebruikt, wat is het doel.
- Welke semantische eigenschappen heeft de user interface of onderdelen ervan.
- Welke patroon is er toegepast voor een onderdeel, van welk AIO is dit afgeleid.

Indien bepaalde informatie niet kan worden afgeleid, zal deze expliciet aangeboden moeten worden (zoals het gebruikte AIO, eisen van de gebruiker, beperkingen of constraints, mappings tussen aspecten). Net zoals UIML een declaratieve beschrijvingstaal is met generische tags, is het contextmodel op zich abstract en kan de context door de auteur van het UIML document vrij gespecificeerd worden. Het hergebruiken van een contextbeschrijving met de mappings op andere aspecten van de context moet natuurlijk mogelijk blijven zoals dit bij vocabularies het geval is.

5.7 Welke componenten aanpassen

We weten al welke aspecten van de user interface we kunnen laten aanpassen in de UIML-beschrijving en op welk niveau. We overlopen kort welke concrete technische componenten we hiervoor moeten aanpakken.

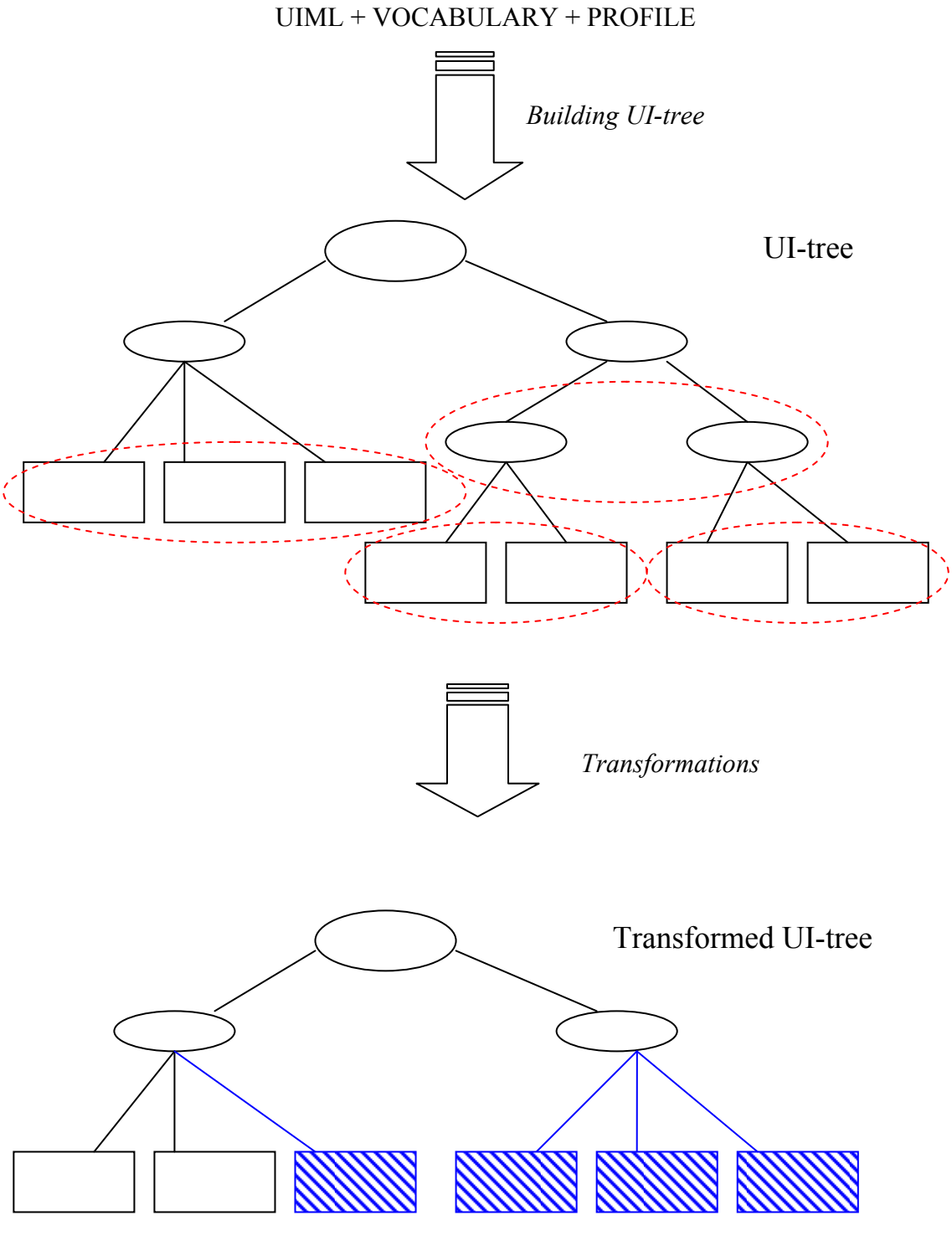
Doordat er geen verschil is in de manier waarop UIML user interfaces worden beschreven zal de adaptatiecomponent aanpassingen doen aan de specificatie van een bepaalde user interface. Plaatsen waar adaptaties / aanpassingen kunnen worden gedaan zijn:

- Classes (widget selectie) :
  Klasse van een part wordt aangepast.
- Inhoud (structures) :
  Verscheidene delen van de user interface worden vervangen door andere (alternatieve subinterfaces).
- Style attributen :
  Eigenschappen van een widget worden aangepast (kleur, zichtbaarheid, positie).
- Inhoud (constants) :
  Literaire inhoud wordt aangepast: tekst, notaties, symbolen,... .
- Behavior :
  Functionele eigenschappen worden aangepast.

Nu al kan men voorspellen dat deze adaptaties niet onafhankelijk van elkaar kunnen gebeuren. Een aanpassing aan de gebruikte klasse, zal een aanpassing aan de style en behavior eigenschappen vereisen.

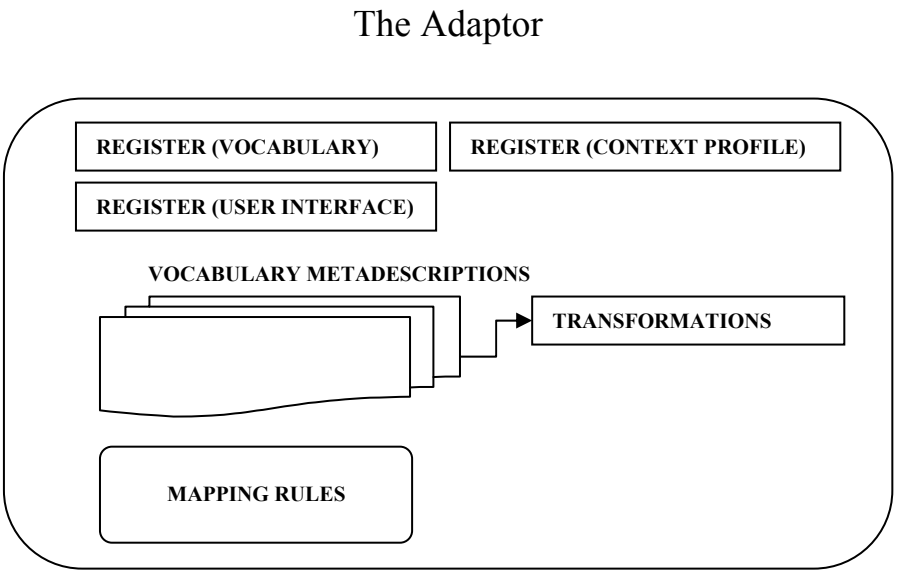
6 The UIML.NET Adaptor

In dit hoofdstuk wordt de implementatie van de resultaten van dit onderzoek besproken op het UIML.NET framework. In het eerste deel wordt de technische opbouw besproken om de doelstellingen uit het vorige hoofdstuk te realiseren, gevolgd door een bespreking van een voorbeeld, en een evaluatie van de resultaten. Als laatste vormen we aan de hand van deze resultaten een algemene conclusie om af te sluiten.



Figuur 33: Algemene werking van de adaptor.

De implementatie kan in 3 grote onderdelen worden verdeeld: het eerste deel zorgt voor de adapteerbaarheid, en bevat de 'adaptatietools' die de user interface toegankelijk maakt voor adaptatie; het tweede deel modelleert de context met metaobjecten zodat deze door het systeem toegankelijk wordt; het derde deel legt de verbanden tussen de verschillende aspecten van de context en integreert deze in de beslissingsprocedures. Om de plaats van elke onderdeel in het geheel niet uit het oog te verliezen, wordt na de bespreking van elk deel onmiddellijk de technische implementatie in UIML.NET besproken. Indien nodig wordt er even teruggevallen op de situering t.o.v. de andere delen.



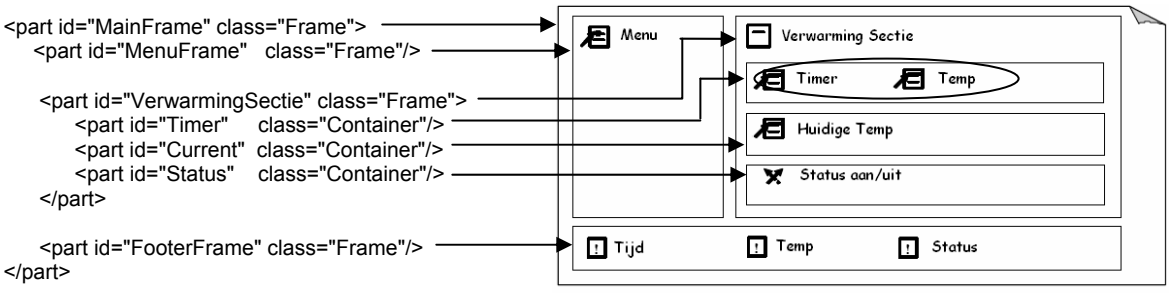
Figuur 34: The Adaptor

6.1 Adaptatie van de user interface

6.1.1 Indeling van de user interface

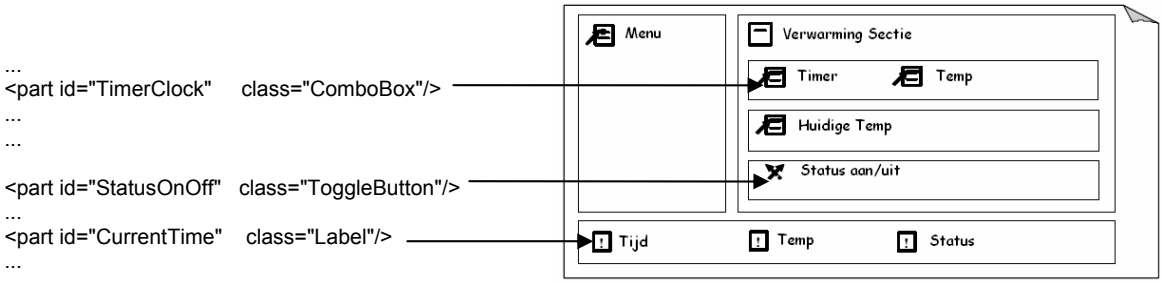
De user interface is een compositie van verschillende onderdelen en kan bestaan uit meerdere subinterfaces die op zich weer verschillende secties bevatten. In het vorige deel (zie 5.7) hebben we afgeleid welke delen en aspecten kunnen geadapteerd worden in UIML user interfaces. Dit zijn aspecten gezien vanuit het oogpunt van de gebruiker. De vraag hierbij is: "Hoe vertaalt zich dit naar een onderverdeling vanuit technisch oogpunt?". We onderscheiden hiervoor de volgende delen van een samengestelde UIML user interface:

- Samengestelde delen : Deel van de user interface dat samengesteld kan zijn uit andere elementaire of samengestelde delen zoals het Digital Item Adaptation principe in de MPEG-21 standaard. Een samengesteld deel kan gezien worden op alle mogelijke niveaus; van een samengesteld widget (compositie van widgets) tot de hele user interface.



Figuur 35: Samengestelde delen in een UIML gebaseerde user interface.

- Elementaire delen : Delen die niet verder kunnen opgesplitst worden in kleinere delen. Het zijn de elementaire widgets of bouwstenen van de user interface. Dit kunnen concrete widgets, afbeeldingen en tekst zijn.

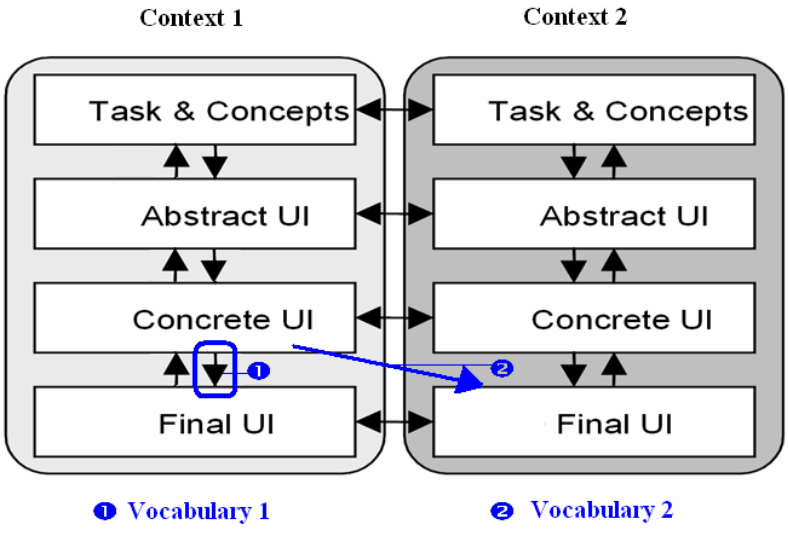


Figuur 36: Elementaire delen in een UIML gebaseerde user interface.

6.1.2 Adaptatieniveaus

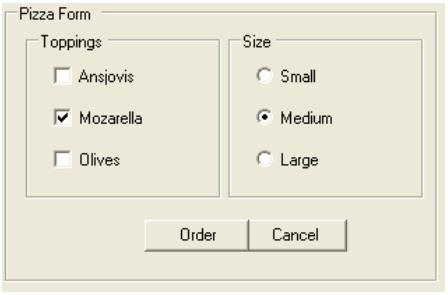
Zoals in sectie 5.4 bepaald, hangen de benodigde gegevens en technieken af van het abstractieniveau waarop we de adaptatie uitvoeren en is deze in UIML begrensd. We gaan hiervoor deze delen (uit het vorige deel) opsplitsen per niveau, te beginnen bij het laagste.

Niveau 1
De gebruikte widgets kunnen in de huidige versie van UIML aangepast worden, door een andere vocabulary (in de 'peers' sectie) voor een bepaald UIML document te gebruiken, waardoor de elementaire delen van een bepaalde classe gemapt worden op andere widgets. Echter zal men niet meer bereiken dan een licht aangepast uiterlijk van de widgets. Zo kan men bijvoorbeeld de elementaire delen van de klasse 'Button' mappen op een 'JButton' uit de Java Swing toolkit ter vervanging van een 'Button' uit de Java AWT toolkit, omdat deze op hun uiterlijk na op dezelfde manier gecreëerd en beheerd worden.
<peers> <presentation base="**JavaAWT_1.3_Harmonia_1.0.uiml**"> </peers> <peers> <presentation base="**JavaSwing_1.3_Harmonia_1.0.uiml**"> </peers>

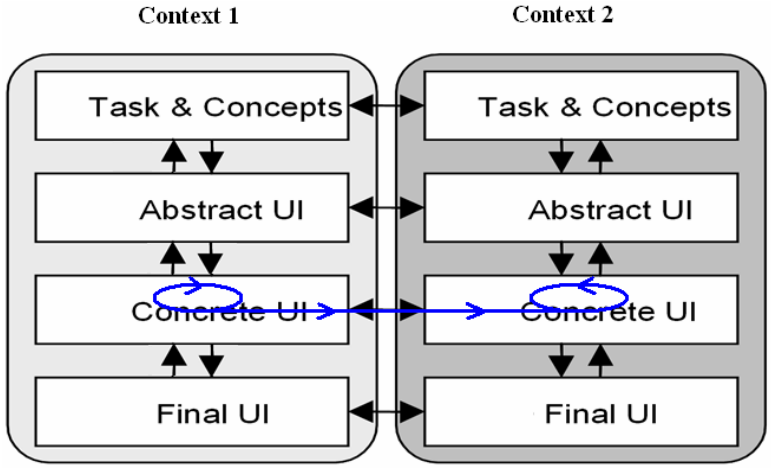


Figuur 37: Situering van adaptatieniveau 1 (gebaseerd op [83]).

Tabel 1: Adaptatieniveau 1.

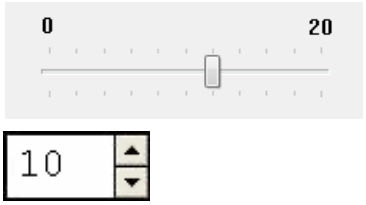
Niveau 2	
Eigenschappen aanpassen van de user interface onderdelen zonder wijzigingen aan te brengen aan zijn structuur of zijn componenten. Zo kan men de kleur, taalinstelling, notaties, grootte, en andere stijlattributen aanpassen.	
<property part-name="btnOrder" name="label"> <reference constant-name="**OrderEnglish**"/> </property> … <content> <constant id="OrderEnglish" value="Order"/> <constant id="OrderDutch" value="Bestel"/> </content> <property part-name="btnOrder" name="label"> <reference constant-name="**OrderDutch**"/> </property> **<property part-class="Frame" name="background"> 255,255,224 </property> <!--light yellow-->** … <content> <constant id="OrderEnglish" value="Order"/> <constant id="OrderDutch" value="Bestel"/> </content>	 
Probleem	**Oplossing**
1. Wat zijn de alternatieve waarden voor een (ordinaal) attribuut?	1. Polymorfische (zie 4.4.1.3) stijleenheden gebruiken voor constanten met ordinale waarden.
2. Welke gevolgen heeft het aanpassen van een bepaald attribuut? Bv. Is gele tekst nog leesbaar op een lichtgele	2. Constraintregels opstellen bij elke vocabulary (vb. achtergrondkleur moet minstens 20% afwijken van de voorgrondkleur). Voor het aanpassen van lay-out

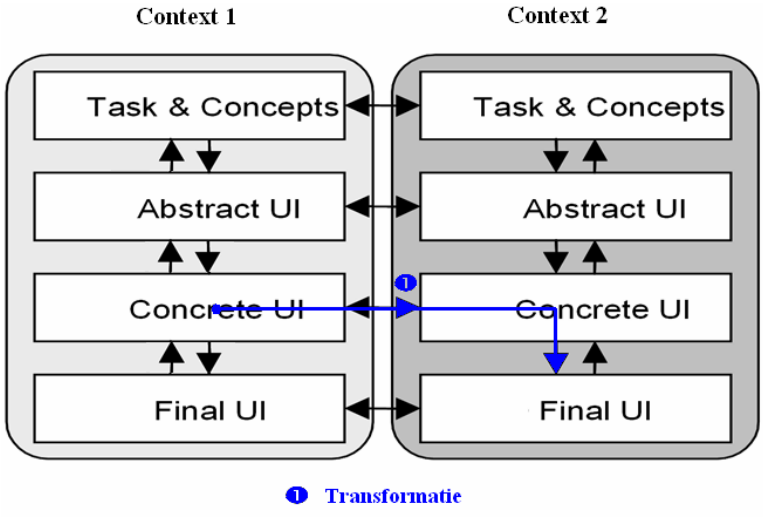
achtergrond? Overlappen objecten elkaar bij het aanpassen van hun grootte?	(positie, groottes) kan de lay-out manager voor UIML.NET gebruikt worden [77].
3. Welke aanpassing krijgt voorrang indien er meerdere aanpassingen gewenst zijn die incompatibel zijn met elkaar?	3. Constraint- en prioriteitsregels opstellen voor specifieke soorten aanpassingsmogelijkheden.



Figuur 38: Situering van adaptatieniveau 2 (gebaseerd op [83]).

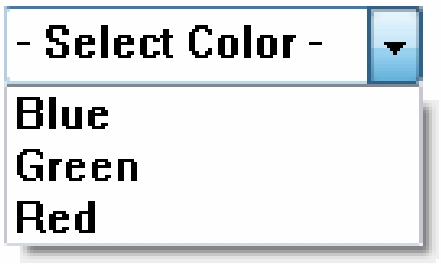
Tabel 2: Adaptatieniveau 2.

Niveau 3	
Een geavanceerdere vorm van adaptatie bestaat er in een ander widget te gebruiken voor 1 specifiek elementair deel uit een specifieke user interface. Hierbij kan er een ander soort widget gebruikt worden dan het oorspronkelijke. Bv. een slider veranderen naar een spinnerbox.	
<part id="speed" class="**HorizontalRange**"/> <part id="speed" class="**Spinner**"/>	
Probleem	**Oplossing**
1. Het systeem weet niet altijd welke taak een bepaald widget ondersteunt in de huidige user interface (bv. wordt de slider ook voor invoer van data gebruikt?).	1. Adapteerbare elementaire delen in de user interface introspectief maken (zie 4.4.1.2) zodat het systeem de ondersteunde taak of functie van het widget herkent.
2. Welke alternatieve widgets zijn er beschikbaar en welke zijn geschikt voor de ondersteuning van deze taak?	2. De widgetklassen uit de vocabulary reflectief maken (zie 4.4.1.1) zodat het systeem weet welke widgetklassen het kan gebruiken en welke daarvan geschikt zijn.
3. Kan het oorspronkelijk widget zomaar vervangen worden door het nieuwe? Welke gevolgen heeft dit? Een slider bv. weergeeft een waarde aan de hand van het 'value' attribuut, terwijl dit bij een spinnerbox gebeurt met het 'text' attribuut.	3. Bij elke vocabulary uitvoerbare transformaties voorzien tussen de widgets.

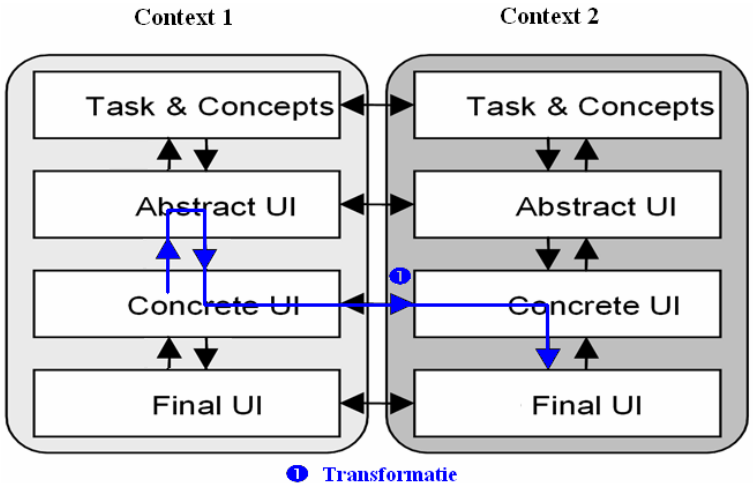


Figuur 39: Situering van adaptatieniveau 3 (gebaseerd op [83]).

Tabel 3: Adaptatieniveau 3.

Niveau 4	
Een andere implementatie gebruiken voor medium level patronen. Het toepassen van een alternatief implementatiepatroon voor een samengesteld deel (vergelijking met abstract interactie object) behoort ook tot dit niveau. Een samengesteld deel kan een samengesteld widget voorstellen dat bestaat uit een compositie van elementaire delen of widgets. Een voorbeeld hiervan is een reeks radiobuttons voor de selectie van een waarde uit een lijst. Een andere implementatie om deze taak te ondersteunen is bijvoorbeeld een listbox.	
<part id=”colorselect” class=”**Frame**”> <part id="Blue" class="**RadioButton**"> <style> <property name="label">Blue </property> </style> </part> <part id="Green" class="**RadioButton**"> <style> <property name="label">Green</property> </style> </part> <part id="Red" class="**RadioButton**"> <style> <property name="label">Red </property> </style> </part> </part> <part id=”colorselect” class=”**ComboBox**”> <style> <property name="content"> <constant model="list"> <constant value="Blue"/> <constant value="Green"/> <constant value="Red"/> </constant> </property> </style> </part>	 
Probleem	**Oplossing**
1. Het systeem weet niet welke samengestelde delen van de user interface een samengesteld widget (composite widget) beschrijven. Met andere woorden: het systeem kan niet uit het UIML document alleen identificeren welke samengestelde	1. en 2. Samengestelde delen van dit niveau introspectief maken zodat het systeem de ondersteunde taak of functie van dit deel herkent.

delen een patroon van dit niveau toepassen.	
2. Het systeem kan niet altijd afleiden welk implementatiepatroon (of samengesteld widget) is toegepast in een zodanig samengesteld deel.	
3. Welke alternatieve implementaties zijn er beschikbaar voor de ondersteuning van de taak? Of: welke implementatiepatronen zijn er beschikbaar voor dit medium level patroon?	3. De vocabulary uitbreiden met samengestelde widgets en deze reflectief maken zodat het systeem weet welke samengestelde widgets er beschikbaar zijn en wat hun functie is.
4. Hoe transformeren we het huidige deel naar het gewenste (al dan niet samengesteld) widget.	4. Transformaties voorzien tussen zowel samengestelde en elementaire widgets.



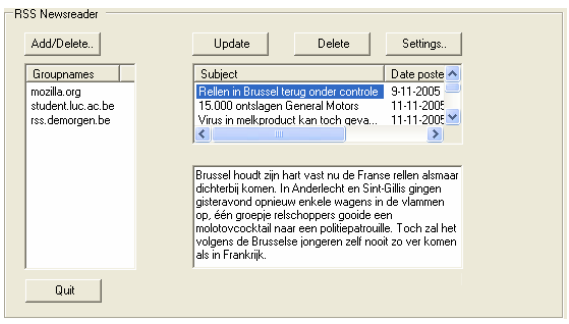
Figuur 40: Situering van adaptatieniveau 4 (gebaseerd op [83]).

Tabel 4: Adaptatieniveau 4.

Niveau 5
Een ander medium of high level patroon gebruiken ter ondersteuning van een (samengestelde) high level taak. Het komt er op neer dat een abstract tot zeer abstract interactie object wordt vervangen door een ander. Dit komt voor wanneer het abstracte interactie object of high level patroon niet plastisch genoeg is voor de huidige context. Het detail view navigatie patroon zal bijvoorbeeld niet bruikbaar zijn indien de gebruiker over een zeer beperkte schermgrootte beschikt (vb. mobiele telefoon).

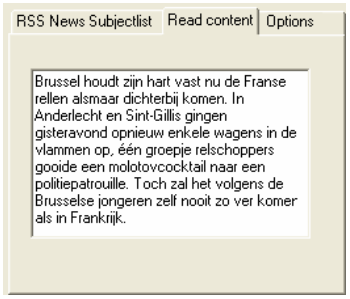
```
<part id=”main” class=”Frame” source="bigscreen"/>

<template id="bigscreen">
  <part class="Container" source="groups"/>
  <part class="Container" source="subjects"/>
  <part class="Container" source="content"/>
</template>
```

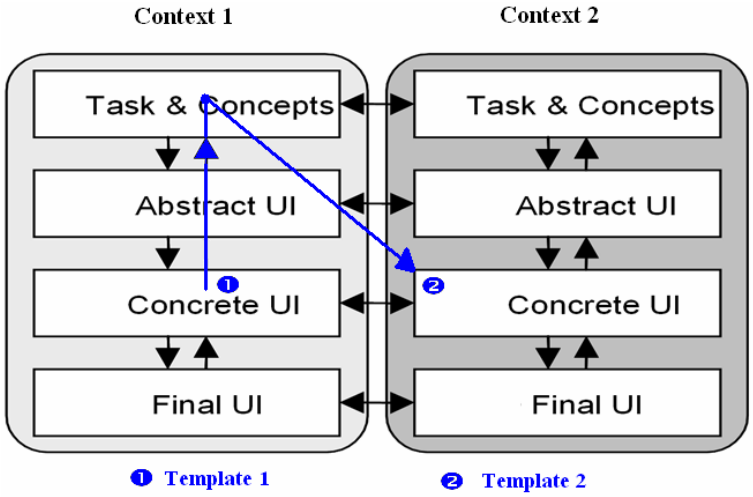


```
<part id="main" class="Frame" source="smallscreen"/>

<template id="smallscreen">
  <part class="Tabs">
    <part class="TabPage" source="groups"/>
    <part class="TabPage" source="subjects"/>
    <part class="TabPage" source="content"/>
  </part>
</template>
```



Probleem	Oplossing
1. Welke samengestelde delen horen tot dit niveau?	1. 2. 3. en 4. Samengestelde delen van dit niveau polymorfisch maken (zie 4.4.1.3) zodat het systeem andere abstracte uitwerkingen (met een ander high level patroon) voor deze delen kan selecteren.
2. Welk patroon passen ze toe?	
3. Welke high level patronen lossen het probleem op? Zijn deze voor handen?	
4. UIML is niet abstract genoeg om intelligente transformaties op dit niveau toe te passen (zie 5.4). Het systeem kan geen abstract model afleiden uit het UIML document om de transformatie toe te passen. Hierbij zou het systeem ook moeten beschikken over AIO → CIO transformaties tussen verschillende abstractieniveaus om het nieuwe abstracte deel te implementeren.	



Figuur 41: Situering van adaptatieniveau 5 (gebaseerd op [83]).

Tabel 5: Adaptatieniveau 5.

6.1.3 Metadata laag

Om de adaptaties op al de niveaus van het vorige deel te ondersteunen, verrijken we de deelnemende entiteiten met extra data (metadata). In dit deel wordt de implementatie hiervan besproken. We houden min of meer de volgorde uit het vorige deel aan (van niveau 1 t.e.m. niveau 5), maar technisch gezien komt dit niet volledig overeen en wordt dus van afgeweken.

Elementaire widgetklassen

Ondanks we niet veel aandacht gaan besteden aan adaptaties op het laagste niveau (1), waar we de mappings op concrete widgets van de finale user interface wijzigen door een andere vocabulary te gebruiken, is het noodzakelijk om deze entiteit meer betekenis te geven door haar elementen reflectief te maken. Dit is vooral van belang bij adaptaties van niveau 3 en 4. Het systeem dient hiervoor de inhoud van de vocabulary te 'kennen', dit wil zeggen: meer weten dan alleen de naam van een widgetklasse. De gegevens worden bij elke vocabulary meegeleverd en worden dus dynamisch ingelezen. Het document waarin deze gegevens vastgelegd worden heeft ongeveer dezelfde structuur als een DIDL-document (Digital Item Definition Language) van MPEG-21 (zie 3.2.3.1). Uit de onderzoeksresultaten (zie 5.6) voorzien we de volgende gegevens over een widgetclasse uit de vocabulary.

CLASS
(vb. ListBox)

Category	DataInput	Any
		SingleFromList
		MultipleFromList
	DataDisplay	Single
		Multiple
	Menu	
	Action	
	Grouping	
Datatype	textual / numeric / boolean	

Tabel 6: Functionele metadata van een 'class' element.

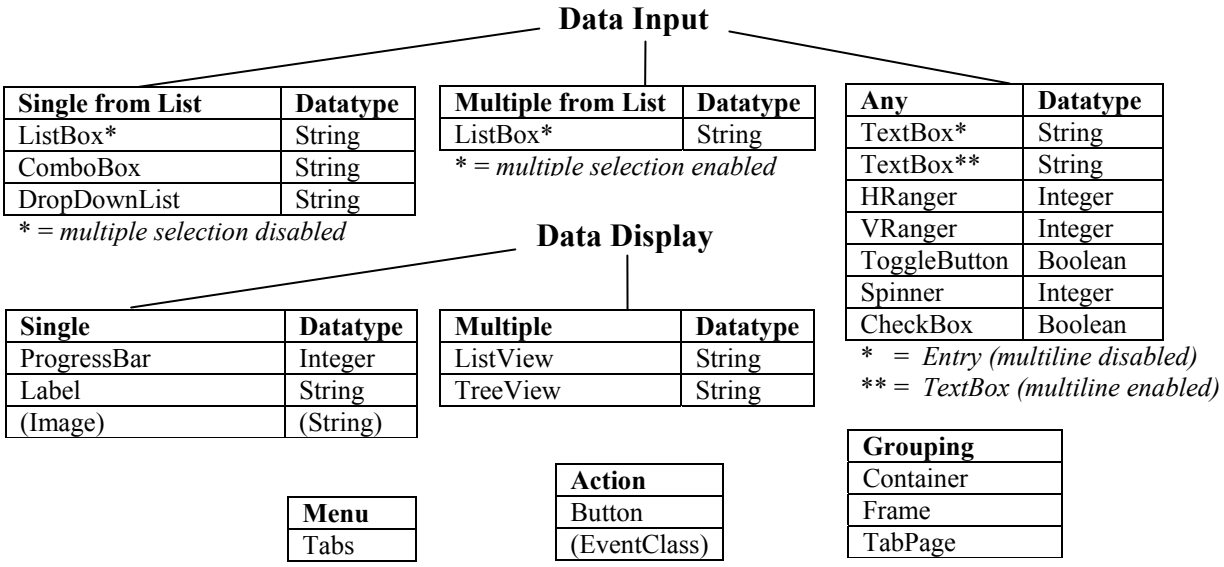
We onderscheiden 5 hoofdcategorieën van widgets naargelang hun functie:

- invoer van gegevens
  - vrije invoer (alle waarden mogelijk)
  - selectie van 1 waarde uit een lijst
  - selectie van 1 of meer waarden uit een lijst
- weergave van gegevens
  - weergave van 1 waarde
  - weergave van 1 of meer waarden
- menu implementatie
- uitvoeren van een actie
- groeperen van widgets (container)

Voor widgets van de categorie 'DataInput' en 'DataDisplay' wordt ook het datatype dat ze ondersteunen geannoteerd: tekstueel (string), numeriek (integer), booleaans (boolean). Dit zou men ook als een nominale subcategorie kunnen beschouwen, dit wil zeggen: string > integer > boolean. Een widget dat met tekstuele waarden overweg kan, kan dus ook numerieke en booleaanse waarden

verwerken. De gegevens worden in de ‘class repository’ opgeslagen door middel van een hashtable.

Op de Windows Forms vocabulary komen we tot een volgende onderverdeling:



Tabel 7: Overzicht van klassen in de SWF vocabulary.

Het betreft hier elementaire widgetklassen. Uiteraard zijn er nog andere mogelijkheden om een elementaire taak te ondersteunen. Hier komen we later in dit deel op terug.

‘Reference’ elementen

Om adaptaties van het 2e niveau te ondersteunen, waarbij we eigenschappen van de user interface onderdelen aanpassen, voorzien we extra informatie over de ‘reference’ elementen in het UIML document. Deze informatie wordt in een DIDL (zie 3.2.3.1) gelijkend document gestoken dat hoort bij de user interface.
Een eerste mogelijkheid is de volgende:

REFERENCE

constant-name	speedMiles	speedKilometers	…

Tabel 8: Metadata van een ‘reference’ element.

De bovenstaande tabel bevat 2 referenties naar constants waarvan hun id kan ingevuld worden in het *constant-name* attribuut.

Het systeem kan dus de waarde ‘speedMiles’ of ‘speedKilometers’ gebruiken.
<property part-name=”speedlabel” name=”label”><reference constant-name=”**speedKilometers**”/></property>

Het is moeilijk om het ‘reference’ element met metadata te annoteren omdat de UIML specificatie geen ‘id’ attribuut voorziet voor dit element. Daarom kan er niet naar gerefereerd worden.

Alternatieve manieren die gebruikt kunnen worden zijn:

1) PROPERTY

value	<reference constant-name=”speedKilometers”/>	<reference constant-name=”speedMiles”/>	Knots	…

Tabel 9: Metadata van een ‘property’ element.

We hebben hier hetzelfde probleem daar het ‘property’ element ook geen ‘id’ attribuut mag hebben.

2) CONTENT

source	EnglishMetrics	EuropeanMetrics	…

Tabel 10: Metdata van een ‘content’ element.

Dit is de enige praktische oplossing en heeft als voordeel dat men op 1 plaats een aanpassing kan doen die gevolgen heeft voor heel de user interface (vb. gebruik van Engelse maten).

```
<part class="Label">
  <reference constant-name="speedUnit">
</part>
...
<content source=”EnglishMetrics”/>
...
<template id="EnglishMetrics">
  <constant id="speedUnit"  value="mph"/>
  <constant id="lengthUnit" value="mile"/>
</template>
```

```
<part class="Label">
  <reference constant-name="speedUnit">
</part>
...
<content source=”EuropeanMetrics”/>
...
<template id="EuropeanMetrics">
  <constant id="speedUnit"  value="kmph"/>
  <constant id="lengthUnit" value="km"/>
</template>
```

Elementaire ‘Part’ elementen

Met de gegevens die we tot nu toe verzameld hebben kan het systeem de functie van een elementair widget (instantie van een widgetclasse) herkennen. De widgetklassen in de vocabulary hebben we immers reflectief gemaakt. In een UIML document dat opgesteld is voor de Windows Forms vocabulary zijn dit ‘part’ elementen met een ‘class’ attribuut waarvan de waarde niet tot de categorie ‘Menu’ of ‘Groupings’ behoort. Toch kan een dergelijk widget of part in verschillende user interfaces verschillende functies hebben. Een ‘Slider’ kan gebruikt worden voor invoer van gegevens maar ook voor strikt gegevens weer te geven. In het laatste geval zou deze dus geconverteerd kunnen worden naar een ‘Label’, dat alleen data kan weergeven. Ook het datatype dat een widget in de huidige user interface moet ondersteunen kan invloed hebben op het aantal adaptatiemogelijkheden. Een dropdown list kan ook gebruikt worden om booleaanse waarden in te voeren (vb. dropdown list met de items ‘met korting’ , ‘zonder korting’). In dit geval kan een checkbox deze taak overnemen en hoeven niet noodzakelijk widgets gebruikt te worden die het type ‘String’ ondersteunen. Om de volledige potentie van adaptatiemogelijkheden te kunnen benutten dient het mogelijk te zijn deze specifieke informatie aan een instantie van een widget toe te kennen. Deze informatie dient eveneens te worden toegevoegd aan het beschrijvend DIDL gebaseerd document van de user interface.

PART

Category	DataInput	Any
		SingleFromList
		MultipleFormList
	DataDisplay	Single
		Multiple
	Menu	
	Action	
	Grouping	
Datatype	textual / numeric / boolean	

Tabel 11: Functionele metadata van een 'part' element.

Uiteraard is deze informatie niet verplicht, maar kan het aantal mogelijke transformaties vergroten. De informatie wordt volgens een union-principe met die van de gebruikte widgetklasse geïnterpreteerd. Bijvoorbeeld: Een instantie van een widgetklasse van het type: DataInput (Any) , textual; dat wordt geannoteerd met de metadata: DataInput (SingleFormList), boolean; mag ook naar een widgetklasse van het type: DataInput (SingleFormList), numeric worden geconverteerd (en is niet meer beperkt tot het type 'textual'). Hierbij worden volgende nominale regels gebruikt:

- DataInput (Any) > DataDisplay (Single)
- DataInput (MultipleFromList) > DataDisplay (Multiple)
- DataInput (SingleFromList) > DataDisplay (Multiple)
- Textual > Numeric > Boolean

Samengestelde 'Part' elementen met low tot medium level patronen

Om elementaire taken te ondersteunen (bv. selectie van waarden uit een lijst), kan men naast 1 elementaire widgetklasse ook een compositie van meerdere widgets gebruiken. Bijvoorbeeld een reeks checkboxen. Dit noemt men een samengesteld widget. Om deze widgets te kunnen transformeren dient het systeem deze te herkennen. Uit een reeks concrete interactieobjecten moet dus een abstract interactie object worden afgeleid (zie schema adaptatie op niveau 4). Dit wordt gedaan door patroon extractie: het systeem gaat aan de hand van de metadata van een dergelijk samengesteld deel, de taak en het gebruikte patroon voor die taak herkennen, om een gepaste transformatie te kunnen toepassen. In UIML documenten die opgesteld zijn voor de Windows Forms vocabulary kan het gaan om de 'part' elementen met een 'class' attribuut met de waarde 'Frame', 'Container' en eventueel 'TabPage', hoewel dit bij deze laatste zelden zal voorkomen. Om het systeem deze delen te laten herkennen wordt deze informatie toegevoegd aan het beschrijvende definitie document van de user interface.

PART

Class (abstract)	RadioButtons / DropdownMenu / TabPages / …

Tabel 12: Metadata van een 'part' element met samengestelde delen.

Deze gegevens kunnen ook worden aangevuld met metadata die voor elementaire delen worden gebruikt om hun specifieke functie te verduidelijken, maar dit is uiteraard ook hier niet verplicht.

Het gaat hier om iets abstractere widgetklassen die niet in de huidige vocabulary voorkomen. Hierin zitten tot nu toe alleen concrete elementaire widgetklassen. Omdat het systeem moet 'weten' waarvoor deze abstracte widgetklassen kunnen gebruikt worden en naar welke hiervan getransformeerd kunnen worden, moeten deze toegevoegd worden aan (de metadata van) de vocabulary, om in de classrepository te worden opgeslagen. In onze toepassing gaan we de Windows Forms (meta-)vocabulary uitbreiden met de volgende widgetklassen:

Data Input

Single from List	**Datatype**
RadioButtons	String

Multiple from List	**Datatype**
CheckBoxes	String
ToggleButtons	String

Menu
Buttons*
ListMenu**
DropDownMenu***
TreeView*

Data Display

Multiple	**Datatype**
Labels	String

* : met action event
** : ListView met action event
*** : DropDownList met action event

Tabel 13: Overzicht van samengestelde klassen in de uitgebreide SWF vocabulary

Samengestelde 'Part' elementen met high level patronen

Bij adaptaties van niveau 5 wordt er een ander high level patroon toegepast voor (een deel van) de user interface, omdat het huidige patroon bijvoorbeeld niet plastisch genoeg is voor de huidige context, ondanks de meerdere implementatiemogelijkheden. Hier kunnen geen universele transformaties voor opgesteld worden zonder UIML uit te breiden met een abstract model, dat even uitgebreid zou zijn als andere abstracte talen. Het is eveneens praktisch onmogelijk om deze automatisch af te leiden, omdat het verschil in abstractieniveau tussen de UIML specificatie en dit model te groot is. Bijvoorbeeld om een nieuwe interactietechniek gebaseerd op de wizard metafoor te gebruiken in plaats van een formulier metafoor met directe manipulatie, zal men voor elke van deze high level patronen een aparte template moeten samenstellen. In elk van de deze templates kan een gedeeltelijk of volledig uitgewerkt deel van de user interface beschreven worden, maar het kan ook gaan over stijl-templates of lay-out templates die de verschillende delen van de user interface anders schikken of de user interface verdelen over meerdere schermen (voor apparaten met een klein scherm bijvoorbeeld). Het kan ook gaan om andere low level patronen die niet automatisch getransformeerd kunnen worden, bijvoorbeeld om een tekstlabel te vervangen door een afbeelding. In UIML documenten die opgesteld zijn voor de Windows Forms vocabulary kan het gaan over 'part' elementen die een template importeren met behulp van het 'source' attribuut. Meestal hebben ze ook een 'class' attribuut met de waarde 'Frame', 'Container' of 'TabPage', maar dit hoeft niet noodzakelijk zo te zijn (bv. bij stijl-templates). Voor dergelijke delen voorzien we de volgende extra informatie die eveneens opgeslaan wordt in het beschrijvende DIDL-gebaseerde document dat hoort bij de user interface.

PART

source	wizardtemplate	formtemplate1	formtemplate2	…

Tabel 14: Metadata van een 'part' element dat een template importeert.

De bovenstaande tabel bevat de alternatieve waarden die kunnen gebruikt worden voor het 'source' attribuut voor het betreffende 'part' element.

Deze vorm van adaptatie wordt toegepast indien het automatisch intelligent opstellen van transformaties onmogelijk of te ingewikkeld is. In principe kan deze techniek dus toegepast worden op alle adaptatieniveaus, maar moet daarvoor per user interface geformuleerd worden.

Transformaties

Bij het selecteren van een geschikte uitwerking of klasse voor een deel van de user interface, dient het systeem te beschikken over transformaties die het kan uitvoeren om van de huidige naar de nieuwe concrete uitwerking of klasse te transformeren. Voor adaptaties van niveau 1, 2 en 5 (vocabulary, properties, templates) volstaat het om een attribuutwaarde te wijzigen. Maar vooral voor adaptaties op niveau 3 en 4 (gebruik van andere widgets) is er op meer dan één plaats in de user interface aanpassing nodig (structuur kan worden gewijzigd, afhankelijke onderdelen dienen eveneens te worden aangepast,…), en vertrekt men vanuit meerdere delen die in de transformatie inbegrepen worden. De elementaire en samengestelde widgetklassen hebben voor elke user interface een unieke configuratie. Toch kunnen er transformaties voor worden gebruikt omdat ze in alle user interfaces dezelfde elementaire subtaak ondersteunen en een systematische structuur hebben. Elke mogelijke "huidige klasse → nieuwe klasse" transformatie moet worden meegeleverd met de vocabulary.

Voor UIML.NET zijn er 2 interessante manieren voor de opstelling en uitvoering van deze transformaties. We bespreken kort beide manieren en wegen daarna de voor- en nadelen af.

1e manier: XSLT.
XSLT staat voor eXtensible Stylesheet Language Transformations en wordt gebruikt om XML-gebaseerde documenten te transformeren naar één of meer andere XML-gebaseerde documenten. Het nieuwe document kan in een andere of dezelfde XML syntax beschreven zijn als het bron document. XSLT is zelf een XML-gebaseerde taal, waardoor transformaties los van de applicatie kunnen samengesteld worden.

2e manier: interne UIML boom of 'UI tree' transformeren.
Bij deze aanpak gaan we de door UIML.NET opgebouwde 'nodetree' van het UIML document transformeren met programmatische transformatiemodules.

	XSLT (op UIML document)	**Transformatiemodules (op UI tree)**
Tijdstip	Voor het inlezen van het UIML document. UIML.NET krijgt een aangepast UIML document als invoer.	Na of tijdens het inlezen van het UIML document. Anders gezegd: na of tijdens de opbouw van de UIML tree.
Dynamiek	XSL Transformaties kunnen dynamisch ingeladen en aangepast worden zonder de applicatie aan te passen.	De transformatiemodules moeten voor elke front-end (bv. SWF) geprogrammeerd worden.
Flexibiliteit	XSLT documenten dienen universeel toepasbaar te zijn en gaan uit van een bepaalde standaardspecificatie. Ze kunnen complex worden door de verschillende manieren om bijvoorbeeld 'properties' aan de 'part' elementen toe te wijzen. Deze kunnen in een <part> </part> sectie gedefinieerd zijn waarop nog eens 'style' templates van	Met transformatiemodules kunnen we gebruik maken van de dynamischere C# programmeertaal. Hierin kunnen condities, berekeningen, functies, procedures en dynamische expressies gebruikt worden. De UI tree is gemakkelijker te manipuleren door functie aanroepen waardoor de transformaties eenvoudiger zijn en toch

	toepassing kunnen zijn, maar ook in een algemene <style> </style> sectie onderaan. XSLT is ook beperkt door waarden die nog 'opgelost' moeten worden. Bv. <property part-name="p5" name="label"> <property part-name="p1" name="text"/> </property> Als deel p1 onafhankelijk van deel p5 blijkt te worden getransformeerd naar een deel zonder 'text' attribuut, zal dit fouten opleveren voor deel p5.	verschillende mogelijke afhankelijke onderdelen in rekening brengen. Zo kunnen er ook gemakkelijk pre- en postbewerkingen toegepast worden.

Tabel 15: Vergelijking tussen XSLT en transformatiemodules.

Men zou een combinatie van beide kunnen gebruiken om hun sterkste punten te benutten, maar de grootste voordelen (dynamiek en tijdfase) van XSLT worden hiermee teniet gedaan.

In dit onderzoek gaan we de transformatiemodules gebruiken voor al de adaptaties. Dit omdat XSL Transformaties snel complex worden en we met kwesties te maken hebben die er niet zijn bij gebruik van transformatiemodules. Later kunnen een aantal relatief eenvoudige transformaties vrij gemakkelijk naar XSLT worden omgezet.

Samenvatting

In dit eerste deel hebben we de mogelijkheid tot adaptatie gerealiseerd door de vocabulary uit te breiden met:

- abstracte samengestelde widgetklassen
- semantische functionele informatie over de widgetklassen
- transformaties tussen de widgetklassen

De user interface wordt uitgebreid met:

- semantische en functionele informatie over de verschillende onderdelen van de user interface ('part' elementen)
- referenties naar alternatieve waarden van attributen waardoor elementen een polymorfisch karakter krijgen ('content' en 'part' elementen).

Dit is net zoals de UIML taal een metamodel waarvan de informatie elementen en transformaties vrij kunnen opgesteld en gebruikt worden door de auteur van het UIML document. Dankzij deze informatie heeft het systeem een begrip over de samenstelling van de user interface en samengestelde widgets, zoals dit bij het Digital Item Declaration principe van MPEG-21 (zie 3.2.3.1) bereikt wordt met multimedia content. Deze gegevens worden later gebruikt om de transformaties te ondersteunen.

6.2 Modellering van de natuurlijke gebruikerscontext

De transformatiekeuze dient natuurlijk te gebeuren in functie van de natuurlijke context. We bespreken hier hoe we deze natuurlijke context modelleren en welke meta-informatie er nog over de user interface nodig is om een transformatie juist te kiezen. Een adaptatiekeuze is juist wanneer deze de meest geschikte oplossingsconfiguratie van de user interface voor de huidige context als gevolg heeft.

6.2.1 Gebruiksomgeving

Zoals besloten in sectie 5.5 gaan we data over gebruiker, apparaat en omgeving als één gebruiksomgeving beschouwen. Er is geen beperking op het aantal parameters of attributen die men gebruikt om de context te beschrijven. In het Ubis World project waar GUMO of het General User Model and Context Ontology (zie 3.2.2.3) wordt gebruikt, zijn een groot aantal aspecten en dimensies gedefiniëerd. Toch kunnen deze nog uitgebreid worden. Ondanks de definiëring van deze attributen vrij staat, is het ten zeerste aanbevolen elementaire factoren te gebruiken die rechtstreeks verband houden met andere factoren die de eindconfiguratie beïnvloeden. Zo zal men best informatie opslaan over de pointing mogelijkheden van de gebruiker in plaats van het al dan niet beschikken over een bepaald invoerapparaat. Als het systeem weet dat de gebruiker een muis ter beschikking heeft, moet deze hier uit kunnen afleiden dat pointing interactie mogelijk is. Natuurlijke omgevingsaspecten zijn minder 'zwart-wit' dan technische aspecten van de user interface. Er bestaan verschillende gradaties en tussenwaarden in natuurlijke factoren en ook deze moeten worden uitgedrukt op een door het systeem interpreteerbare manier. Hiervoor dienen we 'ranges' te gebruiken op de elementen die ook in het GUMO model gebruikt worden. In GUMO kan men uitdrukken dat een persoon een intuïtie heeft (User:Personality:intuiting), maar het is moeilijk om de mate van deze eigenschap uit te drukken. Men zou de 'Rating Dimensions' en 'Ranges' van GUMO kunnen gebruiken, maar deze zijn echter niet meer dan objecten zonder vastgelegde betekenis. Men kan technisch niet bepalen wanneer de waarde 'poor' of 'from 1 to 6' van toepassing is. Een numerieke of evalueerbare waarde is hier meer op zijn plaats.

Voor onze toepassing gaan we voor de gebruikersomgeving een contextmodel opstellen waarbij we over de gebruiker en het apparaat de volgende informatie opnemen:

<USER>

Visual_minimumhightwidth	12...30
Visual_colordeficiency*	green/blue/red/..
Visual_detail	0,0...1,0
Motoric_finesse	0,0...1,0
Semantic_understanding	0,0...1,0
Preference_Interaction_Oneclick	0,0...1,0
Preference_Interaction_Keyintput	0,0...1,0
Preference_Interaction_Drag	0,0...1,0
Preference_visual_color_foreground	green/blue/red/...
Preference_visual_color_background	green/blue/red/...
Preference_Content_Language_English	0,0...1,0
Preference_Content_Language_French	0,0...1,0
Preference_Content_Language_Dutch	0,0...1,0
Preference_Content_Type_Text	0,0...1,0
Preference_Content_Type_Multimedia	0,0...1,0
Preference_Content_Style_Long	0,0...1,0
Preference_Content_Style_Abreviated	0,0...1,0
Preference_Content_Style_Symbolic	0,0...1,0

Tabel 16: Metadata van de gebruiker.

Zoals eerder gezegd is dit de informatie die de adaptatiecomponent gaat gebruiken en kan afgeleid zijn van andere metadata over de gebruiker.

Bijvoorbeeld:

<hasVisualImpairment>0.6</hasVisualImpairment>
--> Visual_minimumfontsize = 12
--> Visual_detail = 0,4
--> Visual_minimumhightwidth = 20

<livesIn>Linkebeek</livesIn>
--> Preference_Content_Language_Dutch = 1,0
--> Preference_Content_Language_French = 0,8

<isDoingActivity><act:DrivingCar/><isDoingActivity/>
--> Motoric_finesse = 0,3

De ranges van 0,0 tot 1,0 kunnen geïnterpreteerd worden als een gewicht dat zegt in welke mate 'de uitspraak' klopt. Of: hoe belangrijk het aspect is voor de persoon. Hoe hoger het gewicht (bij sommige eventueel lager, bv Motoric_finesse), hoe sterker dit aspect de adaptatie zal beïnvloeden. Hier wordt later verder op ingegaan.

Over het hardwareplatform wordt de volgende informatie voorzien.

<DEVICE>

Visual_screenwidth	128...1024
Visual_screenheight	128...768
Visual_screencolordepth	highdefcolor/16-bitcolor/256-color/b&w/monolitic
Multimedia_image	jpeg/gif/jpeg&gif/none
Input_keys	none/numeric/compact/full
Input_pointing	fine/medium/low/none

Tabel 17: Metadata van een apparaat.

Ook dit kan afgeleide informatie zijn.

Bijvoorbeeld:

<hasInputDevice>
<Mouse/> --> Input_pointing = fine (10) <TouchScreen/> --> Input_pointing = medium (7)
<Scroller/> -> Input_pointing = low (2) (niets) --> Input_pointing = none (0)
<keyBoard/>
<isOfType>alphanumeric</isOfType> --> Input_keys = full (10)
<isOfType>compact</isOfType> --> Input_keys = compact (6)
<isOfType>numeric</isOfType> --> Input_keys = numeric (3)
(geen keyBoard) --> Input_keys = none (0)

Dit model zou voor de meeste user interfaces bruikbaar moeten zijn doordat we generische en elementaire parameters gebruiken. Maar zoals eerder vermeld staat het vrij een ander model te definiëren en te gebruiken en liggen deze niet vast in de programmacode. Niet al deze parameters

moeten gedefinieerd zijn, maar hoe meer er zijn gedefinieerd, hoe beter de adaptatie op de context zal afgestemd zijn. Deze meta-informatie wordt in hetzelfde document opgenomen als dat van het gebruikersprofiel.

6.2.2 User interface

In deel 6.1.1.3 hebben we transformaties opgesteld en functionele attributen van user interface onderdelen kenbaar gemaakt aan het systeem. Het systeem is dus tot nu toe in staat om adaptaties uit te voeren maar heeft geen enkele kennis over het resultaat of het 'effect' van een dergelijke adaptatie. Om het 'effect' te berekenen dient het systeem te beschikken over de semantische eigenschappen van de onderdelen van de user interface, en de widgetklassen uit de vocabulary.

Elk onderdeel dat betrokken kan worden bij adaptatie, wordt geannoteerd met semantische informatie om de adaptatie te sturen. Dit wordt min of meer op dezelfde manier gedaan als de modellering van de natuurlijke omgeving. De parameters voor dit model zijn vrij te kiezen door de auteur van het UIML document. In onze toepassing stellen we een reeks parameters op die bruikbaar zouden moeten zijn voor de meeste toepassingen.

CLASS
(zowel enkelvoudige als samengestelde 'classes')

TEMPLATE
(geïmporteerd door delen van niveau 5)

Interaction_effort_pointing	0..10
Interaction_effort_keystroke	0..10
Interaction_style_oneclick	0..10
Interaction_style_keyinput	0..10
Interaction_style_drag	0..10
Interaction_speed	1..10
Visual_size_width	0..10 / (0..1024)*
Visual_size_height	0..10 / (0..1024)*
Visual_finedetail	1..10
Visual_semantic	1..10
Output_required_color	0..1
Output_required_multimedia	0..1

* : alleen bij lay-out templates

Tabel 18: Semantische metadata van een 'class' of 'template' element.

Afhankelijk van de inhoud van de template of widgetklasse zullen sommige aspecten van de tabel niet aanwezig zijn of geen waarde hebben. De informatie wordt in het beschrijvend document van de user interface opgenomen. Uiteraard hoeven niet alle parameters gedefinieerd te worden, hoewel dit bij widgetklassen wel aangeraden is.

Er kan opgemerkt worden dat de waarden voor deze parameters een subjectief karakter hebben en deze meestal niet ondubbelzinnig 'gemeten' kunnen worden. Anderzijds is er de noodzaak om deze waarden exact en concreet voor te stellen om door een computersysteem geïnterpreteerd te worden. Om gewenste adaptaties te bereiken is het aangewezen deze waarden op een schaal te ijken en indicatoren op te stellen om te bepalen welke waarde van toepassing is. Dit is

een wetenschap op zich. We gaan hier daarom niet te diep op in. We komen in het laatste deel kort terug op deze kwestie.

Informatie bij de parameters	
Interaction_effort_pointing	Geeft aan hoe precies (exact) er moet gepoint worden
Interaction_effort_keystroke	Geeft aan hoeveel er handmatig moet ingetoetst worden.
Interaction_style_oneclick	Geeft aan in welke mate er 'one-click' acties vereist zijn.
Interaction_style_keyinput	Geeft aan in welke mate er 'keyinput' acties vereist zijn.
Interaction_style_drag	Geeft aan in welke mate er sleep of 'drag' acties vereist zijn.
Interaction_speed	Geeft de snelheid vd. interactie (efficiëntie) aan. (voor voltooiing)
Visual_size_width	Geeft de breedte aan van de footprint.
Visual_size_height	Geeft de lengte aan van de footprint.
Visual_finedetail	Geeft aan in welke mate er kleine visuele details gebruikt worden.
Visual_semantic	Geeft aan in welke mate de metafoor/weergave begrijpbaar is.
Output_required_color	Geeft aan of er kleurweergave vereist is.
Output_required_multimedia	Geeft aan of er multimediaweergave vereist is.

Tabel 19: Uitleg bij de semantische parameters.

Voor ‘constant’ elementen en templates met lexicale inhoud (deze bezitten hoofdzakelijk ‘constant’ elementen) wordt de volgende metadata voorzien.

CONSTANT

TEMPLATE
(geïmporteerd door content elementen)

Language	English/French/Dutch/none
Type	Text/Image:jpeg/Image:gif
Style	Long/Abreviated/Symbolic

Tabel 20: Semantische metadata van een ‘constant’ of ‘template’ element.

Deze meta-informatie zal vooral nuttig zijn voor adaptaties van niveau 2 (inhoud) en niveau 5 (stijl). De waarden van de attributen zijn slechts voorbeelden. Voor een stijl-template zouden de waarden voor de parameter ‘Type’ bijvoorbeeld ‘pastelcolors’, ‘richcolors’, ‘highcontrastcolors’ en ‘blackandwhitecolors’ kunnen zijn. Voor templates met constanten voor meeteenheden kan ‘Language’ gebruikt worden voor ‘EnglishMetrics’, ‘EuropeanMetrics’, enz..

Om intelligente adaptaties van niveau 2 (configuratie van de user interface ‘properties’) toe te passen, is het interessant om metadata op te slaan over de eigenschappen (‘style properties’) van een widgetklasse. Indien de gebruiker bijvoorbeeld voorkeur geeft aan een ‘licht’ of fris uiterlijk, is het nuttig om het systeem te laten weten welke paramaters hier invloed op hebben. Andere aspecten die verband houden met eigenschappen van widgetklassen zijn ‘notatievoorkeur’, ‘grootte’, ‘voorkeur voor hulpondersteuning’.

Er zijn andere regels vereist om de waarde van een dergelijk ‘property’ te bepalen dan voor andere elementen. We gaan hier in de toepassing niet op in. Meestal hebben de ‘properties’ van elke widgetklasse dezelfde naam voor hetzelfde aspect, op enkele uitzonderingen na (vb. voor een waarde weer te geven wordt er afhankelijk van de widgetklasse de attribuutnaam ‘value’, ‘content’, ‘text’ of ‘label’ gebruikt). Zoals eerder ondervonden (zie 6.1.2) dient er rekening gehouden te worden met beperkingen op combinaties van waarden tussen verschillende ‘properties’ onderling. Zo mag de tekst bv. best niet dezelfde kleur hebben als de achtergrond. De uitgewerkte toepassing

biedt wel de mogelijkheid om expliciete voorkeuren in te stellen en op basis hiervan autonoom de user interface te personaliseren. Voor een gebruiker die graag lichtblauwe achtergronden heeft, zal het systeem de juiste parameters wijzigen om dit te bekomen, en hierbij rekening houden met eventuele ongewenste gevolgen.

CLASS

Aspect	**property**
Content	(vb content)
Visual_Size_Width	(vb width)
Visual_Size_Height	(vb height)
Visual_Color_BackGr / Look	(vb BackColor)
Visual_Color_ForeGr / Look	(vb ForeColor)
Enabling / Help	(vb Enabled en Visible)
Interaction	(vb OnClick)

Tabel 21: Metadata van 'class' attributen.

6.3 Mappings en beslissingsprocedures

Het systeem beschikt naast transformaties over reflectieve toolkits, introspectieve user interfaces en metaobjecten, maar daar stopt het. De informatie over entiteiten die belang hebben in het adaptatieproces, de context genaamd, wordt nog niet gebruikt om geschikte transformaties te bepalen omdat de verbanden tussen deze entiteiten niet gekend zijn. Tot nu toe wordt er enkel bepaald of een transformatie technisch mogelijk is. Bepalen welke transformatie het meest geschikt is voor een gebruiker op basis van natuurlijke parameters, is een moeilijke zaak. Dit heeft vooral te maken doordat er van nature niet-formele verbanden, formeel moeten vastgelegd worden.

Een andere uitdaging is de context die uit meerdere dimensies bestaat. Een verband tussen een aspect van de gebruikersomgeving en een aspect van de user interface bestaat meestal onder voorwaarden van aspecten uit andere dimensies van de context. Er zijn verbanden die op elkaar inwerken en zelfs tegenspreken. De vraag in zo een geval is: "Welk verband heeft prioriteit?". Dit zijn natuurlijke keuzen die we als mens maken, soms onbewust, maar meestal zeer doeltreffend in zijn geheel. Het systeem zou moeten kunnen denken als de mens door sociale kennis en alfawetenschappen toe te passen. De implementatie van deze verbanden en evaluatietechnieken om hieruit te bepalen of er adaptatie nodig is en welke transformatie er in dat geval uitgevoerd wordt, wordt in dit onderdeel besproken.

6.3.1 Formalisatie van verbanden door mappings

Om de relatie tussen een aspect uit de natuurlijke buitenwereld (gebruiker, apparaat, omgeving) en een aspect van de user interface vast te leggen gebruiken we het volgende mapping model.

Aspect van de gebruiksomgeving ⟶ Aspect van de user interface

Bijvoorbeeld: Motoric_finesse ⟶ Interaction_effort_pointing

De pijl mapt het aspect Motoric_finesse op het aspect Interaction_effort_pointing en geeft aan dat er een verband is tussen de motorieke capaciteit van de gebruiker en de vereiste richtprecisie voor de interactie met de user interface. Hiermee gaan we naar hetzelfde principe als het Digital Item Adaptation principe van MPEG-21 (zie 3.2.3.1), waar gebruikersattributen gelinkt worden aan attributen van de digital items. Doordat we deze aspecten uitdrukken met attributen van een entiteit, gaan we vanaf nu de term attribuut gebruiken.

Er bestaan in grote lijnen 2 soorten attributen van de context:

- Attributen die een beperking opleggen.

  Bv. Multimedia_image ⟶ Output_required_multimedia
  Indien er geen multimediaweergave (afbeeldingen) ondersteund wordt, kunnen er geen afbeeldingen gebruikt worden in de user interface.

- Attributen die een invloed hebben op de bruikbaarheid van de user interface.

  Bv. : Semantic_understanding ⟶ Visual_semantic
  Een gebruiker die weinig ervaring heeft met elektronische user interfaces en weinig kennis heeft over de functie van widgets, zal delen van user interfaces met bepaalde symbolen of interactiewerkwijzen die niet voor de hand liggen, minder geschikt maken.

Welk verband een mapping uitdrukt dient eveneens formeel vastgelegd te worden. We onderscheiden de volgende soorten expressies en drukken deze met een functie uit:

Bij attributen die een beperking opleggen wordt de voorwaarde uitgedrukt met een operator:

- de waarde van het attribuut moet groter zijn : >
- de waarde van het attribuut moet kleiner zijn : <
- de waarde van het attribuut moet verschillend zijn : !=
- de waarde van het attribuut moet gelijk zijn : =

Bij attributen die een variabele invloed hebben wordt het verband uitgedrukt met een functie:

- er is een recht evenredig positief verband : x · y
- er is een omgekeerd evenredig positief verband : (1 – x) · y
- er is een recht evenredig negatief verband : -1 (x) · y
- er is een omgekeerd evenredig negatief verband : -1 (1 – x) · y

Een positief of negatief verband wil zeggen dat de waarde van het attribuut de bruikbaarheid respectievelijk ten goede komt of doet afnemen.

Een recht evenredig of omgekeerd evenredig verband wil zeggen dat respectievelijk hoe hoger of hoe lager de waarde van het attribuut is, hoe groter de invloed er van is.

Voor onze toepassing stellen we de volgende mappings op:

Context parameter (omgeving)	**Mapping functie**	**Context parameter (user interface)**
Visual_detail	*f*(-1 (1 – x))	Visual_finedetail
Motoric_finesse	*f*(-1 (1 – x))	Interaction_effort_pointing
Semantic_understanding	*f*(-1 (1 – x))	Visual_semantic
Semantic_understanding	*f*(x)	Content_style
Preference_Interaction_Oneclick	*f*(x)	Interaction_style_oneclick
Preference_Interaction_Keyinput	*f*(x)	Interaction_style_keyinput
Preference_Interaction_Drag	*f*(x)	Interaction_style_drag

Preference_Content_Language_English	*f*(x)	Content_Language
Preference_Content_Language_French	*f*(x)	Content_Language
Preference_Content_Language_Dutch	*f*(x)	Content_Language
Preference_Content_Type_Text	*f*(x)	Content_Type
Preference_Content_Type_Multimedia	*f*(x)	Content_Type
Preference_Content_Style_Long	*f*(x)	Content_Style
Preference_Content_Style_Abreviated	*f*(x)	Content_Style
Preference_Content_Style_Symbolic	*f*(x)	Content_Style
Input_pointing	*f*(-1 (1 – x))	Interaction_effort_pointing
Input_keys	*f*(-1 (1 – x))	Interaction_effort_keystroke

Tabel 22: Lijst van mappingfuncties.

Context parameter (omgeving)	**Mapping operator**	**Context parameter (user interface)**
Visual_screenheight	>	Visual_size_height
Visual_screenwidth	>	Visual_size_width
Visual_screencolordepth	>	Visual_finedetail
Visual_screencolordepth	=	Output_required_color
Multimedia_image	=	Output_required_multimedia
Multimedia_image	=	Content_type
Input_keys	>	Interaction_effort_keystroke
Input_pointing	>	Interaction_effort_pointing
Visual_detail	>	Visual_finedetail
Motoric_finesse	>	Interaction_effort_pointing

Tabel 23: Lijst van mappingoperators.

Opmerking: Er kan zowel een verband als een beperking bestaan tussen dezelfde paramaters, zoals Input_pointing en Interaction_effort_pointing. Dit wil zeggen dat de eerste een invloed heeft op de tweede maar dat hier eveneens een voorwaarde aan verbonden is (en in dit geval groter moet zijn).

Om transformaties van niveau 2 te ondersteunen worden ook deze mappings opgesteld:

Context parameter	**Mapping operator**	**Property**
Visual_minimumfontsize	<	fontsize
Visual_minimumheightwidth	<	height (size)
Visual_minimumheightwidth	<	width (size)
Preference_visual_color_foreground	=	foreground
Preference_visual_color_background	=	background
Visual_Colordeficiency	!=	color
Visual_Colordeficiency	!=	foreground

Visual_Colordeficiency	!=	background

Tabel 24: Lijst van Mappingoperators tussen contextattributen en 'class' attributen.

Property	**Mapping operator**	**Property**
background	!=	foreground
foreground	!=	background

Tabel 25: Lijst van mappingoperators tussen 'class' attributen onderling.

6.3.2 Beslissingsprocedures en regels

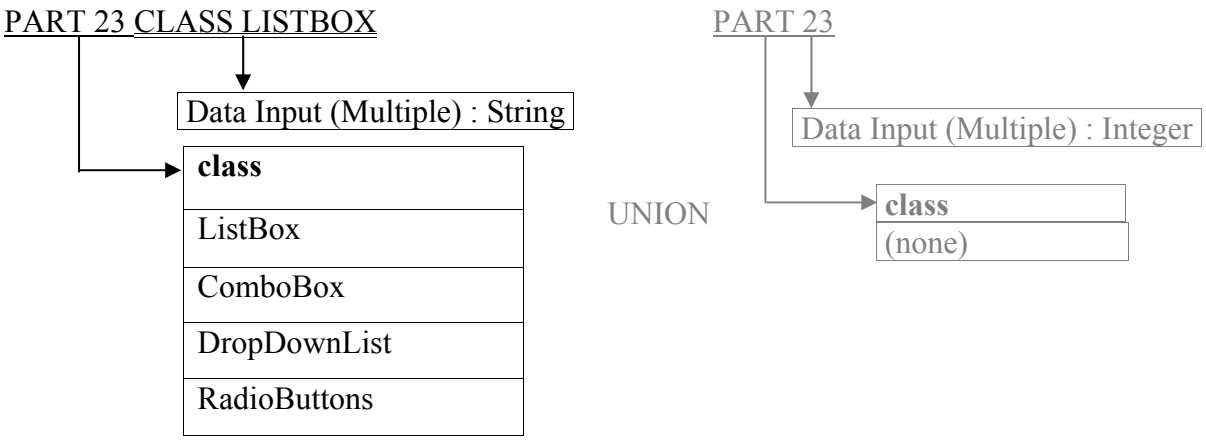
Met de gegevens in het vorige deel wordt er via een beslissingsmodel de juiste transformatieoptie gekozen. Dit gebeurt gedeeltelijk zoals het principe in [63]. In combinatie met de metadatalaag (zie 6.1.3) bereiken we hiermee hetzelfde resultaat als het Digital Item Adaptation principe van MPEG-21 (zie 3.2.3.1). Het is niet zo voor de hand liggend om een 'goede' keuze te maken wanneer er meerdere factoren inwerken op een bepaald onderdeel. Er dienen 2 evaluaties uitgevoerd te worden:

- is de bepaalde configuratie van een user interface onderdeel technisch geschikt?
- welke van de technisch mogelijke oplossingen is het meest geschikt?

Voor elk onderdeel dat adapteerbaar is, wordt de metadata opgevraagd aan de hand van het 'id' attribuut van het UIML element.

Indien het gaat om een samengesteld of elementair widget wordt de metadata van elk alternatief widget dat dezelfde taak ondersteund opgevraagd.

- Het UIML element wordt ingelezen.
  <part id="23" class="ListBox">
- De metadata wordt opgevraagd van de klasse 'ListBox', en van 'part 23' indien beschikbaar.

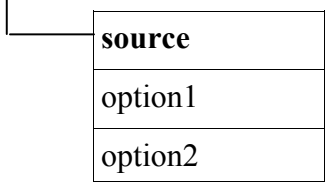


Figuur 42: Stap 1 van beslissingsprocedure voor widget selecties.

Voor polymorfe samengestelde delen waarvoor meerdere templates beschikbaar zijn, moet worden bepaald welke template plastisch genoeg is voor de huidige context. Indien dit er meer dan 1 is, dient hiervan de meest geschikte te worden gekozen.

- Er wordt een abstract samengesteld element ingelezen.
  <part id="51" class="Container" source="...">

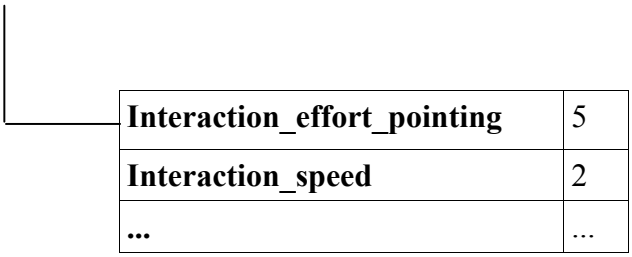
- De metadata wordt opgevraagd van het container part.
  PART 51
  Het source attribuut kan 2 waarden aannemen:'option1' of 'option2'.



Figuur 43: Stap 1 van beslissingsprocedure voor polymorfische delen.

De volgende stap is hetzelfde voor klasse transformaties en polymorfische samengestelde delen. Voor elke mogelijke optie wordt de metadata opgevraagd en aan de hand van de contextwaarden een score berekend. Dit wordt gedaan door de attribuutwaarden van de gebruikersomgeving te laten inwerken op de attribuutwaarden van elke alternatieve optie.

CLASS RADIOBUTTONS

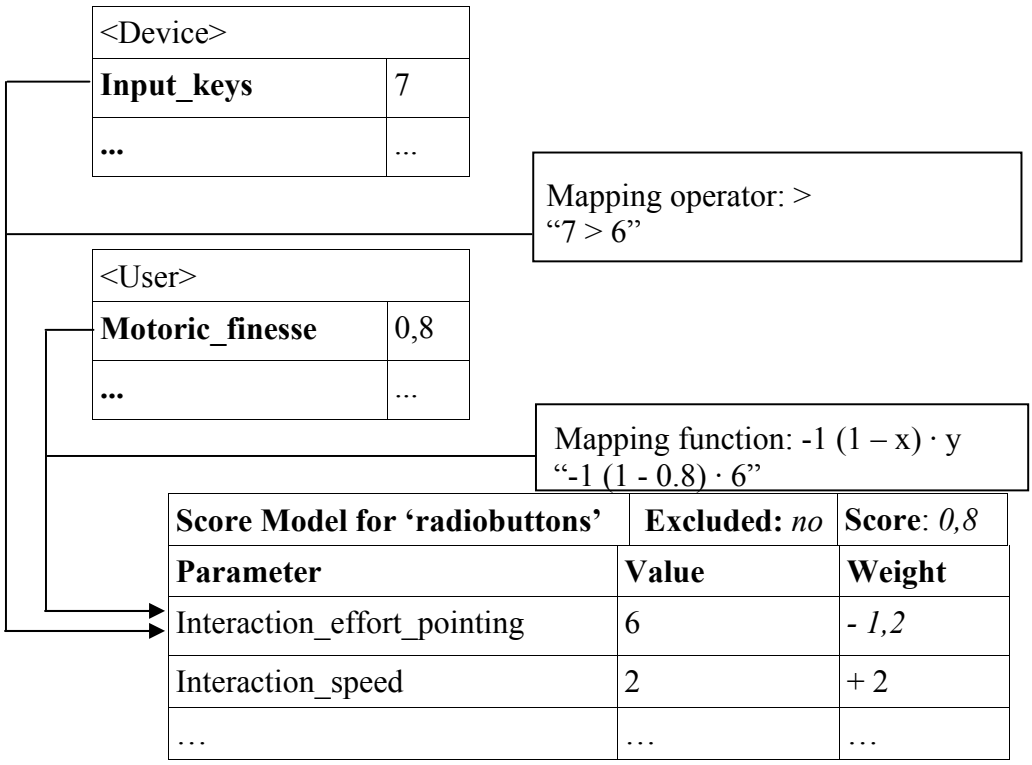


Figuur 44: Stap 2 van beslissingsprocedure.

Dit wordt eveneens gedaan voor de klassen 'ListBox', 'ComboBox', 'DropDownList' en bij het voorbeeld van het polymorfische deel voor de templates 'option1' en 'option2'.

De attributen van de gebruiker en het apparaat die in verband staan met de attributen van het kandidaat onderdeel, worden gebruikt om te bepalen in welke mate elk aspect zal doorwegen in de eindscore. Voor optimale performance wordt best eerst de informatie over het eindapparaat verwerkt en dan pas de natuurlijke informatie over de gebruiker, omdat apparaatattributen meestal beperkingen opleggen (constraints) en daarom primair zijn. Deze hebben voorrang en kunnen er voor zorgen dat sommige opties al uitgeschakeld (geschrapt) worden omdat ze niet geschikt zijn voor het apparaat en dus niet meer gebruikt hoeven te worden bij de verwerking van gebruikersinformatie.

- Voor elk context aspect (parameter) kunnen er mappings bestaan op één of meerdere aspecten (parameters) van user interface onderdelen (parts, templates, content, style).
- Voor de verwerking wordt er een intern datamodel gebruikt.



Figuur 45: Stap 3 in beslissingsprocedure: opstelling beslissingsmodel.

Het "Input_keys" attribuut wordt gemapt op het "Interaction_effort_pointing" attribuut van de klasse 'RadioButtons'. De operator ">" bepaalt hoe de waarde van het context attribuut (in dit geval 7) moet vergeleken worden met de waarde van het attribuut van de kandidaatklasse. Het resultaat van de vergelijking bepaalt of de optie uitgesloten wordt of niet. In dit geval is de optie geschikt (tot nu toe) omdat 7 > 6.

Indien de template/class/part (hier 'RadioButtons') nog niet uitgesloten is, wordt de score berekend. De property "Motoric_finesse" heeft een mapping met "Interaction_effort_pointing". De waarde van "Motoric_finesse" bepaalt hoe sterk de waarde van "Interaction_effort_pointing" zal doorwegen in de score. De waarde -0,2 (uitkomst van -1 (1 – 0,8)) wordt vermenigvuldigd met 6 waardoor we op een gewicht van -1,2 komen.

Er is geen enkele eigenschap van de context die gemapt wordt met "Interaction_speed" waardoor het voorgedefinieerde gewicht +2 wordt gebruikt (in de meeste andere gevallen is het standaardgewicht 0). Indien de gebruiker en/of het platform bijvoorbeeld geen of evenveel negatieve invloed heeft op de score van de verschillende opties, zal de optie met de hoogste waarde van "Interaction_speed" de hoogste score halen. Hierdoor worden de efficiëntste widgets gebruikt indien er geen beperkingen worden opgelegd en zullen er geen onnodig compacte en minder efficiënte widgets worden gebruikt (ook al zijn deze geschikt). Indien het scherm voldoende groot is en er verder geen relevante beperkingen zijn zal bijvoorbeeld de optie met sliders gebruikt worden en niet die met de spinners.
Door alle waarden op te tellen komt men aan de score. In dit geval 2 + (-1,2) = 0,8.

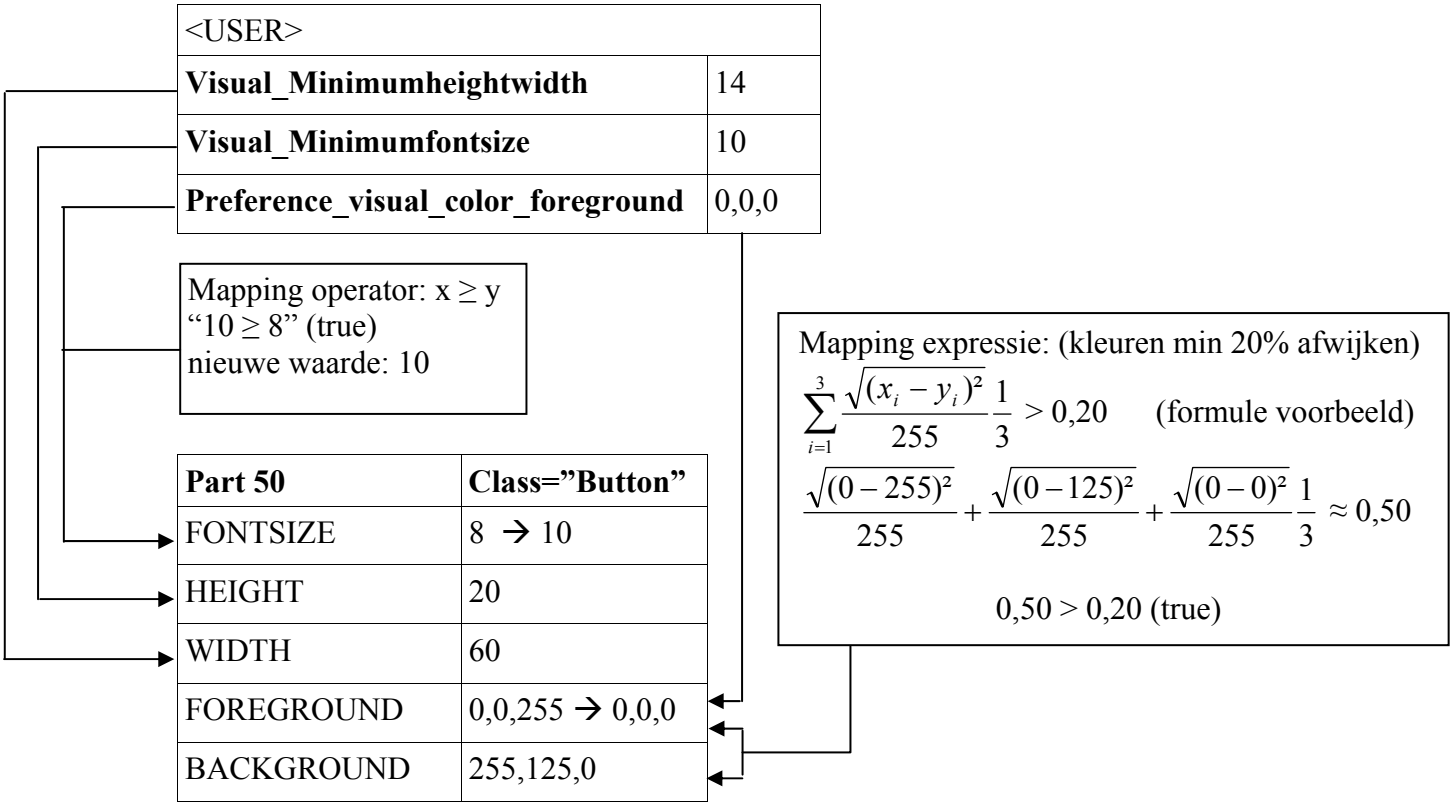
Het alternatief met de hoogste score wordt gekozen. In het geval er een template geïmporteerd wordt, wordt de naam van de template met de hoogste score ingevuld in het 'source' attribuut.
bv. <part id="51" source="**optie1**" how="replace"> …

In het geval van een widget wordt de transformatie uitgevoerd van het huidige naar het nieuwe gekozen widget.
bv. <part id="23" class="**ComboBox**"> …

Op het scoremodel kunnen ook checks worden uitgevoerd (door constraintmappings), waardoor de optie als “Excluded” kan aangeduid worden. Er kan een tweede mapping voor “Motoric_finesse” bestaan die de waarde met 10 vermenigvuldigt (10x > y) en controleert of deze waarde groter is dan de waarde van de parameter waarop deze gemapt is. In dit geval zou dit zijn: 0,8 x 10 = 8 > 6. Deze vergelijking is waar. Als de gebruiker een zware motorische handicap heeft (Motoric_finesse = 0,1) zou de waarde vermenigvuldigd met 10 (=1) kleiner zijn dan 6 en wordt de optie als ongeschikt beschouwd, ook al zou deze de hoogste score hebben (dankzij andere parameters). Als alle opties op deze manier als ongeschikt zijn aangeduid zou deze met de hoogste score kunnen gebruikt worden. Uiteraard is dit niet de beste oplossing en bestaan er andere methodes om toch de meest geschikte optie te kiezen (vb. kleinst overschrijdende waarde van dergelijke parameter).

Een andere methode is de uitkomstwaarden van de negatieve mappings in het kwadraat nemen en te vermenigvuldigen met -1, waardoor sterke afwijkingen sterker gaan doorwegen dan meerdere kleine.

Voor adaptaties op niveau 2 wordt een gelijkaardig model gebruikt. Hierin wordt de user interface geconfigureerd door de ‘property’-waarden van de elementen aan te passen (zie 6.1.2).



Figuur 46: Adaptatiebeslissingen op niveau 2.

Bij deze adaptatievorm wordt de waarde van het attribuut ingevuld indien de expressie waar is.
Er wordt gecheckt naar de waarde van het property “fontsize”, deze is kleiner dan de minimum lettertypegrootte en zal aangepast worden naar 10.
De meta-property “Visual_Minimumheightwidth” heeft 2 mappings (op de properties height en width). Deze voldoen aan de vereisten en worden niet aangepast.
Indien de ‘property’ niet vastgelegd is, wordt deze toegevoegd.

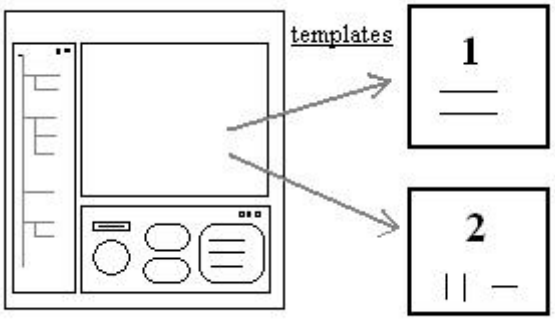
6.3.3 Adaptatieproces

In het vorige hoofdstuk (zie 5.4) hebben we vastgelegd welk type adaptatie in welke fase uitgevoerd wordt. Hier wordt het adaptatieverloop besproken waarbij de verschillende uitgewerkte concepten worden geplaatst in het adaptatieproces van de toepassing.

De gehele adaptatie gebeurt in 3 stappen.

Stap 1

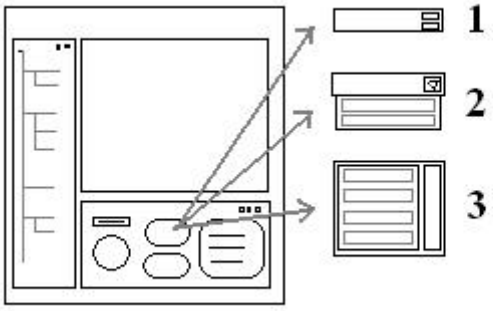
Eerst worden adaptaties op het hoogste niveau (5) uitgevoerd waarbij polymorfische delen van de user interface worden ingesteld. Ook bepaalde configuraties van niveau 2 gebeuren al in deze fase (stijl- en content-templates) om efficiëntie redenen. Dit gebeurt tijdens het inlezen van het UIML document, dus tijdens de opbouw van de UIML tree.
Concreet worden hier de templates toegewezen aan samengestelde delen.



Figuur 47 : toewijzing van templates (geïmporteerd door polymorfische delen van niveau 5).

Stap 2

Vervolgens worden de transformaties van niveau 4 en 3 uitgevoerd op samengestelde en elementaire delen (inclusief de inhoud van de toegewezen templates). Dit gebeurt na het inlezen van het UIML document en voor de mapping op de concrete widgets (via de vocabulary).



Figuur 48: transformatie van widgets (niveau 3 en 4).

Stap 3

In de laatste stap worden de user interface delen verder geconfigureerd (niveau 2). Hierbij worden de evaluaties (constraint checks) van de 'property' mappings uitgevoerd (zie 6.3.1) om de configuratie te valideren.

De user interface specificatie is volledig geadapteerd naar een andere specificatie. Dit kan worden

vergeleken met deel 7 (DIA) van het MPEG-21 principe (zie 3.2.3.1). Hierna kan UIML.NET de aangepaste user interface specificatie via de vocabulary mappen op concrete widgets en de user interface realiseren.

6.3.4 Voorbeeld

We beschouwen een user interface om domotica faciliteiten aan te sturen.

Deze wordt aangeroepen door 2 personen: Opa en zijn kleindochter.

De kleindochter beschikt over het volgende profiel:

Motoric_finesse	1.0
Semantic_understanding	0.9
Preference_interaction_keyinput	0.1
Preference_interaction_oneclick	1.0
Preference_color_background	0,128,0
Preference_color_back_frame	0,128,0
Preference_color_back_control	0,128,0
Preference_color_fore_control	255,255,255
Preference_color_text	255,255,255
Preference_content_symbolic	1.0

Tabel 26: Voorbeeldprofiel 1.

Uit het profiel kunnen we afleiden dat de kleindochter veel ervaring heeft met user interfaces, liefst met klik interacties werkt, niet graag het toetsenbord gebruikt en symbolen boven tekstuele notaties verkiest.

Opa beschikt over het volgende profiel:

Motoric_finesse	0.7
Semantic_understanding	0.4
Preference_interaction_keyinput	1.0
Preference_interaction_drag	0.1
Preference_color_background	200,255,255
Preference_color_back_frame	255,255,200
Preference_color_back_control	200,0,0
Preference_color_fore_control	255,255,255
Preference_color_text	200,0,0
Preference_content_long	0.8

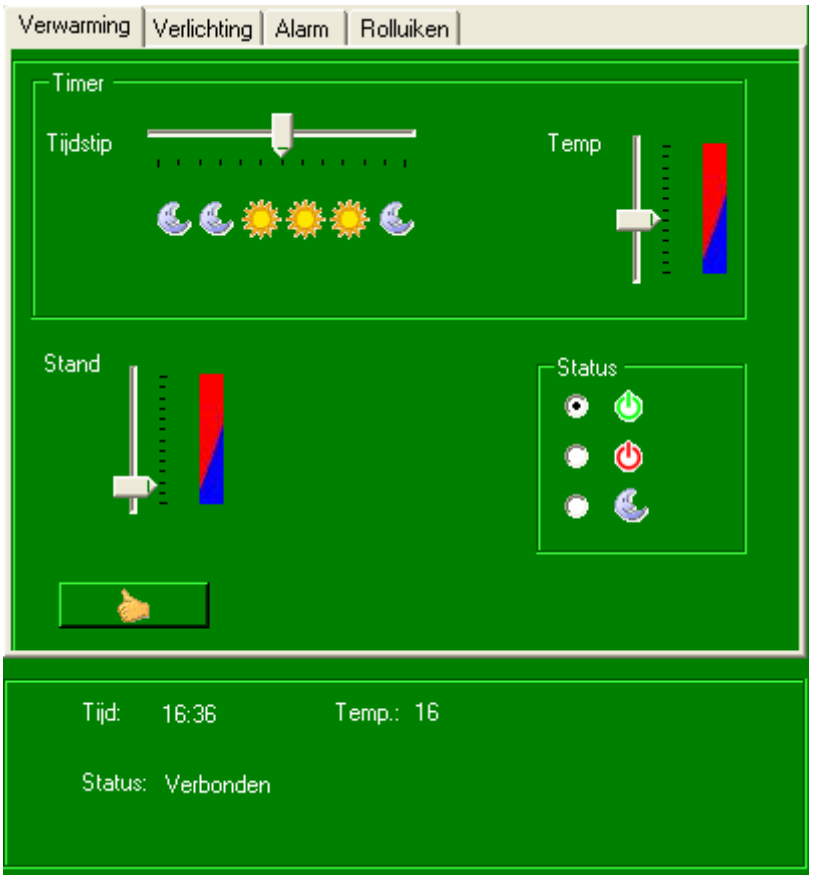
Tabel 27: Voorbeeldprofiel 2.

Uit het profiel kunnen we afleiden dat opa waarschijnlijk bevende handen heeft, een hekel heeft aan sleepbewegingen, weinig ervaring heeft met user interfaces, liefst werkt met het toetsenbord en tekstuele weergave verkiest.

De user interface aangeroepen door opa:

Figuur 49: Resultaat van adaptatie (1).

De user interface aangeroepen door de kleindochter:



Figuur 50: Resultaat van adaptatie (2).

7 Discussie

7.1 Evaluatie

Met de uitgewerkte toepassingsuitbreiding op het UIML.NET framework kunnen we de user interface beschrijvingen verrijken met informatie op andere niveaus (zie 5.5, 5.6 en 6.1.3). Hierdoor laten we de onderdelen van de user interface 'spreken' en weerspiegelen ze semantische informatie over zichzelf. Door deze informatie te mappen op attributen van metaobjecten die de buitenwereld voorstellen (zie 6.2.1), kunnen we het systeem autonoom de noden van de gebruiker laten opmaken op basis van informatie die een menselijke designer in acht zou nemen om deze vereisten vast te stellen (zie 6.3.1). De reactie op deze opgestelde eisen wordt ondersteund door de semantische effecten van mogelijke transformaties kenbaar te maken aan het systeem, die deze gegevens toetst aan de eisen van de context (zie 6.3.2). Een doeltreffende adaptatiebeslissing wordt genomen door te berekenen in welke mate een bepaalde transformatie een positief of negatief effect zou hebben op de bruikbaarheid.

Als we de resultaten vergelijken met deze van andere onderzoeken over adaptatie ondersteuning van user interfaces, zouden we kunnen stellen dat met de gerealiseerde architectuur, het mogelijk zou zijn in elk domein adaptatie te kunnen ondersteunen, terwijl dit in de meeste benaderingen beperkt is tot adaptatie in een enkel domein zoals multi-device ondersteuning, platformonafhankelijkheid, gebruikersprofiel op basis van ervaring, enz. De ontwikkelaar staat vrij om te kiezen welke contextfactoren hij of zij wil opnemen voor de adaptatieafstelling (zie 5.3). De applicatie maakt zich uniek door zijn mogelijkheid om user interfaces op een relatief concreet niveau te kunnen samenstellen en toch doeltreffende adaptaties op een hoger niveau te bereiken. De eenvoudige werkwijze in het gebruik van de architectuur onderscheidt zich van gelijkaardige aanpakmethodes die een strikte scheiding van abstractielagen opleggen, waarbij meerdere ingewikkeldere aspecten in acht moeten genomen worden. De keerzijde van deze benadering zit in het feit dat deze begrensd is tot op een middelhoog abstractieniveau waarop intelligente adaptatie kan worden toegepast (zie 5.3). In de praktijk is dit niet zo een grote beperking doordat adaptaties op hogere niveaus, minder waarschijnlijk nodig zijn bij gematigde contextveranderingen. Toch zijn adaptaties boven deze abstractiegrens zonder beperkingen mogelijk door het gebruik van voorgedefinieerde adaptatie-uitwerkingen waarbij de plasticiteit voor de huidige context geëvalueerd kan worden (zie niveau 5 in 6.1.2).

Deze technieken zijn net zoals de UIML specificatie generisch opgesteld en worden als metamodel gebruikt voor de specificatie van contextstrategieën. Ze steunen hierbij op de declaratief opgestelde elementen, geïnspireerd op het DIA-principe van MPEG-21 (zie 3.2.3.1), zoals templates, gescheiden stijl elementen, de hiërarchische 'structure' sectie, vocabularies, attributen enz. Mede hierdoor wordt de graad en complexheid van de adaptatie door de auteur van een UIML document zelf bepaald, een drempelvereiste die bij de meeste architecturen acuut ontbreekt.

7.2 Future Work

De uitgewerkte architectuur bevat de fundamenten om in alle domeinen adaptatie op eender welk niveau te implementeren. Een volgende stap in deze ontwikkelingen zou kunnen bestaan uit onderzoeken naar methoden om de contextgegevens of metadata van de user interface automatisch te verkrijgen.

7.2.1 Normaliseren van subjectieve attribuutwaarden

Voor contextattributen worden waarden opgesteld volgens subjectieve beoordelingen van de auteur van de user interface. Men kan tot nu toe niet bepalen wat een contextattribuut (vb. Visual_finedetail) met de waarde 6 voorstelt. Indien men een contextmodel wil hergebruiken, is het

gebruik van een geijkt schaalmodel gewenst. Hiervoor zou men gestandaardiseerde indicatietesten kunnen opstellen waarmee men rationeel een waarde kan toekennen aan een bepaald attribuut (met behulp van Fitt's Law enz.) en deze gebruiken voor het berekenen van de optimale user interface configuratie [63]. Men kan dit vergelijken met de rationele formules die gebruikt worden om bijvoorbeeld de risicofactor van een beleggingsfonds te bepalen.

Gebruikers kennen meestal zelf hun karakteristieken of voorkeuren niet. Voor gebruikersattributen in het bijzondere, kan men hiervoor gestandaardiseerde likert scales en performance testen opstellen voor een bepaald contextmodel, om automatisch te bepalen welke waarde van toepassing is. Ook de efficiëntie of voorkeur voor bepaalde widgets of interactiemethodes kunnen hiermee 'gemeten' worden. Indien een user interface toch is opgesteld volgens een afwijkend schaalmodel, zou men bijvoorbeeld een adaptatieparameter kunnen instellen voor het gebruikersprofiel om een bepaald contextattribuut met een factor 1,5 te vermenigvuldigen, waardoor het plasticiteitdomein van een optie 'uitgerekt' of 'ingekrompen' wordt en een bepaalde optie sneller gekozen zal worden of omgekeerd.

7.2.2 Methodes om de plasticiteitwaarde te berekenen van high level patronen

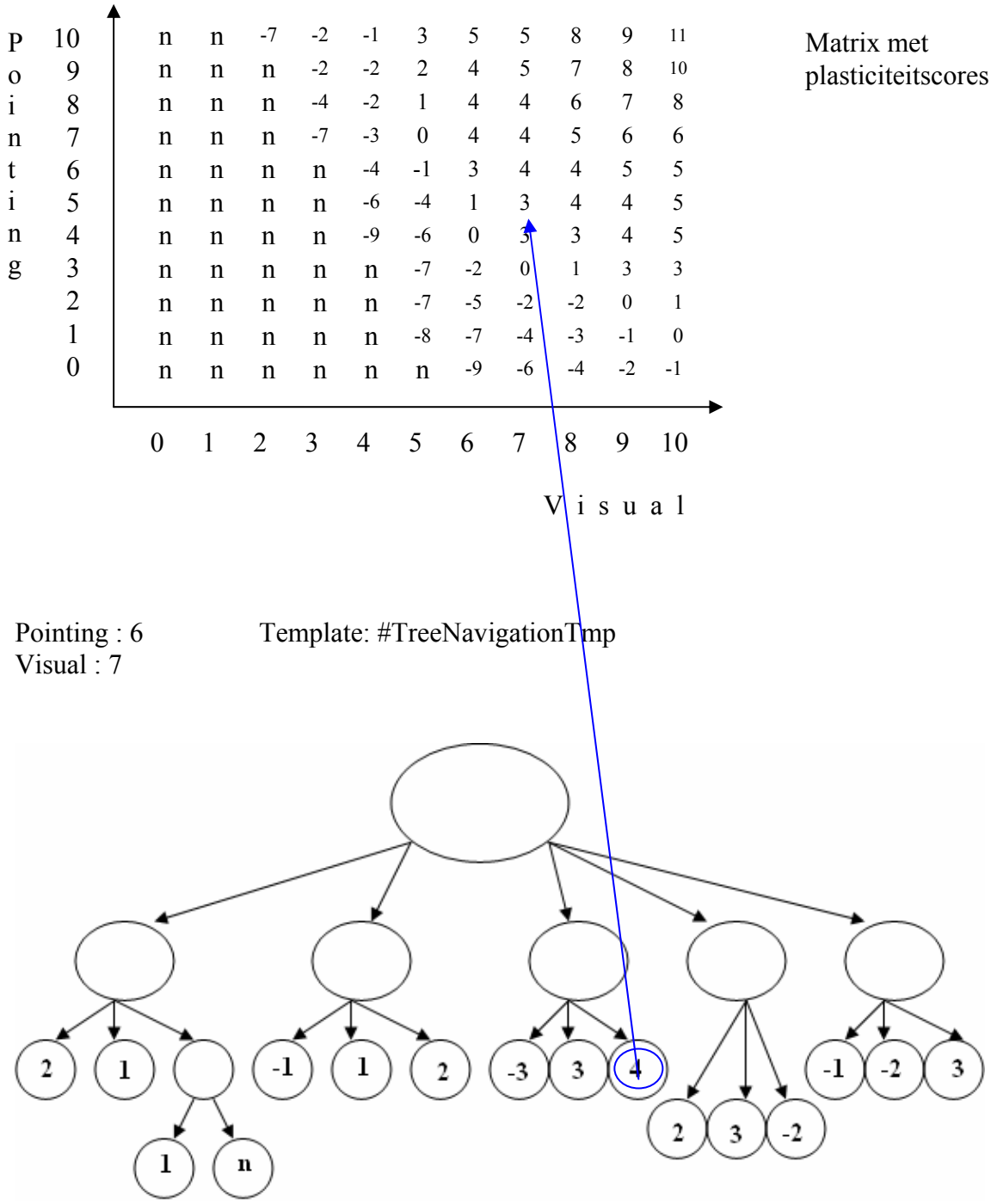
De huidige toepassing gebruikt naast transformaties voor low en medium level patronen ook (volledig of gedeeltelijk) uitgewerkte transformaties voor high level patronen. Doordat op deze voorgedefinieerde uitwerkingen in een latere fase nog adaptaties op low en medium abstractieniveau kunnen uitgevoerd worden, is het moeilijk om de semantische attributen hiervan gedetailleerd te definiëren, en te berekenen of het patroon plastisch genoeg is voor de huidige context.

Een oplossing hiervoor zou een branch-and-bound methode [63] kunnen zijn die we gebruiken om van de verschillende implementatiemogelijkheden van een zodanig onderdeel, de metadata van diegene met de hoogste score gebruiken. Hierbij wordt er voor elke combinatie transformaties een intern model opgebouwd van dit onderdeel van de user interface, zonder het effectief te compileren. Van elke configuratie wordt vervolgens de score berekend. Zo kan van een alternatief onderdeel dat een high level patroon bevat, waar enkel implementaties met afbeeldingen bruikbaar zijn, automatisch worden bepaald dat deze niet bruikbaar is als het toestel geen afbeeldingen ondersteunt. En dit terwijl deze meta-informatie niet expliciet aan het onderdeel zelf verbonden is.

Op zich is dit een exhaustieve en verkwistende methode die het user interface generatieproces onnodig kan vertragen, zeker bij grotere toepassingen met uitgebreide adaptatiemogelijkheden. Dit is op zich geen probleem doordat deze data niet tijdens het transformatieproces hoeft berekend te worden. Ze kan evengoed of zelfs beter berekend worden in de ontwerpfase van de user interface. Doordat we in deze fase nog niet over de contextinformatie beschikken en deze uiteraard kan veranderen, dienen we deze werkwijze licht aan te passen. In plaats van de score voor één specifieke context te berekenen gaan we het plasticiteitdomein van elk high level patroon bevattend onderdeel berekenen. In UIML is dit bijna altijd een template. Hiervoor kan de volgende werkwijze gebruikt worden.

Men stelt een matrix op met evenveel dimensies als de natuurlijke gebruikerscontext (= aantal contextattributen). In het voorbeeld gebruiken we voor de duidelijkheid een gebruikerscontext met 2 dimensies (in de praktijk zullen dit er meestal veel meer zijn).

Voor elke mogelijke situatie die bestaat uit een combinatie van attribuutwaarden (= elk element van de matrix) wordt zoals eerder vermeld met een branch-and-bounce methode gewerkt, waarbij voor elke mogelijke configuratie (als gevolg van low en medium level adaptaties) de score wordt berekend voor die context. De hoogste score van alle uitwerkingen wordt ingevuld in het matrixelement van deze mogelijke situatie.



Figuur 51: Berekening van plasticiteitmatrices voor templates.

Dit wordt voor elke mogelijke omgevingssituatie berekend en dus voor elk element van de meerdimensionale matrix.

Voor elke template met high level patronen wordt deze matrix als metadata meegegeven. Voor een bepaalde omgevingscontext dient het systeem enkel de overeenkomstige score in de matrix op te zoeken om voor de template te gebruiken.

7.2.3 Generatie van mappings en adaptatieregels door artificiële intelligentietechnieken

De kern van de adaptatiemechanismen verwerken geraffineerde fundamentele attributen van de context zoals 'pointing effort', 'visual meaningness', 'preference' enz. Deze gegevens kunnen rechtstreeks gelinkt worden aan andere fundamentele eigenschappen van user interface elementen. Wanneer deze verbanden en mappingfuncties correct gedefinieerd worden zou dit na het adaptatieproces tot de best afgestemde en efficiëntst mogelijke user interface moeten leiden [63]. Deze gegevens moeten eenmalig worden ingevoerd maar dienen met de nodige zorgvuldigheid worden vastgelegd om gewenste resultaten te hebben. Er zou verder onderzoek kunnen gebeuren naar manieren om deze gegevens te verkrijgen, dit zijn langs de ene kant contextattributen en mappingsregels langs de andere kant.

Het modelleren van kennis is reeds aan bod gekomen. Er is geconcludeerd dat er nog veel onderzoek kan gebeuren naar het modelleren van kennis over usability van user interfaces. Met het OWL (Web Ontology Language) framework (zie 3.2.2.2) zou men door het modelleren van deze kennis, de elementaire attributen van de natuurlijke omgeving automatisch kunnen afleiden. Deze informatie zit trouwens meestal impliciet in de beschikbare omgevingsinformatie verwerkt, zoals de gemeten gegevens van verschillende sensoren, gedragsobservatie, gebruikersprofiel enz. Wanneer de gebruiker in de wagen zit zou dit systeem op een intelligente manier de waarde van de motorieke capaciteit van de gebruiker kunnen aanpassen, zonder dat er een apart gebruikersprofiel is opgesteld voor deze specifieke situatie. Het adaptatiesysteem zal hiermee rekening houden zoals nu het geval is en ervoor zorgen dat de gebruiker geen precieze beweging hoeft uit te voeren. Ook de voorkeuren kunnen op deze manier worden afgeleid door observatie van eerdere handelingen. Dit kan uit meerdere gegevens worden afgeleid zoals bv. de combinatie van een mobiel platform en een donkere omgeving kan wijzen op het aanpassen van visuele parameters. Het huidige systeem zal hierdoor minder felle kleuren gebruiken om de gebruiker niet te 'verblinden'. Bij een uitgebreide ontologische modellering kan men zelfs meerdere soorten profielen door elkaar gebruiken en hoeft er geen standaardprofiel te worden opgelegd.

Als we nog een niveau hoger gaan, kan men met een ontologische modellering de mappingregels automatisch laten opstellen. Hierdoor hoeven we niet meer expliciet aan het systeem duidelijk te maken dat een listbox met meer dan 10 items niet aangeraden is voor visualisatie op kleine schermoppervlakken, en een 'slider' widget moeilijk te gebruiken is bij afwezigheid van een pointing apparaat. Concreet wil dit niets meer zeggen dat we zelf niet meer de aspecten *screenzize* en *footprint*; en *motoric finesse* en *pointing effort* moeten linken met een functie. Op zich heeft dit in de praktijk weinig voordelen omdat deze regels slechts eenmalig gedefinieerd moeten worden. Toch betekent dit op korte termijn een enorme vooruitgang. Naarmate er een nieuwe generatie invoerapparaten of user interface widgets gebruikt worden, gaan er andere aspecten meespelen die hun bruikbaarheid bepalen. Door enkel de ontologie in het framework uit te breiden, hoeft men niet iedere keer opnieuw deze mappingsregels in het contextmodel aan te passen om de user interfaces die hiervan gebruik maken compatibel te maken met deze nieuwe gebruikersplatformen. Het model zal hiermee veel langer meegaan in de evolutie van user interfaces en gebruikt kunnen worden voor applicaties in verschillende domeinen waarbij human-computer interaction een steeds belangrijkere rol aanneemt. Afstelling aan mogelijk nog onbestaande applicaties, zoals bv. industriële toepassingen, ruimtevaart, entertainment, domotica, met elk hun eigen type user interface kan gerealiseerd worden zonder van voor af aan een nieuw generisch contextmodel op te stellen of fundamentele aanpassingen te doen aan het framework.

7.3 Algemene conclusie

In het user interface ontwikkelingsproces worden tal van beslissingen genomen waarbij sociale en abstracte factoren in rekeningen worden gebracht. Deze gebruikersgerichte methoden steunen op het menselijke vermogen om dit soort informatie te verwerken. Het is een grote uitdaging deze taak onder de verantwoordelijkheid van een computersysteem te plaatsen. Toch wordt dit steeds meer een noodzaak door het groeiende aantal heterogene toestellen, gebruikerstypen, en omstandigheden waarin een applicatie gebruikt wordt. Het handmatig aanpassen van de user interface aan deze snel evoluerende begrippen, zal hierdoor steeds kostelijker en moeilijker tot bijna onmogelijk worden.

De tot nu toe gerealiseerde architecturen uit andere onderzoeksstudies passen methoden toe om onder invloed van de context, vanuit abstracte objecten de geschikte concrete objecten af te leiden (zie 2.2.3). Uit de resultaten van dit onderzoek is vastgesteld dat adaptatie op concrete specificaties van user interfaces mogelijk zijn door ze te verrijken met meta-informatie (zie 3.2). Deze zorgt ervoor dat uit concrete interactie objecten, abstracte objecten kunnen afgeleid worden die nodig zijn voor het adaptatieproces. Zo kan men door patroon extractie geschikte alternatieve patronen identificeren. Anderzijds is er gebleken dat er een grens ligt in het abstractieniveau van deze objecten. Als deze patronen op te hoog niveau liggen, is de kost voor een transformatie veel groter tegenover een nieuw user interface model af te leiden, vertrekkend van een hoger abstractieniveau. Uit de onderzoeksresultaten is gebleken dat de abstractiegrens voor intelligente adaptatie van concrete user interface beschrijvingen zoals UIML, niet veel lager ligt dan deze van abstracte gelaagde beschrijvingen en frameworks doordat geen van deze frameworks volautomatische mappings toepast op hoog niveau (zie 2.2).

Een geautomatiseerd adaptatieproces handelt met dezelfde kwesties als handmatige adaptaties waarbij contextherkenning, afleiding van vereisten en opstellen van een oplossing die aan de vereisten tegemoet komt, noodzakelijk aan bod komen. De vereiste benodigdheden voor de ondersteuning van intelligente adaptatie, zijn driedelig en bestaan uit transformatiemechanismen, contextmodellering en redeneringsprocedures (hoofdstuk 3). De manier waarop deze 3 ‘werktuigen’ tot stand komen worden sterk beïnvloed door het niveau en type van de adaptatie die gerealiseerd moet worden (hoofdstuk 4).

Transformatiemechanismen zijn slechts een onderdeel in een adaptatieproces en worden gestuurd op basis van eigenschappen van de context die met elkaar in verband worden gebracht (hoofdstuk 5). Het is hierbij belangrijk dat al deze contextinformatie beschikbaar is. In het domein van adaptatie beschouwen we context als alle entiteiten die meespelen in het adaptatieproces. Dit zijn de natuurlijke gebruikersomgeving (gebruiker, apparaat, omgeving), de user interface zelf en de beschikbare user interface toolkits of systeemfaciliteiten. Het modelleren van deze entiteiten maakt het mogelijk deze te benaderen vanuit het systeem (zie 5.5). Door de klassen van een UIML vocabulary met de juiste metadata te verrijken kunnen doeltreffende adaptaties bereikt worden tot op middelhoog abstractieniveau. Concreet zijn dit transformaties op delen van de user interface die een enkelvoudige of veel voorkomende samengestelde taak ondersteunen. Adaptatieoplossingen op hoog niveau kunnen ook bereikt worden mits gebruik van een voorgedefinieerde transformatie of alternatieve opstelling van een deel van een specifieke user interface. Een laatste belangrijke stap is het systeem de juiste kennis te laten toepassen om vanuit deze informatie zinvolle adaptaties uit te voeren (zie 5.6). Deze kennis gaat uit van abstracte aspecten van de user interface en vereist dat deze ook op een abstracte manier wordt beschreven. Dit was het moeilijkste deel van deze onderzoeksstudie omdat er abstracte kennis toegepast moet worden op exacte maar ook semantische en subjectieve gegevens. Deze kennis en gegevens programmatisch op een universele manier toepassen is een complexe materie.

Om de opgedane kennis en modellen op grote schaal in de praktijk toe te passen, zullen er andere facetten moeten bestudeerd worden die kruisen met andere thema’s. Zo zal er geëvalueerd worden hoe gebruikers reageren op adaptaties waar ze zelf geen controle over hebben. Begrippen die hier aan bod komen zijn privacy (gebruiker voelt zich geobserveerd) [78], voorspelbaarheid en

kwaliteit van adapteerbare user interfaces [79]. Het is op dit moment nog onbekend of de ideale technieken voor adaptatie ooit benaderd zullen worden, waarbij zonder user interface model of specificatie de user interface volledig automatisch gegenereerd en geconfigureerd wordt. Indien de huidige trends zich in dit tempo verder zetten, lijkt dit op het eerste zicht wel zo. Toch zal deze evolutie waarschijnlijk vroeg of laat zijn hoogtepunt bereiken omdat user interfaces op een of andere manier altijd applicatieafhankelijk zijn en machines moeilijk het redeneringsvermogen van de mens kunnen evenaren. Het is uiterst moeilijk om kennis uit niet-exacte wetenschapsdomeinen waarbij onzekerheden aan te pas komen te modelleren [80] en deze door een computersysteem te laten gebruiken om beslissingen te nemen [81] . Eén ding staat vast: adaptatie zal in de toekomst een vast begrip worden in de ontwikkeling van user interfaces.

Referenties

[1] Website: http://www.thefreedictionary.com
[2] Korvemaker, B. & Greiner, R. The Trials and Tribulations of Building an Adaptive User Interface. Department of Computer Science, University of Alberta, Edmonton, Canada.
[3] Dieterich, H., Malinowski, U., Kühme, T. & Schneider-Hufschmidt, M. State of the Art in Adaptive User Interfaces.
[4] Menkhaus, G. & Pree, W. (2002). A Hybrid Approach to Adaptive User Interface Generation. Software Research Lab, Constance University, Germany.
[5] Norman, D. A. (1988). The Design of Everyday Things.
[6] Website: http://www.interaction-design.org/encyclopedia/personas.html
[7] Calvary, G., Coutaz, J. & Thevenin, D. Embedding Plasticity in the Development Process of Interactive Systems. In 6th ERCIM Workshop "User Interfaces for All".
[8] Dey, A. K. Understanding and Using Context.
[9] Calvary, J., Coutaz, J. & Thevenin, D. (2001). Supporting Context Changes for Plastic User Interfaces: A Process and a Mechanism.
[10] Abstract User Interface Representations: How Well Do They Support Universal Access?
[11] Subramanian, N. & Chung, L. (2002). Tool Support for Engineering Adaptability into Software Architecture: Proceedings of the International Workshop on Principles of Software Evolution, ACM Press.
[12] Subramanian, N. & Chung, L. (2001). Software Architecture Adaptability: An NFR Approach. To appear in the Proceedings of International Workshop on Principles of Software Evolution, Vienna.
[13] Abrams, M. (2000). Device-independent Authoring with UIML. In W3C Workshop on Web Device Independent Authoring, Bristol.
[14] Müller, A., Forbrig, P. & Cap, C. H. (2001). Model-Based User Interface Design Using Markup Concepts.
[15] Calvary, G., Coutaz, J., Dâassi, O., Balme, L. & Demeure, A. (2004). Towards a new generation of widgets for supporting software plasticity: the "comet".
[16] Website: www.ximl.org
[17] Puerta, A. & Eisenstein, J. (2002). XIML: A Common Representation for Interaction Data.
[18] Puerta, A. & Eisenstein, J. (2001). XIML: A Universal Language for User Interfaces.
[19] Crowle, S. & Hole, L. (2003). ISML: An Interface Specification Meta-Language.
[20] Vetro, A., Timmerer, C. & Devillers, S. (2003). ISO/IEC 21000-7 FDIS: MPEG-21 Digital Item Adaptation.
[21] Luyten, J., Vandervelpen, C. & Coninx, K. (2002). Adaptable User Interfaces in Component Based Development for Embedded Systems.
[22] Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In Ashlund, S., Mullet, K., Henderson, A., Hollnagel, E. & White, T. (1993).
[23] Puerta, A. & Eisenstein, J. (1999). Towards a General Computational Framework for Model-Based Interface Development Systems.
[24] Cogliati, A. User Interface Adaptation for Personal Content Management. In T-111.5550 Seminar on Multimedia.
[25] Stephanidis, C., Akoumianakis, D. & Savidis, A. Design Representations and Development Support for User Interface Adaptation. Institute of Computer Science Foundation for Research and Technology-Hellas.
[26] Website: http://www.w3schools.com/xml
[27] Website: http://xmlns.com/foaf/spec/
[28] RDF Primer. Website : http://www.w3.org/TR/rdf-primer
[29] The Semantic Web: 1-2-3. Website: http://www.disobey.com/detergent/2002/sw123
[30] Dublin Core Metadata Initiative. Website: http://dublincore.org/documents/dces

[31] Horrocks, I. (2002). DAML+OIL: A Description Logic for the Semantic Web. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering.
[32] Houben, G. J. (2007). WIS Data: Semantic Web, RDF(S).
[33] Website: http://jena.sourceforge.net
[34] Broekstra, J. & Kampman, A. (2004). RDF(S) Manipulation, Storage and Querying Using Sesame.
[35] Raptor RDF Parser Library. Website: http://librdf.org/raptor
[36] Tauberer, J. Semantic Web/RDF Library for C#.NET. Website: http://razor.occams.info/code/semweb
[37] Bormans, J. & Hill, K. (2002). MPEG-21 Overview v.5. Website: http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm
[38] Website: http://giove.cnuce.cnr.it/cameleon.html
[39] Website: http://giove.cnuce.cnr.it/teresa.html
[40] Mori, G., Paterno, F. & Santoro, C. (2004). Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. Institute of Science and Information Technology, National Research Council.
[41] Dey, A., Salber, D., Futakawa, M. & Abowd, G. (1999). An Architecture to Support Context-Aware Applications, Technical Report GIT-GVU-99-23, GVU.
[42] Rey, G. (2001). Le Contexte en Interaction Homme–Machine, Diplom d'Etude Approfondies (DEA). Université Joseph Fourier–Grenoble I, France.
[43] Calvary, G. (1998). Proactivité et Réactivité: De l'Assignation à la Complémentarité en Conception et Evaluation d'Interfaces Homme–Machine, PhD thesis. Université Joseph-Fourier-Grenoble I, France.
[44] Naganuma, T., Luther, M., Wagner, M., Tomioka, A., Fujii, K., Fukazawa, Y. & Kurakake, S. (2006). Task-Oriented Mobile Service Recommendation Enhanced by a Situational Reasoning Engine. The Semantic Web – ASWC, p.768-774.
[45] Luther, M., Mrohs, B., Wagner, M., Steglich, S. & Kellerer, W. (2005). Situational Reasoning: A Practical OWL Use Case; Autonomous Decentralized Systems. ISADS Proceedings Volume, Issue 4-8., p. 461-468
[46] Weithöner, T., Liebig, T., Luther, M., Böhm, S., Von Henke, F. & Noppens, O. (2007). Real-world Reasoning with OWL. In: 4th European Semantic Web Conference, Innsbruck, Austria.
[47] Buriano, L., Marchetti, M., Carmagnola, F., Cena, F., Gena, C. & Torre, I. (2006). The Role of Ontologies in Context-Aware Recommender Systems. Department of Computer Science, University of Turin..
[48] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L., Florins, M. & Vanderdonckt, J. (2002). Plasticity of User Interfaces: A Revised Reference Framework, Université Catholique de Louvain.
[49] Wiley, J. & Sons. (2001). A Pattern Approach to Interaction Design (PAID).
[50] Carmagnola, F., Cena, F., Gena, C. & Torre, I. (2005). A Multidimensional Approach for the Semantic Representation of Taxonomies and Rules in Adaptive Hypermedia Systems, Université di Torino, Italy.
[51] Website: http://www.hcipatterns.org
[52] Website: http://c2.com/cgi/wiki?DesignPatterns
[53] Luyten, K., Thys, K., Vermeulen, J. & Coninx, K. (2006). A Generic Approach for Multi-Device User Interface Rendering with UIML.
[54] Clerckx, T., Luyten, K. & Coninx, K. (2004). The Mapping Problem Back and Forth: Customizing Dynamic Models While Preserving Consistency. In: TAMODIA (third annual conference on task models and diagrams), november 15-16, Prague, Czech Republic. ACM Press, p. 33-42.

[55] Abrams, M., Helms, J., Schaefer, R. & Vanderdonckt, J. (2004). Quentin OASIS User Interface Markup Language (UIML) Technical Committee (TC): Discussion of HCI Design Patterns. In: Meeting Held via Teleconference Hosted by Harmonia, Inc., Limbourg.
[56] Luyten, K., Thys, K. & Coninx, K. (2005). Profile-aware Multi-device Interfaces: An MPEG-21-Based Approach for Accessible User Interfaces. In: Accessible Design in the Digital World Conference, Hasselt University Expertise Centre for Digital Media.
[57] Luyten, K., Vermeulen, J. & Coninx, K. (2006). Constraint Adaptability of Multi-device User Interfaces, Hasselt University - Transnational University Limburg - Expertise Centre for Digital Media - IBBT.
[58] White paper on MPEG-21 Digital Item Adaptation. (2006). Website: http://www.chiariglione.org/MPEG/technologies/mp21-dia/index.htm
[59] FOAF Project. Website: http://www.foaf-project.org
[60] Heckmann, D., Schwartz, T., Brandherm, B., Schmitz, M. & Von Wilamowitz-Moellendorff, M. (2005). GUMO: The General User Model Ontology.
[61] Website: www.ubisworld.org
[62] Martinez, J. (2004). MPEG-7, Overview. Website: http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm
[63] Gajos, K. & Weld, D. S. (2004). SUPPLE: Automatically Generating User Interfaces.
[64] Javahery, H., Seffah, A., Engelberg, D. & Sinnig, D. (1998). Migrating User Interfaces Across Platforms Using HCI Patterns.
[65] One Model, Many Interfaces. In: CADUI, 2002 (Fourth International Conference on Computer-Aided Design of User Interfaces).
[66] Paterno, F. (2002). Splitting Rules for Gracefull Degradation of User Interfaces. Valenciennes, France.
[67] Seffah, A., Javahery, H.,Wiley, J. & Sons. (2004). Multiple User Interfaces: Cross-platform Applications and Context-aware Interfaces. New York.
[68] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. & Vanderdonckt, J. (2003). A Unifying Reference Framework for Multi-target User Interfaces. Interacting with Computers, 15, p. 289-308.
[69] Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley, Reading, MA, USA.
[70] Botterweck, G. A Model-driven Approach to the Engineering of Multiple User Interfaces. Institute for IS Research, University of Koblenz-Landau, Germany.
[71] Bézivin, J., Breton, E., Dupé, G. & Valduriez, P. (2003). The ATL Transformation-based Model Management Framework.
[72] Constantine, L. L. (2003). Canonical Abstract Prototypes for Abstract Visual and Interaction Design.
[73] Subramanian, N. & Chung, L. Richardson. Adaptable User Interface Generation.
[74] UIML.NET. Website: http://research.edm.uhasselt.be/~kris/projects/uiml.net
[75] Berben, I. (2006). Implementatie van de UIML Standaard. In: UIML.NET.
[76] Souchon, N. & Vanderdockt, J. A Review of XML-compliant User Interface Description Languages. Université Catholoque de Louvain, Institut d'Administration et de Gestion, Belgium.
[77] Vermeulen, J. (2005). Widget set independent layout management for UIML. Master's thesis Hasselt University.
[78] Jettmar, E. & Nass, C. (2002). Adaptive Testing: Effects on User Performance.
[79] Gajos, K. Z., Everitt, K., Tan, D. S., Czerwinski, M. & Weld, D. S. (2007). Predictability and Accuracy in Adaptive User Interfaces.
[80] Bonissone, P.P. & Decker, K.S. (1986). Selecting uncertainty calculi and granularity: an experiment in trading-off precision and complexity. In Uncertainty in Artificial Intelligence, p. 217-247.

[81] Smithson, M. (2006). Scale construction from a decisional viewpoint. In Minds and Machines, 16, p. 339-364.
[82] User Interface Markup Language (UIML) Specification Working Draft 3.1. (2004). Abrams, M. & Helms, J.
[83] User interface development: craft vs. engineering. (2006). Limbourg, Q. & Coyette, A.