Model-based approach for the design and generation of adaptive user interfaces on resource-constrained devices

Karim Yaici

Submitted for the Degree of
Doctor of Philosophy
from the
University of Surrey

Centre for Vision, Speech and Signal Processing, I-Lab
Faculty of Engineering and Physical Sciences
University of Surrey
Guildford, Surrey GU2 7XH, U.K.

January 2010

© Karim Yaici 2010
Abstract

While mobile computing devices have gained importance in recent years, thanks to their increased functionality and affordability, their diversity in hardware and software characteristics remains an obstacle to the provision of a common user experience across these heterogeneous systems. Furthermore, current user interfaces (UI's) running on these devices are developed with the assumption that the context in which they will be used is constant over time, including the profile of the user, the platform on which the user carries her tasks, and the environment in which the interaction takes place.

Traditional software design techniques are not adequate to support multiple contexts, since they require a partial or complete re-write of the user interface to match the characteristics of each device and the software running on it. This in turn necessitates additional time and resources. This situation is further complicated by the need of the user interface to accommodate varying contexts such as the user profile and environmental conditions. Consequently, there is a need to develop a new software methodology that is flexible enough to take into account contextual aspects early on the design phase, and enables the separation of the interaction aspects from implementation concerns. One way to effectively achieve this is to use abstraction.

To this end, we propose a model-based approach for adaptive user interface design and generation for resource-constrained devices. This approach consists in using a selected number of models which describe the UI at different levels of abstraction with specific terms and syntax. By focusing on the interaction aspects, it becomes possible to eliminate the need to rely on context-specific interaction capabilities, such as hardware and software, in the early stages of user interface development. It also becomes possible to enable context-adaptation at runtime, after the UI models have been defined. These models are specified using standard-based notations, and are then transformed and mapped to result in adaptive and adaptable user interfaces. A methodology to translate user requirements into a set of UI models is also introduced. Finally, a demonstrator application has been implemented to provide a practical example of how to apply these concepts and use semi-automatic processes developed as part of our research work.

Key words: Context-aware User Interfaces, HCI, Model-based User Interface Development, UML, XML
Acknowledgments

I would like to express my deepest gratitude and thanks to my supervisor Professor Ahmet Kondoz for many reasons, including giving me the opportunity to study for a PhD and work on exciting research projects, supervising my work and providing useful guidelines, being a friend and a person who I can trust and turn to for advice.

Many thanks go to EPSRC (Engineering and Physical Sciences Research Council) for partially funding my PhD and the I-Lab for providing such a stimulating research environment.

I would like to thank my friends, the people I spent most of my time with during the PhD years and who have supported me in many different ways. In no particular order, my gratitude to Wasim Ahmad, Zaheer Ahmad, Mustapha Ait Iften, Murat Badem, El-hadj Benkhelifa, Janko Calic, Safak Dogan, Erhan Eknecioglu, Sertan Kaymak, Bounediene Ramdani, Djelloul Mahboub, Xiyu Shi, Halil Uzuner and Emre Unver. Thank you also to those who can not be named for one reason or another but who had a profound impact on my life - for better or worse.

Finally, I want to thank my family for supporting me in every step of the way, for always being there for me, for their love and patience.
Publications

Conferences


Others


Terminology

AIO: Abstract interactor object.

AUI: Abstract user interface.

Application: The back-end logic behind a UI that implements the interaction supported by the user interface.

Context: Any entity that can influence the human to machine interaction and vice versa. The entity could be a person, place, or object considered relevant to the interaction.

Context-aware User Interface: User interface that changes with context. Adaptation can be initiated by the system (adaptive user interfaces) and/or user initiated changes adaptable user interfaces. See also Plasticity.

CASE: Computer-aided software engineering (CASE), tool used to help with the software development life cycle.

CUI: Concrete user interface.

CIO: Concrete interactor object.

Developer: The person who builds the UI using CASE tools and/or programming tools.

Designer: The person who analyses the application scenario and produces recommendations on the characteristics of the UI and its adaptation behaviour.

Document Type Definition (DTD): It is a set of markup declarations that define what XML elements and referents may appear in an XML document.

Device: A device is a physical object with which an end user interacts via a UI, e.g. PC, mobile phone, tablet PC, etc.

Framework: A set of processes and procedures.

MBUID: Model-based User Interface Development

Middleware: A software implementation of the framework.

Modality: A style of interaction.

Model: A model is a set of concepts and relationships that abstracts aspects of the UI.

User Interface (UI): Anything that enables and facilitates the dialogue between the man and the platform.

UI Element: A UI element is the primitive building block provided by any UI toolkit for the creation of UIs.

Platform: The combination of device, operating system and UI toolkit.

Plasticity: Ability of the user interface to adapt to context and be modified by the user.

Profile: It is an extension method (in the context of UML); it is (in context representation).

Rendering: It is the implementation of the UI on a given platform. It can be compiled with the application source code, or interpreted.

Stereotype: A Stereotype is a UML model element that is used to classify other UML elements. It may introduce additional properties (known as tagged values), additional constraints and a new graphical representation.

TERESA: Transformation Environment for inteRactivE Systems representAtions is a Java-based transformation-based environment that is used supports the design of an interactive application at different abstraction levels as well as generate of a user interface for various types of platforms using
XML-based representations. It also refers to the processes and procedures used to map and transform the UI models.

**TeresaXML**: A set of XML-based specifications for UI models used by TERESA, including the task model, abstract presentation model and concrete presentation model.

**Toolkit**: A toolkit is a software library upon which the application's UI can be rendered. In the context of this work, toolkit does not only mean software API's that need to be compiled, but also markup language that can be interpreted and rendered. The latter include VoiceXML, XML, HTML.

**UIDL**: User Interface Descriptive Language

**Widget**: See UI Element.

**XML Schema**: It is a document that contains a set of rules on which an XML document must conform in order to be considered valid according to that schema.
# Contents

1 Introduction .................................................. 1
    1.1 Problem Statement ........................................ 1
    1.2 Research Objectives ........................................ 3
    1.3 Dissertation Outline ....................................... 4

2 Design and Development of Context-Adaptive User Interfaces 6
    2.1 Introduction .................................................. 6
    2.2 Characteristics of Mobile Computing ...................... 7
        2.2.1 Dimensions of Mobile Computing .................. 7
        2.2.2 Dimensions of Mobile User ......................... 8
        2.2.3 Separation of Concerns ............................... 8
    2.3 User Interface Design Strategies ......................... 9
        2.3.1 Model-View-Controller (MVC) ...................... 9
        2.3.2 Presentation-Abstraction-Control (PAC) ............ 9
    2.4 User Interface Programming Paradigms .................... 10
    2.5 User Interface Software Architectures .................... 12
        2.5.1 Service-oriented Development Environments ...... 12
            2.5.1.1 Jini .................................................. 12
            2.5.1.2 OSGi ................................................. 12
            2.5.1.3 PUC ................................................ 13
        2.5.2 Proxy-based Development Environments .............. 13
            2.5.2.1 Draco ............................................... 13
            2.5.2.2 ICrafter ............................................. 14
        2.5.3 Reflexive Middleware .................................. 14
            2.5.3.1 Multi-layer AOP approach ...................... 14
            2.5.3.2 MobilePhoto ...................................... 15
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5.4</td>
<td>Agent-based Architectures</td>
<td>15</td>
</tr>
<tr>
<td>2.6</td>
<td>Model-Based Approach</td>
<td>16</td>
</tr>
<tr>
<td>2.6.1</td>
<td>Background</td>
<td>16</td>
</tr>
<tr>
<td>2.6.2</td>
<td>The Modelling Process</td>
<td>17</td>
</tr>
<tr>
<td>2.6.3</td>
<td>Core User Interface Models</td>
<td>18</td>
</tr>
<tr>
<td>2.6.3.1</td>
<td>The context model</td>
<td>18</td>
</tr>
<tr>
<td>2.6.3.2</td>
<td>The task model</td>
<td>21</td>
</tr>
<tr>
<td>2.6.3.3</td>
<td>Adding support for context in CTT</td>
<td>22</td>
</tr>
<tr>
<td>2.6.3.4</td>
<td>The dialog model</td>
<td>25</td>
</tr>
<tr>
<td>2.6.3.5</td>
<td>The presentation model</td>
<td>25</td>
</tr>
<tr>
<td>2.6.4</td>
<td>Model-based Development Environments</td>
<td>26</td>
</tr>
<tr>
<td>2.7</td>
<td>Proposed Approach</td>
<td>27</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Requirements</td>
<td>27</td>
</tr>
<tr>
<td>2.7.2</td>
<td>Transformation-based Technique</td>
<td>29</td>
</tr>
<tr>
<td>2.7.3</td>
<td>General Overview of UI Models</td>
<td>30</td>
</tr>
<tr>
<td>2.7.4</td>
<td>TERESA Environment</td>
<td>33</td>
</tr>
<tr>
<td>2.7.5</td>
<td>Task Model</td>
<td>34</td>
</tr>
<tr>
<td>2.7.6</td>
<td>Dialog Model</td>
<td>35</td>
</tr>
<tr>
<td>2.7.7</td>
<td>Presentation Model</td>
<td>36</td>
</tr>
<tr>
<td>2.7.8</td>
<td>Context Model</td>
<td>37</td>
</tr>
<tr>
<td>2.7.8.1</td>
<td>Definition and Classification of Context</td>
<td>37</td>
</tr>
<tr>
<td>2.7.8.2</td>
<td>Requirements</td>
<td>38</td>
</tr>
<tr>
<td>2.7.8.3</td>
<td>Context Categories and Usage Principles</td>
<td>39</td>
</tr>
<tr>
<td>2.7.8.4</td>
<td>Context properties</td>
<td>40</td>
</tr>
<tr>
<td>2.7.8.5</td>
<td>Context model XML schema</td>
<td>41</td>
</tr>
<tr>
<td>2.7.9</td>
<td>Context Adaptation</td>
<td>44</td>
</tr>
<tr>
<td>2.7.9.1</td>
<td>Choice nodes</td>
<td>44</td>
</tr>
<tr>
<td>2.7.9.2</td>
<td>External adaptation rules</td>
<td>45</td>
</tr>
<tr>
<td>2.7.9.3</td>
<td>Context adaptation process</td>
<td>46</td>
</tr>
<tr>
<td>2.8</td>
<td>Conclusion</td>
<td>49</td>
</tr>
</tbody>
</table>
3 UML-Based User Interface Design

3.1 Introduction ................................................................. 51
3.2 UML Notation ............................................................... 51
3.3 UML for User Interface and Context Modelling ................... 52
   3.3.1 UML for Task Modelling ........................................... 52
   3.3.2 Key UML-based Frameworks ...................................... 53
       3.3.2.1 WISDOM ......................................................... 53
       3.3.2.2 UMLi ............................................................ 55
       3.3.2.3 Context-sensitive User interface Profile (CUP) .......... 56
   3.3.3 Model-specific UML-based Frameworks ......................... 58
       3.3.3.1 Nobrega’s approach ........................................... 58
       3.3.3.2 LEAN CUISINE+ ............................................. 59
       3.3.3.3 XIS ............................................................... 60
   3.3.4 UML for Context Modelling ....................................... 61
       3.3.4.1 ContextUML ..................................................... 61
       3.3.4.2 Simple Mobile Services (SMS) ............................... 62
       3.3.4.3 Ayed and Berbers’ Approach ................................. 63
       3.3.4.4 The Context Modeling Profile (CMP) ..................... 65
   3.3.5 Discussion .............................................................. 65
3.4 Proposed Approach ....................................................... 67
   3.4.1 Introduction ........................................................... 67
   3.4.2 Modelling Tool ........................................................ 68
   3.4.3 Task Model ............................................................ 70
   3.4.4 Context Model ......................................................... 73
   3.4.5 Context Adaptation Model ......................................... 74
3.5 Mobile User Interface Development Process ......................... 78
   3.5.1 Introduction ........................................................... 78
   3.5.2 Software Development Cycle for Mobile User Interfaces .... 79
       3.5.2.1 Mobile use cases ............................................... 79
       3.5.2.2 Updated mobile UI development process ................. 82
       3.5.2.3 From scenario description to task model ................ 84
       3.5.2.4 Elaboration of adaptation rules ............................. 86
3.6 Conclusion ................................................................. 87
4 Markup Based User Interface Model Description

4.1 Introduction .................................................................................................................... 88

4.2 Markup-based Description Languages ..................................................................... 89

4.2.1 XML for UIDL ................................................................................................. 89

4.2.2 Review of XML-based UIDL's . ....................................................................... 90

4.2.2.1 AUIML .............................................................................................. 90

4.2.2.2 UsiXML .......................................................................................... 90

4.2.2.3 UIML ................................................................................................. 91

4.2.2.4 XForms .............................................................................................. 91

4.2.2.5 XIML ................................................................................................. 91

4.2.2.6 XUL ................................................................................................. 92

4.2.2.7 Others .............................................................................................. 92

4.2.3 Discussion .......................................................................................................... 92

4.3 Proposed Approach ....................................................................................................... 93

4.3.1 Introduction .................................................................................................... 93

4.3.2 UML and XML ................................................................................................. 93

4.3.3 Task Model ....................................................................................................... 94

4.3.4 Abstract Presentation Model ....................................................................... 96

4.3.5 Concrete Presentation Model . ......................................................................... 98

4.3.6 Final UI Description .........................................................................................101

4.3.7 Context Adaptation Model .............................................................................102

4.3.7.1 Condition validation and evaluation .............................................103

4.3.7.2 Task model structural operations ...................................................105

4.3.7.3 Specification of the initial task model .........................................108

4.3.7.4 Task level attribute operations .....................................................108

4.3.7.5 Concrete presentation level attribute operations ........................109

4.4 Conclusion ......................................................................................................................112

5 User Interface Generation and Adaptation .............................................................. 113

5.1 Introduction ......................................................................................................................113

5.2 The UI Management System ..................................................................................113

5.2.1 Recommended UIMS Architecture ................................................................113

5.2.2 Implementation Considerations ........................................................................115

5.2.2.1 Java Mobile Edition (JME) ..................................................................115
## Contents

5.2.3 Java ME and XML Support ........................................... 117
5.2.4 Java ME Graphical UI Rendering Engines ...................... 117
   5.2.4.1 Thinlet .......................................................... 119
   5.2.4.2 Kuix ............................................................ 121

5.3 Changes to TERESA Processes ........................................... 123
   5.3.1 Introduction ...................................................... 123
   5.3.2 Objectives ........................................................ 123
   5.3.3 From Task Model To Abstract Presentation Model ............ 124
   5.3.4 From Abstract Presentation Model To Final UI ............... 125

5.4 Amazon Book Browser Demonstrator ................................. 127
   5.4.1 Scenario and User Requirements ............................... 127
   5.4.2 Use Cases ........................................................ 128
   5.4.3 Generation of the Task Model ................................. 129
   5.4.4 From Task Model to Dialog Model ............................... 131
   5.4.5 From ETS to Abstract Presentation Model .................... 131
   5.4.6 From Abstract to Concrete Presentation Model .............. 134
   5.4.7 Generation of the Adaptation Model ......................... 136
   5.4.8 From Concrete Presentation Model to Final GUI ............ 140
      5.4.8.1 Accommodating the rendering engines .................. 140
      5.4.8.2 UI layout heuristics ........................................ 142
      5.4.8.3 UI styling preferences ...................................... 144
      5.4.8.4 Application of adaptation rules ......................... 144
   5.4.9 Implementation and Deployment ................................. 147
   5.4.10 Comparison with TERESA Default Rendering ............... 148

5.5 Museum Guide Demonstrator ............................................ 150
   5.5.1 Scenario and User Requirements ............................... 150
   5.5.2 Generation of the Task Model ................................. 151
   5.5.3 Generation of the Adaptation Model ............................ 152

5.6 Discussion ..................................................................... 156
   5.6.1 Mobile Usability Evaluation Techniques and MBUID ........ 156
   5.6.2 MBUID Approach Evaluation .................................... 157
   5.6.3 Context Model Evaluation ....................................... 160
   5.6.4 UI Notation Evaluation .......................................... 161
   5.6.5 UI Generation Tools Evaluation ................................. 162

5.7 Conclusion ..................................................................... 164
6 Conclusions And Future Work

6.1 Conclusions ................................................................. 165
6.2 Contributions ............................................................. 166
6.3 Future Work ............................................................... 168
   6.3.1 UML Modelling .................................................. 168
   6.3.2 UI Generation ..................................................... 169
   6.3.3 MBUID in Pervasive Computing Environments ......... 170
   6.3.4 MBUID and Multimodality ................................... 170
   6.3.5 User-generated Adaptation Rules ......................... 172
   6.3.6 Usability Evaluation ............................................ 172

References ................................................................. 173

A TERESA Model Specifications ......................................... 181
   A.1 Current User Interface Context Schema .................. 181
   A.2 Task Model Schema .............................................. 182
   A.3 Mobile Concrete Presentation Model Schema ............. 183
   A.4 Adaptation Model Schema ..................................... 186

B Introduction to UML Diagrams ......................................... 189

C Context-sensitive User interface Profile ......................... 193
   C.1 CUP .................................................................... 193
   C.2 CUP 2.0 ............................................................ 196

D Review of Java Mobile GUI Frameworks .......................... 199

E Amazon Book Browser .................................................. 201
   E.1 Final UI Model Descriptions ................................. 201
      E.1.1 Thinlet ......................................................... 201
      E.1.2 Kuix .......................................................... 202
   E.2 Amazon Book Browser Deployment Proposal ............ 204
      E.2.1 Server side ................................................... 204
      E.2.2 Client side .................................................. 206
      E.2.3 Client-server communications ......................... 206
List of Figures

2.1 Model-View-Controller design strategy .............................. 10
2.2 AOP approach to application development: (a) standard OO approach (b) with aspect weaving .......................... 11
2.3 Architecture of the implicit plasticity engine .................... 15
2.4 Model distribution and scope ........................................... 17
2.5 Example of task model with CTT notation ....................... 22
2.6 Contextual CTT with decision trees ................................. 23
2.7 New task categories defined in Contextual CTT ................ 24
2.8 Client-server framework presented by Bisignano et al. ....... 25
2.9 Use of transformation techniques with PAC for mobile user interface generation .............................. 30
2.10 The reference development process for supporting context-sensitive UI's ........................................... 31
2.11 Summary of work packages .............................................. 33
2.12 XML-based transformations supported by TERESA .......... 34
2.13 UI models’ structure and behaviour matrix ..................... 34
2.14 Presentation tasks sets generation in TERESA ................ 36
2.15 Screenshot of TERESA AUI to CUI Editor ...................... 37
2.16 Illustration of context profile composition ....................... 42
2.17 Information flow diagram of the choice nodes approach .... 45
2.18 Information flow diagram of the external adaptation rules approach ........................................... 46
2.19 Derivation of user requirements based on decomposition of scenario ........................................... 47
3.1 UML notation for CTT in WISDOM ................................. 54
3.2 Example of presentation model in UMLi of a search book user interface ........................................... 56
3.3 Context integration in UI modelling according to CUP ........ 56
3.4 Context-sensitive user interface profile .............................. 57
3.5 Comparison of three representations of the choice temporal operator: Nobrega’s/WISDOM/CTT ................. 59
3.6 Example of UI dialogue tree with Lean Cuisine+ notation .... 60
List of Figures

3.7 The multi-view approach of XIS ................................................................. 61
3.8 ContextUML metamodel ........................................................................... 61
3.9 The three levels of UML context modelling in SMS ................................. 63
3.10 Ayed and Berbers' context and adaptation profile ................................. 64
3.11 CMP UML Profile ...................................................................................... 65
3.12 MagicDraw main window .......................................................................... 69
3.13 Simplified CTT notation in UML ................................................................. 71
3.14 Simple task model example with CTT and UML .................................... 72
3.15 Context Model Profile ................................................................................ 75
3.16 Introduction of choice nodes to CTT specifications ................................. 75
3.17 Context Adaptation Model UML Profile .................................................. 77
3.18 Models and related tools in the user interface development process .......... 78
3.19 Example of Use Case diagram with adaptation support ......................... 82
3.20 Mobile UI development process ................................................................. 83
4.1 Example of AIO to CIO to final UI mappings ........................................... 99
4.2 Flowchart for the validation adaptation rules .......................................... 104
4.3 Illustration of the task-level 'add' operations ........................................... 106
4.4 Impact of applying adaptation rules on a task model ............................... 109
5.1 Reference UIMS architecture for context-aware adaptive user interface gen-

5.2 Deployment diagram of a typical Thinlet application .............................. 120
5.3 Deployment diagram of a typical Kuix application .................................. 122
5.4 Composition of TERESA transformation modules ................................ 124
5.5 Joining when 'Enabling ETS' grouping heuristic .................................... 125
5.6 Amazon book browser use case diagram ............................................... 131
5.7 Amazon book browser task tree in CTT ................................................... 132
5.8 Amazon book browser ETS ....................................................................... 133
5.9 Abstract presentation model of Amazon book browser .......................... 134
5.10 UML diagrams of a partial context model and adaptation model of the Ama-

5.11 UI navigation flow of the Amazon book browser application ............... 141
5.12 Transformation from TeresaXML to Thinlet .......................................... 142
5.13 Transformation from TeresaXML to Kuix .............................................. 142
5.14 Illustration of the different approaches to display output information . . 143
5.15 GUI renderings in optimal conditions ............................................ 145
5.16 GUI rendering in degraded network conditions (or at peak times) with no graphic support .......................................................... 146
5.17 Adaptation of the UI of Amazon book browser to different user preferences . 147
5.18 Final UI's generated with standard TERESA for the Amazon book browser . 149
5.19 Museum guide task model using standard CTT notation .................. 152
5.20 Museum guide task model using CUP 2.0 notation ......................... 153
5.21 Example of the context model for the mobile museum guide .............. 154
5.22 UML diagrams of a partial context model and adaptation model of the museum guide .............................................................. 155
B.1 Example of use case diagram ............................................................. 190
B.2 Examples of class diagrams .............................................................. 190
B.3 Example of sequence diagram .......................................................... 191
B.4 For loop using activity diagram ......................................................... 192
B.5 Example of deployment diagram for web browsing ......................... 192
C.1 UML profile for Contextual CTT ....................................................... 194
C.2 Example of CUP context model ....................................................... 195
C.3 CUP action stereotypes (a) user, (b) device, (c) interaction and (d) environment 196
C.4 Example of a system interaction model in CUP 2.0 for a museum guide application .................................................. 197
E.1 Deployment diagram of the Amazon book browser ......................... 204
E.2 Flow diagram of the UI management system .................................... 205
E.3 Component diagram of the Amazon book browser architecture - server side 206
E.4 Component diagram of the Amazon book browser architecture - client side 207
E.5 Communication protocol for context profile management ................ 209
List of Tables

2.1 Requirements for our model-based approach ......................................................... 27
2.2 Features requirements from the context model ..................................................... 38
2.3 Examples of context decomposition into atomic elements ................................... 47
2.4 Examples of context element impact on mobile UI ............................................ 48
2.5 Different transformation actions and the nature of their impact on the UI .......... 49
2.6 Structural, behavioural and style transformation operations per task type and corresponding CIO .......................................................... 50

3.1 WISDOM class and association stereotypes ......................................................... 54
3.2 UML 2.0 notation mappings for task temporal relationship according to Nobrega 58
3.3 Comparison of UML editing tools ...................................................................... 69
3.4 Current User Interface Context UML Profile ....................................................... 74
3.5 Example of mobile use cases evaluation matrix elaboration .............................. 81
3.6 Mapping between textual descriptions and CTT notation .................................. 85
3.7 Example of an adaptation rule ............................................................................ 86

4.1 Review of key high level UI description languages ............................................ 92
4.2 Mappings between AIO's and CIO's in TERESA ............................................. 99
4.3 Parameters required to add a new task .............................................................. 106
4.5 Examples of parameters for the adaptation rules .............................................. 111

5.1 Open source JME-compatible GUI libraries ....................................................... 118
5.2 List of events triggered by the most common widgets in Thinlet ....................... 121
5.3 List of events triggered by the most common widgets in Kuix .......................... 122
5.4 TERESA AUI to CUI mapping defaults ............................................................. 125
5.5 Mapping between Abstract/Concrete UI, Thinlet and Kuix widget tags .......... 126
5.6 Listing of tasks for amazon book browser application ...................................... 130
5.7 Abstract presentation structure for the Amazon book browser application .... 134
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.8</td>
<td>Detailed decomposition of the abstract presentation model for the first demonstrator application</td>
<td>135</td>
</tr>
<tr>
<td>5.9</td>
<td>Mapping between abstract operators and interactors, and their concrete equivalents</td>
<td>136</td>
</tr>
<tr>
<td>5.10</td>
<td>Mapping between concrete presentation units and XML files for Thinlet</td>
<td>141</td>
</tr>
<tr>
<td>5.11</td>
<td>Listing of tasks for museum guide scenario</td>
<td>152</td>
</tr>
<tr>
<td>5.12</td>
<td>List of context-sensitive tasks</td>
<td>152</td>
</tr>
<tr>
<td>5.13</td>
<td>Adaptation rules for the museum guide application</td>
<td>153</td>
</tr>
<tr>
<td>5.14</td>
<td>Comparison of our approach with other methods found in the literature</td>
<td>158</td>
</tr>
<tr>
<td>5.15</td>
<td>Comparison between the MBUID-based approaches</td>
<td>159</td>
</tr>
<tr>
<td>5.16</td>
<td>Evaluation of context and adaptation models</td>
<td>161</td>
</tr>
<tr>
<td>5.17</td>
<td>Key terminology from cognitive dimensions framework</td>
<td>162</td>
</tr>
<tr>
<td>5.18</td>
<td>Comparison between Thinlet and Kuix</td>
<td>163</td>
</tr>
<tr>
<td>5.19</td>
<td>Evaluation of Thinlet and Kuix</td>
<td>164</td>
</tr>
<tr>
<td>C.1</td>
<td>AUI stereotypes and XForms tags mapping</td>
<td>198</td>
</tr>
</tbody>
</table>
Listings

2.1 Example of UAProf profile ................................................................. 20
2.2 Example of Current User Interface Context ........................................ 42
4.1 Excerpt from a Task Model defined with TeresaXML .......................... 95
4.2 Excerpt from an Abstract Presentation Model defined with TeresaXML 97
4.3 Excerpt from a Mobile Concrete Presentation Model defined with TeresaXML 100
5.1 Partial XML file for the adaptation rules of the Amazon book browser 138
5.2 Partial XML file for the adaptation rules document of the museum guide
   application ................................................................................................. 154
A.1 Schema for Current User Interface Context model .............................. 181
A.2 Schema for Task model ....................................................................... 182
A.3 Schema for Mobile Concrete Presentation model .............................. 183
A.4 Schema for Adaptation model ............................................................ 186
Chapter 1

Introduction

1.1 Problem Statement

As the number of devices surrounding our daily lives increases and the use of different personal appliances is booming, the need to facilitate the design and deployment of a User Interface (UI) to accommodate the diversity of devices is emerging as a critical task. Furthermore, current user interfaces are developed with the assumption that the context in which they will be used is constant over time, including the user profile, the platform on which the user carries her/his tasks, and the environment in which the interaction takes place. However, the current trends have forced the review of these assumptions:

1. There are more people using connected devices (including mobile devices), and who have different preferences and usage patterns.

2. There is increasing number of mobile handsets thanks to the 'commoditisation' of mobile technologies. These devices come with different capabilities from low-end phones to high-end smartphones. Consequently, designing UI's for devices with such heterogeneous capabilities can prove particularly challenging.

3. Thanks to the increasing deployment of sensors in the mobile device itself (e.g. GPS) and around the users, more context information is available. This large amount of context data can be used to improve the user's interaction with the device.

Consequently, the design of interaction systems in a pervasive environment, which consists of heterogeneous systems interacting with each other and with human users in a transparent way, raises problems that require the development of new flexible methods in comparison with traditional methods. In fact, the two major problems for the design of interaction in pervasive computing environments is the diversity of the available devices and the constraints imposed by the interactive system itself (e.g. hardware, software and I/O modalities), the environment (e.g. network availability, noise level) and the user (e.g. user preferences, user capabilities). An interactive software system is no longer bound to a particular hardware/software platform, or designed for a single fixed context-of-use.

Traditional UI design techniques, which consists of a design, software specification and software implementation phases, are only suitable for single-context user interface design, and they are not
1.1. Problem Statement

adequate for multiple contexts. The problem with these approaches is that the design is highly context-specific and the resulting UI is generally hard coded in the application for a particular platform. Consequently, porting an existing UI to a different device or to be used in a different context implies the re-design and re-implementation of the complete UI to match the new usage characteristics and constraints. In addition, another consequence of the device-dependent implementation of the UI is the lack of support for adaptivity (ability of UI to adapt to change in the environment in which the application is operating) and adaptability (ability of the UI to adapt to user’s preferences).

When designing the user interface for these new pervasive computing environments, an abstract approach is necessary because the portable platform is diverse. These differences can be in terms of interaction resources (keyboard, stylus, speech input, etc.) and different screen resolutions (basic phones, feature phones and smartphones). The portability problem could be minimised with the use of a common software development environment, which enables the compilation and running of an application on a platform that supports that software environment, independently from the device capabilities and resources. Standards like Java Micro Edition (JME) [76] are bridging the gap between different portable devices to accomplish device-independent software development. However, the interactive part of an application is not as portable as the logic part because it has to take into account the specific device constraints. To the best of our knowledge there is no standard software environment for portable systems that makes this explicit in the framework provided to the UI designer and developer.

In the pervasive computing vision, computers are expected to connect in an ad hoc manner that is transparent to the user, providing seamless access to the services. This transparency implies that the application logic is separated from the application UI to enable remote access and adaptive UI to the application context-of-use. For these reasons, we believe that, to facilitate the work of UI software developers, a new software approach that maintains the separation between the interaction part and the business logic part is needed.

One of the techniques supporting this degree of flexibility for the user interface consists in using a number of models that describe the different aspects of the user interface. This approach is called Model-Based User Interface Development (MBUID), which eliminates the need to rely on the platform-specific interaction capabilities in the early stages of user interface development, and can be extended to support other sources of context such as the user and the environment in which the interaction occurs.

Model-Based User Interface Development uses a selected set of models to describe the different aspects of the human-computer interaction process, such as the user cognitive capabilities, user tasks, and the information flow of the user interface. In our approach, we intend to use five main models: context model, task model, dialog model, presentation model, and adaptation model, introducing improvements in model specification and integration. We also intend to define a methodology and develop a framework supported with tools to achieve a multi-context user interface design and creation cycle which stretches from context information representation to user interface realisation. The different models are transformed, derived, annotated and mapped to result in adaptive and adaptable user interfaces.

In summary, if we had to capture the problem we try to solve in a short paragraph, it could be as the following:

Current user interface design and development techniques lack the flexibility required
1.2 Research Objectives

This research work aims at exploiting current knowledge in the domains of Model-Based User Interface Design (MBUID), High-Level User Interface Description Languages (HLUID) and software modelling, to facilitate the creation of solutions to the problem stated above. The main objectives of our approach are summarised as follows:

Objective 1

As we have seen above, traditional UI design methodologies and processes are mainly oriented towards UI's for fixed-context such as desktop computers, and are less adapted to mobile UI development which are characterised by having a more dynamic context. In this work, we investigate how to design and develop these UI's by identifying a minimum set of models required for the task and defining their inter-relations. We also propose to incorporate context information using a flexible and extensible adaptation mechanisms, which directly affects the task and presentation models.

Objective 2

Traditional approaches to UI development do not incorporate context information in a sufficiently flexible manner, and are concerned with a limited definition of context which generally only considers the software and hardware characteristics of the device, and which is handled by the application execution environment, to which the UI is bound. We propose a more comprehensive, yet practical, context model which covers the user aspects, platform features and environment characteristics.

Objective 3

 Provision of common notation to support the design process of the user interaction for different context-of-uses, based on which developers can implement user interfaces in a more systematic and consistent way, and synchronise it with the design and implementation of the business logic of the application. To this end, UML was used to provide a more formal description of some of the models which can be understood and manipulated by software developers.

Objective 4

For increased abstraction and ease of user, we use high level languages to describe the aspects of UI including the task, dialog and presentation, in a way that is independent of the target devices and the modalities used for the interaction. Instead of developing a completely new set of languages, we have exploited existing notations which are based on XML, and made the required modifications to reflect the enhancements that have been introduced. By providing a dual notation (UML and XML), we aim to provide powerful representations of the UI models to support user developers in building multi-context UI's.

Objective 5

Objective 1-4 discuss the different parts that form the foundations of our approach. The final objective is to demonstrate the feasibility of dynamically generating mobile user interfaces that
resemble native applications and offer advanced features supported by modern graphical user interface renderers, such as richer widget set and customised styling. This is to show that MBUID can be effectively used to create fully working user interfaces that go beyond what has been achieved so far with HTML-based prototypes. This involves the development of software tools that process the models and enable their mapping and transformation. Special emphasis has been put on the support of resource-constrained devices and the ability to maintain the UI functionality in less-than-optimal usage conditions.

In this research, low-feature mobile devices\(^1\) will be used as a representative type of resource-constrained devices. Also, the main modality to be considered will be the Graphical User Interface (GUI). Our argument is that only a fraction of applications that are used on a daily basis require true multi-modality which encompasses more advanced capabilities such as text to speech encoders or graphics to speech. The cross-modal transformation is still a complex research problem, and solving it is beyond the scope of this dissertation. Besides, despite the advancements made in this field, the graphical user interface is particularly suitable for resource-constrained devices and remain the main modality used to access mobile services and manipulate mobile handsets.

As stated above, our approach aims to provide a complete solution for the specification, design and implementation of adaptive UIs. This approach is based on current practices which are improved at different stages to support context information and adaptation mechanisms. The approach is also supported by software runtime that is used for the processing of the UI models, including model transformation, mapping and derivation, application of adaptation rules, and generation and adaptation of the final UI. The methodology and runtime will be used to design and implement a functional demonstrator that uses two different GUI rendering engines and support context adaptation.

### 1.3 Dissertation Outline

The dissertation is organised in six chapters, four corresponding to the objectives and a final one is concerned with conclusions and future work. We generally start each chapter by reviewing the state-of-the-art in the related research area then expose our proposed approach while highlighting the difference with current and past approaches.

- **Chapter 2:** This chapter will provide a review of the different approaches to mobile UI design, with particular focus on model-based UI development (MBUID) and supporting platforms. We will also present a general overview of the models used in our approach and outline the novel ideas introduced with regard to context specification and adaptation.

- **Chapter 3:** This chapter will provide a review of the use of UML in UI and context modelling as well as provide details of UML extension to support our MBUID approach. In addition, it will give details about an updated software development cycle which combines standard processes and model-based tasks. We will also show how to extract UI model, the context-of-uses of an application and its associated context adaptation rules from a scenario description.

\(^1\)Handset that have an application execution runtimes like Java ME, from basic phones with less than 5k colours, small screen, no graphic support and mono/dual-band, to those with less than 65k colours, bigger screen, Tri-band/3G and supports graphics.
1.3. *Dissertation Outline*

- Chapter 4: This chapter provides a review of the state-of-the-art in mark-up UI description languages and how they can support our MBUID, from task model to final UI. We will also present in details the syntax of the context and adaptation models.

- Chapter 5: This chapter describes the changes that we introduced to the TERESA's model transformation processes, on which our approach is based, and explained the rationale behind our choice of GUI renderers. We also present two demonstrators to explain in details the steps to follow to develop a UI using MBUID and specify its adaptation conditions and effects, independently from the application's business logic. Finally, we present an evaluation of the modelling and implementation aspects of our approach.

- Chapter 6: This provides a summary of our contributions to the areas of UI model-based development in terms of model description and specifications, model transformations and the use of a context model and an adaptation model.
Chapter 2

Design and Development of Context-Adaptive User Interfaces

2.1 Introduction

Porting a desktop application into the mobile environment almost always results in less than optimal user experience. This is due to the differences between the two environments in terms of hardware and software capabilities, input/output modalities, display characteristics and usage style. These differences make it harder for developers to write their applications once and deploy them anywhere, and this is particularly true for the user interface.

Traditional methods for developing UI's use specialised stand-alone tools that consist in a linear process by which user interactions are modelled, analysed then user interfaces rendered. This is suitable for static UI's and do not work for new situations because the UI's are designed for fixed usage scenarios. As a result, each scenario requires the implementation of a new application. This approach results in fine-tuned UI's at the cost of lesser flexibility and longer development cycle.

In this chapter, we present a review of the main software approaches to designing mobile applications, with particular emphasis on those that support adaptation. We will focus in particular on the techniques that enable the design and development of adaptive user interfaces. After comparing the different approaches, we present our approach which is based on Mobile-based User Interface Development (MBUID). Finally, we discuss the novelties introduced with regard to context support and UI adaptation. To better understand the rest of the chapter, we first need to clarify some of the terminology used. We use the words design strategies, programming paradigms and software architectures to mean different things in the context of mobile UI design:

- Design strategies (or patterns) provide guidelines for UI design (such as Model-View-Controller).
- Programming paradigms indicate a style of developing programs and how its elements (e.g. objects, functions) are abstracted and represented, such as object-oriented programming (OOP) and aspect-oriented programming (AOP).
- Software architectures refer to the structure of the software program and the relationship between its elements, such as service-oriented architecture (SOA) and agent-based architecture.
2.2 Characteristics of Mobile Computing

2.2.1 Dimensions of Mobile Computing

A number of dimensions that characterise the mobile computing environment need to be considered when designing and implementing mobile applications and user interfaces. These dimensions are not completely orthogonal with respect to each other, but they are sufficiently separate to be distinct elements. The dimensions are as follows [6]:

1. **Location awareness**: Location provides a number of opportunities to enhance user experience and provide relevant information. Location can be absolute (e.g. provided by GPS) or relative to a reference point (e.g. in the office). Regardless of how this information is obtained and its accuracy, location is the major difference between mobile and stationary systems.

2. **Network quality of service (QoS)**: This represents a summary of a set of metrics that provides information about the connectivity between the end user device and the network and its performance (e.g. error rate, bandwidth, etc.), over wired or wireless networks. Because mobile applications depend on the network, they are more sensitive to QoS levels. However, application and UI need to continue to operate even when disconnected from the network.

3. **Limited power supply**: The design and implementation of mobile applications and their UI's is less affected by this dimension of mobility since the power management is managed by the OS and/or the software platform. However, we acknowledge the fact that it has an effect on the choice of platform and other architectural and implementation aspects.

4. **Available modalities**: It is nowadays common to find modern mobile devices that support various modalities such as voice control, touch screen displays and miniature keyboard. This multiplicity of modalities creates a challenge to mobile application designers and developers since they are constantly changing. Despite the advancement in the technologies, the UI development methodologies still fail to take into consideration the various concerns of the types of interaction. In addition, most mobile applications exhibit a strong coupling between the UI and application logic, which means that there is less flexibility on the design of the UI. As a result, most applications need to be re-designed to support a new set of user interfaces.

5. **Platform proliferation**: The availability of a variety of mobile handsets and platforms complicate the task of designing applications and UI's that work across them. For this reason, it becomes more important to favour methods that help develop applications and UI's that are independent from the platform.

6. **Limited device storage and CPU**: Despite the advancements in processing power and storage technologies in portable devices, the increasing complexity of applications and user interfaces that will be integrated into mobile devices to compete with the desktop computing platform, will continually require more storage and CPU. For this reason, mobile applications must be designed to optimize the use of data storage and processing power of the device. However, user interfaces do not get affected by this dimension as much as native applications, especially that modern operating systems and the use of virtual machines made it easier and cheaper to develop an application for multiple types of devices.
2.2.2 Dimensions of Mobile User

In addition to the characteristics of the mobile environment, the mobile user is also subject to variability. The conditions of the mobile user are not limited to the physical aspects of the user but extend to his/her preferences, expectations and his/her capabilities. User mobility is also an opportunity for developers to provide customised experience. The mobile user is different from the stationary one in different ways, including:

1. **Location**: User location can provide hints to what the current context of the user is and the types of connectivity used when using an application. Other underlying parameters contain the speed at which the mobile user may be travelling, and the mode of connectivity used.

2. **Lack of focus**: The constant mobility of the users, and multi-tasking nature of the mobile user lead to a lack of focus. Other modalities such as voice could be used to remediate to this problem.

3. **Immediacy**: Mobile users have higher expectations of performance from their devices than stationary users do. A short delay in application responsiveness can degrade the perceived user experience.

4. **Ubiquity**: The mobile user expects to be able to retrieve data and use it at any given moment and any given place.

These dimensions require solutions that need to satisfy conflicting goals from a technological point of point. For instance, a mobile applications need to be accessible from anywhere and lightweight, while at the same time developers need to make the user interface sufficiently friendly and responsive, even when the device is not connected to the network for an extended period of time. For this reason, there is a need to balance the solution to each of the dimensions of mobile while satisfying application requirements.

2.2.3 Separation of Concerns

The different dimensions of mobility affect the design and implementation of mobile user interfaces because the UI needs to accommodate the variations that relate to the dimensions of mobility. For these reasons, we need methods that can separate the concerns of mobility from the business logic of the application. In general terms, there are broadly three approaches to designing a UI to support multiple dimensions on resource constrained-devices:

1. Select a set of devices and usage scenarios that share similar characteristics and design a UI that only works best for them. This can be the case for handsets that are from the same family, such entry-level handsets, and for scenarios that have limited set of functions embedded.

2. Select the least common features of most devices and usage scenarios and design a UI that works in most cases. This is facilitated by the availability of widely deployed software execution environments. However, these frameworks do not provide a mechanism to support new usage requirements and adapt to them.
3. Rely on abstraction to describe some aspects of the human-device interaction, using for example standard notations such as XML, which are then transformed and translated into another format according to the characteristics of the context, so that the UI is optimised for the targeted device. This approach allows the separation between the UI part and the application logic part. This approach is the most promising, because it is the one that hides away the specificity of the context, permits more flexibility with regard to context support and exhibit less dependency on the software and hardware characteristics early on the design of the UI.

2.3 User Interface Design Strategies

In this section, we discuss two of the most prevailing design strategies for the separation of concerns between the interaction aspects of the UI and the application's business logic, MVC (Model-View-Controller) and PAC (Presentation-Abstraction-Control).

2.3.1 Model-View-Controller (MVC)

Model-View-Controller (MVC) is a design pattern for the separation of concerns of applications that involve an interaction with end user. MVC divides an interactive application into three areas: processing, output and input [10]. The Model is the internal implementation of the application and does not encapsulate any data or have any behaviour related to interaction with the user or presentation of data to the user. The View encapsulates any output through the UI to the user. The Controller processes the input of the user into the system. The system may have one or more views and controllers. The Controller allows the user to enter input which modifies the Model, and these modifications are reflected in the UI through the View (Figure 2.1). MVC allows separation of three different concerns: receiving input from the user (controller), implementing components that model business logic and operations that build the core functionality of the application (model), and presenting information to the user (view).

MVC is widely implemented in client-server and web-based applications, where there is usually one type of view and one type of controller, with possible support for multiple views. However, MVC can have limitations when multiple views are used, since it becomes difficult to maintain multiple UI's (e.g. audio, graphical, etc.) rendered through multiple channels which can come from numerous controllers.

2.3.2 Presentation-Abstraction-Control (PAC)

The PAC pattern defines a structure for interactive software systems in the form of a hierarchy of cooperating agents [10]. Each agent is a unitary aspect of the system, operating as a node in the agent hierarchy. Each agent consists of three components:

- Abstraction which abstracts away the core functionality and data used by the agent
- Presentation which provides access to the agent
- Control which controls the interaction between the interaction and presentation layers by passing messages between them
2.4 User Interface Programming Paradigms

It is similar to MVC in the sense that it hides the internal implementation of the logical functions of the system from the user interface (abstraction part in PAC is equivalent to controller in MVC). Since components are very decoupled, PAC can scale well, making it easy the composition of complex UI’s based on simple components. Consequently the PAC pattern fits the problem of mobile user interfaces much better than MVC because it provides well-defined hooks to handle the dimensions of mobility and affect the UI without exposing this functionality to the core logic of the application, which is embedded in the control component. PAC also provides one single layer for presentation, allowing to encapsulate the channels and modalities of the presentation in the same layer, which makes it more suitable to support for multimodality and adaptation.

Purely object-oriented (OO) design patterns mostly address statical flexibility i.e. adaptivity during the software evolution process. However, some OO patterns can be adapted to support richer adaptability at runtime. For instance, the Strategy\(^1\) pattern provides an infrastructure for dynamically exchanging a certain functionality at runtime. A Decorator\(^2\) pattern can be used to provide additional behaviour before the execution of a method, or even replace it completely with new functionality. Adding and removing decorators to/from objects can be managed dynamically by an intelligent manager. Similarly, a Visitor\(^3\) pattern could be used to allow adding new functionality at runtime without changing class code or compilation. However, these patterns are more oriented towards application adaptation rather than user interface adaptation. Making the business logic adaptable to changing context is certainly needed for a more complete context-aware service, however, it does add unnecessary complexity to the application without helping in the separation of concerns as far as the UI is concerned.

With the emergence of the Web, ubiquity of networking and the increased need for decoupling application functionality, service-oriented architecture (SOA), where software modules are provided

---

\(^1\)It allows the selection of algorithms at runtime.
\(^2\)It allows new or additional behaviour to be added to an existing class dynamically.
\(^3\)It allows the addition of new operations to existing object structures without modifying those structures.
as a service, has been successfully adopted to create scalable applications. SOA produces a loose coupling between the services and the underlying technologies and systems, and use data passed between the service requester and service provider.

In the last decade, we have witnessed the emergence of a new paradigm called Aspect Oriented Programming (AOP). AOP is a programming methodology which allows cross-cutting concerns to be declared as aspects [34]. AOP requires that a program is broken down into distinct parts. Conventional Object-Oriented Programming (OOP) techniques do not offer an easy way to change the design decision about how the system should adapt, because of the concern is scattered among several classes of the system and a code may have different concerns. AOP is a suitable approach since it allows the design decisions regarding different adaptation policies to be specified separately, making it easy to design them and to switch from one to another. These are called crosscutting concerns because they cut cross multiple abstractions in a program. An aspect is introduced to alter the behaviour of a program by applying an advice, which consists of additional code executed before, after or around various joint points - particular points in a program execution - specified in a pointcut, which indicates what part of the code or behaviour to intercept. The advice code is woven into the application code at compile time using an aspect weaver. This results in reducing code tangling and code scattering. AspectJ (http://www.eclipse.org/aspectj/) is the most known used AOP language.

![Figure 2.2: AOP approach to application development: (a) standard OO approach (b) with aspect weaving](image)

In fact, the above mentioned patterns are not mutually exclusive and it is common to find applications and service implemented using a combination of OO, AOP and SOA to increase modularity and maximise the separation of concerns. With the increased complexity of applications, these software patterns are being adapted to the mobile world by building mobile software architectures that enable the adaptation of applications and their UI's. We will review some of those architectures in the section below.
2.5 User Interface Software Architectures

In this section we will describe some of the software toolkits and middlewares that were developed or are currently under development to enable runtime adaptation of mobile applications, and in some cases that extend to the user interface. We define a middleware as being "a collection of services and functions that are needed by applications to function well in a networked environment" [33]. It provides abstraction to hide the complexities of the underlying sub-systems and mechanisms, and decouples application from any dependencies on the underlying layers (e.g. hardware platforms operating system, protocols, etc.). In the mobile UI domain, it represents a layer that resides between the application and the user interface layer for different usage scenarios.

For the sake of brevity, we will consider the software architectures that are pertinent to user interfaces development, and in particular those designed for resource-constrained devices. This list is by no means an exhaustive account of the state-of-the-art of the interaction aspects of ubiquitous computing, but a representative sample of the research activities in this area. We categorise those development environments into four categories depending on the central pattern they follow: service-oriented, proxy, reflection and agent-based architectures.

2.5.1 Service-oriented Development Environments

2.5.1.1 Jini

The Jini [77] framework is a service architecture that provides service discovery mechanisms to enable the location of services. This is a set of API's which enables easy creation and deployment of services. The traditional way of adding UI to Jini services is by adding attributes that hold UI-code that be instantiated on other hosts. This approach has been use by Artima Software in their ServiceUI project [3]. It can theoretically support arbitrary user interfaces (i.e. tradition GUI or voice interfaces), which can be developed by third parties and 'plugged' into existing services. To avoid the risk of overloading small devices with heavy services bundled with multiple user interfaces that would be instantiated instantaneously (as it would be normally done in Jini), ServiceUI defines descriptions (or macros) that indicate the purpose of the user interface (e.g. 'administration interface', 'about box') and the type of user interfaces (i.e. defined in terms of Java packages required to create the UI). Based on these descriptions, the client can select the UI it can render and request the downloading of required libraries, then instantiate the actual user interface. Note that Jini offers no specific support for the presentation layer.

2.5.1.2 OSGi

OSGi [1] is another open services platform similar to the Jini framework, with the advantage that it offers bridging functionality to link up with different kind of services (e.g. Jini, UPnP4, etc.). OSGi often uses the concept of a component which provides multiple services. Similarly, these components can discover and query each other for finding their functions. However, they are different in their

---

4The UPnP (Universal Plug'n'Play) architecture is a distributed (peer-to-peer), open networking architecture that enables seamless proximity networking in addition to control and data transfer among networked devices.
approach to connecting to each other, since they directly address each others’ interface and have precise information about the component connected to (in contrast to indirect connection using connectors). OSGi was originally developed for headless devices (i.e. devices with no display), which explains the lack of support for graphical user interfaces. Possible approaches that have been investigated include using web-based standards e.g. HTML, XML [36] and vector-based solutions e.g. SVG (scalable vector graphics, Macromedia Flash) [37].

2.5.1.3 PUC

Hodes [29] described a service centric approach whereby the device (or universal interactor) adapts its functionality to use the newly discovered services in the environment. This is accomplished with the use of translation protocols, which map the device functionality to a UI suitable for the representation of the service. A similar idea has been developed further by Nichols et al. [43] using the term personal universal controller (PUC), which is a remote control device that interfaces with complex appliances. The system includes a two way communication protocol and adapters to support proprietary appliance protocols (e.g. Jini, UPnP), a high-level specification language to describe the functions of an appliance, and interface generators that automatically build interfaces based on those specifications. An interesting novelty is the inclusion of dependency information, which describes the availability of each function relative to the appliance state. This technique enables better interface layout since redundant components are removed before generation.

2.5.2 Proxy-based Development Environments

The idea behind the use of a proxy is to decouple the application adaptation mechanism from the adaptation manager. This results in a clear separation of concerns between the adaptations available, adaptation mechanism and decision process. Policy specification is usually persistent text-based declarative representation of policy rules that can be read by users, programmers and applications. A rule is made up of an event specification that triggers the rule, which is often fired as a result of a monitoring operation, an action to perform in response to the trigger, and a target object that is part of the managed system upon which that action is performed.

2.5.2.1 Draco

Rigole et al. [65] present a component runtime infrastructure for pervasive environments, that enables the easy creation of distributed, interactive applications. The middleware supports the SEESCOA\textsuperscript{5} component-oriented design methodology. It uses the concepts of components (functional entities that make up the applications), ports (interfaces for asynchronous communications), connectors (set up a communication links between two ports) and contracts (define the specification of a component or port). The applications supported by this middleware are composed of interconnected components that send asynchronous messages to each other via their ports. An additional extension called Distribution Module (DM) is used to enable proxy-based distribution functionality to the core platform and is responsible for managing proxy components and generating them.

\textsuperscript{5}SEESCOA (Software Engineering for Embedded Systems using a Component-Oriented Approach) is a software methodology that aims at developing robust applications for high-end embedded systems. Home page: http://www.cs.kuleuven.ac.be/cwis/research/distrinet/projects/SEESCOA/
To illustrate its usefulness for distributed user interface development, a special type of Draco components, called Interaction Components (IC's), were created. An IC is basically a user interface rendering component that acts as a proxy between users and other service components. IC's can be located on one of the user's personal mobile devices where they provide access to the services of components that are available in the user's computing space. This interface is generated based on the high-level user interface description provided by these components. This description provides information about the hierarchical structure of the interface. More specifically, it provides presentation-level Abstract Interaction Objects (AIO's), which will be mapped to Concrete Interaction Objects (CIO's) during the UI rendering process. IC's have been defined to generate Java-based GUI's and HTML.

### 2.5.2.2 ICrafter

A somewhat similar approach was followed by Ponnekanti et al. [61] in their service framework for ubiquitous computing. It uses a central Interface Manager (IM) from which the user devices in the interactive workspace can request UI's for registered services. These requests result in the selection of a generator for that service, which sends a UI markup to the device to enable interaction with that service. This part is in contrast with Draco, in which HLUID of the UI is directly attached to the interaction component, and does not a central Interface Manager (IM).

### 2.5.3 Reflexive Middleware

Reflexive middleware is a system that can 'reason' about itself. This is generally achieved by maintaining a representation (a meta model) of itself which is linked to its own operation, so that if the system changes its representation, then the system adapts. The reflective system adapts its own behaviour by associating meta objects with the objects in the application, where meta objects control or adapt the behaviour of the application objections. This is generally achieved by adopting Aspect Oriented Programming (AOP). Some examples of the use of AOP in mobile computing are summarised below.

#### 2.5.3.1 Multi-layer AOP approach

Sendín and Lorés [69] proposed an AOP-based reflexive architecture that separates explicit plasticity (ability to automatically generate and redesign a UI, generally server based) and implicit plasticity (incremental adaptation due to environmental changes, generally locally in the device). Figure 2.3 shows the architecture of the implicit plasticity engine. The client-server architecture supports properties such as reification (making hidden aspects explicit and accessible) and reflection (reason about itself and act upon itself). The functionality of the application is left in the base level (logical layer), and the self-representation of the system is located in the metalevel. The latter is also responsible for controlling changes in the different contextual factors. The context-aware layer is in charge of detecting the environment and notifying changes to the meta objects. By using reflection techniques, they managed to separate functional and non-functional parts.

Furthermore, they made a mainly quantitative comparison between AOP, OO patterned and purely OO approaches in designing a mobile news reader [70]. They found that there is 51% increase in code
2.5. User Interface Software Architectures

size in the AOP-based version with respect to patterned and object oriented versions\(^6\). However, the aspectual version gains in reusability, orthogonality, pluggability and maintainability according to their evaluation.

![Diagram of Architecture of the implicit plasticity engine (69)](image)

Figure 2.3: Architecture of the implicit plasticity engine ([69])

2.5.3.2 MobilePhoto

Young [100] proposed a mobile photo application that adapts to the intrinsic differences exhibited by the device, such as Java virtual machine, supported API’s and hardware characteristics. His aims has been to facilitate the adaptation of the source code of an application to different types of mobile hardware and software platforms and enable additional functionality (e.g. adding new menu items) using weaving with AspectJ.

He used ANT (http://ant.apache.org/) and ANTENNA (http://ant.apache.org) to automate the compilation process by setting size constraints, features to include, and any optional packages use. He also implemented the same feature using OOP patterns and pre-processing to check for specific compilation requirements, creating the same effect as with AOP. In his evaluation, he found that OOP provides modularisation but suffers from code scattering, and requires adding a pre-processing statement. Furthermore, AspectJ offers more flexibility as it supports dynamic weaving, but has a side effect to increase application size.

2.5.4 Agent-based Architectures

A software agent is a self-contained application that provides a limited set of functionality and can communicate with other agents. In a mobile application context, a mobile agent can be in

---

\(^6\) They claimed that it is possible to create a demo on mobile devices though they did not provide an example.
charge of rendering the user interface, whereas another one can be responsible for tracking the device location. An agent can encapsulate sufficient intelligence to discover the capabilities of the device and immediately adapt the application to those capabilities that suit the device. The strength of this approach resides in its ability to distribute the tasks among different agents that communicate with each other and collaborate to handle a complex task such as generating and adapting a user interface to location. However, using a multitude of mobile agents on a resource-constrained devices is going to be resource prohibitive, which defies the point of creating a large degree of separation between the different parts of a given agent if the code base that makes up the agent is fairly small, and hence reduces its benefits. Much of the separation of concerns in such a situation can be provided by the host API that allows the agent to perform certain tasks. Breaking it up into user interface component, logic component, communication component, and control component may not have enough benefit to justify the associated cost and the increased dependencies among agents.

A typical deployment of mobile agent will be to have a set of agents responsible for specializing the generic user interface generated by another set of agents. In this manner, the agents that specialize the interface may migrate to the device and communicate with the agents that produce the generic user interface, using a client-server architecture. The advantage of using agents is that the agent that migrates to the client can take advantage of the mobile context, location sensitivity, and all of the other pieces of dimensions of mobility that are more accessible on the device as opposed to the network. Examples of such approach include Kao and Yuan [32], who proposed an XML-based environment for the separation of concern between the UI and the business logic, where software agents would handle most of the request processing, transformation and context response. They used XSL for transforming XML descriptions, their own LGML (Logic Markup Language) for event handling and an XML-based language to describe the UI. CC/PP [92] was used to describe the device capabilities and a customised web server to resolve HTTP requests that contain CC/PP.

### 2.6 Model-Based Approach

After reviewing the standard approaches used for designing mobile applications and their user interfaces, including strategies, patterns and architectures, we shift our focus to model-based approaches, and explain how they can complement, and in some instance, substitute traditional methods. In what follows, we provide a background review of the model-based approaches, its processes and the different models that compose this approach.

#### 2.6.1 Background

Although a number of papers have been written in the domain of Model-Based User Interface Development (MBUID), there is no consensus on the exact definition of this approach. At the core of MBUID is the idea of model. A model can be described as a set of concepts and relationships that abstracts aspects of the UI. The model's constituents can be described with primitives and terms that can be used to capture knowledge about the UI and its attached environment. Thus, MBUID can be thought of as a set of models where each model describes the UI at a specific level of abstraction with its own terms and relations (with intra-model relationships), and the different models can be related to each other (with inter-model relationships).
A natural question would be 'How many models are needed to capture knowledge about a UI?' A single model is probably too complex to create and manipulate easily, because of the number of entities and relations involved. A large number of models results in an increased number of inter-model relations. The central objective behind model-based approaches is to strike a balance between detailed control over the design of the UI and automation. The models cover a spectrum from the most abstract ones on one side, which describe design decisions independent from implementation considerations, and more concrete ones on the other side. All models should be consistent and precise, and contain as much information as possible about the system (Figure 2.4). There are different models that have appeared in the literature related to MBUID, most commonly domain model, application model, task model, dialog model, (abstract and concrete) presentation models and user models.

Domain and application models can be situated at the end of the application logic of the system. They describe interfaces to link up with the interactive system. The task model is closest to the user and specify the tasks the user executes. The dialog model and presentation models are closest to the final user interface. More recently, with the emergence of ubiquitous computing, the context has become an essential dimension that is factored in the model-based approach; thus the need for the context model. The latter describes the conditions in which the interaction takes place.

![Figure 2.4: Model distribution and scope](image)

**2.6.2 The Modelling Process**

The purpose of model-based design is to identify high-level models which allow designers to specify and analyse interactive software applications from a more semantic-oriented level rather than starting immediately to address the implementation issues. The model-driven approach consists of four main inter-related parts:

1. **Models**: As described above, they are representations which capture semantically meaningful aspects of the interaction at different levels of abstraction. By using models, designers can more easily manage the increasing complexity of interactive applications and analyse them both during their development and when they have to be modified.

2. **Modeling Languages**: Because concrete models are derived from earlier, more abstract models, both entities should be written in a standard, well-defined language. This way, manual
and automatic corrections can be made throughout the transformation process. Models are usually specified and visualised using a Model-Based User Interface Development Environment (MBUIDE).

3. **Transformation Rules**: The specifications of how models are transformed to other ones, and how code is generated are key to this process. They need to be specified in a way that keep them separated from the tools that execute them, so that they can be reused, extended and even manipulated with different tools.

4. **Transformation Tools**: The mapping, derivation and transformation applied onto the abstract models to generate the concrete ones, and then the source code for compilation or interpretation. This represents a central part in model-based approaches because it allows the automation of a substantial portion of the UI development process. These tools should offer the users the ability to tune these transformations to their specific needs. This transformation process can be carried out at design time i.e. part of UI generation process with one UI per context, or at runtime i.e. apply adaptation after the UI has been deployed.

### 2.6.3 Core User Interface Models

Some attempts have been made to provide a classification of UI models, which includes the work of Pinheiro da Silva [59] who surveyed fourteen MBUIDE's and identified four common models shared by most MBUIDE's which are: Application model, Task-Dialog model, Abstract presentation model, and Concrete presentation model. In the recent literature, we have also observed a number of recurrent models used for the design of context-aware UI's and have shown to be sufficient for the development of functional UI's. These models are:

- **Context Model**
- **Task Model**
- **Dialog Model**
- **Presentation Model (abstract and concrete)**

The context model, which affects the visual look-and-feel of the UI and its usage pattern, is a relatively recent addition to MBUID as current approaches aim to factor in the external parameters that influence the interaction between the user and the application. In addition, abstract and concrete presentation models are considered to be two facets of the same model. Finally, the application model has been excluded from this list because its structure and semantics change drastically with the change of technologies used for application development, software patterns used, how interaction actions and events are captured from the user interface, and how feedback is exposed on the UI, increasing the dependency of model-based development process on these aspects.

#### 2.6.3.1 The context model

Context is an extremely important factor in mobile computing, as it denotes the dynamic characteristics of mobile computing such as device capabilities, user's location and the noise and lighting levels of the physical environment. By incorporating context information into mobile computing, it
becomes possible to design systems and applications that leverage context information and provide more intelligent services in an unobtrusive way.

There are many types of context information, and their different properties lead to different ways to express and model them. According to the review given by Chen and Kotz [13] on context-aware mobile computing, numerous incompatible context models exist, with varying data structures to represent and exchange context information, but there are still no well-established techniques for modelling context according to Strang and Linhoff-Popien [75]. In the literature review, we have identified five main context modeling approaches:

1. **Markup-based Models**: Context data is represented using tags encoding, generally using a markup language such as XML, and defining the data as a set of tagged values e.g. `<context><location/></context></location>`.

2. **Key-value Models**: This simple scheme relies on associating each context element with a value. The advantage of this approach is that it is easy to manage context data but it lacks sophisticated structuring and efficient context retrieval algorithms. For instance, Schilit et al. [66] used a simple key-value pairs to model context information, such as location, as an environment variable in the application execution environment.

3. **Graphical Models**: These models are characterised by a strong graphical component. The most used model is the Unified Modeling Language (UML) [48]. Examples of such approaches include CUP 2.0 [84], ContextUML [71] and CMP [72]. They will be reviewed in section 3.3.4, where we explore the use of UML in context modelling.

4. **Object-Oriented Models**: These models use two concepts from object oriented modeling namely *encapsulation* and *reusability* to resolve parts of the problems arising from the dynamics of the context in ubiquitous environments. Context is abstracted from the sensors that generate the primitive data which are grouped into multiple collections of data that can be further combined to deduce higher-level, meaningful context information. The details of context processing is encapsulated in an object level and hence hidden to other components, whereas access to contextual information is provided through specified interfaces or methods only. Examples of such approaches include [30, 87].

5. **Ontology-Based Models**: An ontology refers to the formal description of a shared conceptualization of a domain of interest which is made of a set of entities, relations, instances, functions, and rules [27]. Notably, these approaches generally make use of OWL [89]. However, OWL requires reasonable computation power to enable context reasoning, which makes it less suitable for resource-constrained devices. Examples include Preuveneers et al. [62] who proposed an adaptable and extensible context ontology for creating context-aware computing infrastructures which incorporates information about user, service, environment, hardware and software aspects. Chen et al. [14] proposed a content broker architecture (CoBrA) using an ontology to describe *persons, places, and intentions* although less emphasis is put on service and user interface aspects. Finally, Henricksen and Indulska [28] presented a context model that describes context based on several types of facts e.g. sensed data, static and profiled.

Some of these modelling approaches, like graphical and object-oriented models, are more suitable for design purposes and to show the structure and behaviour of context elements. On the other hand, ontology-based modelling is more suited for reasoning about context information and deriving
meaning automatically. Finally, key-value and markup-based modelling, are more oriented towards context data and how they can be processed. In what follows, we provide more details on two XML-based models since it forms the basis of the notation proposed for our context model.

**CC/PP (Composite Capabilities/Preferences Profile)** CC/PP [92] is an RDF-based framework for describing software and hardware capabilities, user preferences and specific qualities about the user agent that affect the context processing and display. A CC/PP vocabulary defines specific components and their attributes, but it does not define a particular vocabulary. Instead, vocabularies are defined by other organisations or applications for specific domains. Furthermore, CC/PP also does not define a protocol for transporting an instance of a CC/PP vocabulary.

The CC/PP framework defines a relatively simple structure - a two-level hierarchy of components and attribute/value pairs. A component may be used to capture a part of a delivery context (e.g. network characteristics, software supported by a device, or the hardware characteristics of a device). A component may contain one or more attributes corresponding to the context parameters.

Improvements over CC/PP have been proposed, such as **CC/PP Context Extension** by Indulska et al [31]. They extended CC/PP vocabulary by a number of component-attribute trees related to some aspects of context such as application requirements and session information. The authors concluded that their approach is capable of enabling context awareness to applications, however they acknowledge that it is difficult and non-intuitive to capture complex contextual relationships and constraints due to limitations in CC/PP. These limitations stem from the fact that CC/PP can only support expressions with conditional OR's at the leaf levels and not a mixture of OR's of AND's. In addition, there is no standard for parsers specifically implemented for CC/PP, which means that parsing it with XML parsers will yield different serialisations.

**UAProf (User Agent Profile)** UAProf [52] is a variant of CC/PP with a specific vocabulary, resolution rules and protocol for transmission that have been adopted by handset manufacturers. It defines six base categories (or components) which group a number of attributes; these are Hardware-Platform, SoftwarePlatform, NetworkCharacteristics, WAPCharacteristics, PushCharacteristics, and BrowserUA. It has been mainly used to specify the capabilities of the mobile handset and WAP browser so that a web server can adapt the content accordingly. An example of UAProf profile is given in Listing 2.1 which specifies information about the web browser and the types of content that can be rendered on the mobile device:

```
Listing 2.1: Example of UAProf profile

<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ccpp="http://www.w3.org/2000/07/04-ccpp#"
    xmlns:uaprof="http://www.wapforum.org/UAProf/ccp_schema-19991014#">
    <Description about="http://www.example.com/MyProfile">
        <ccpp:component>

7The RDF is a metadata data model which is used to describe 'web resources' e.g. document, image, service, etc., in the form of subject-predicate-object expressions. The subject indicates the resource, whereas the object denotes an aspect of the resource, and the predicate expresses a relationship between the subject and the object.
The UAProf protocol supports both static and dynamic profiles. A static profile is accessed via a URI (Uniform Resource Identifier). This has several advantages: a client's request to a server can only contain a URI rather than the whole XML document, thus minimizing communication traffic. Also, the client does not have to store and/or create the profile. On the other hand, dynamic profiles are created on-the-fly and consequently do not have an associated URI. They may consist of a profile fragment containing additional data on top of the static profile. A request may contain any number of static profiles and dynamic profiles. However, the ordering of the profiles is important as later profiles override earlier profiles in the request. Since UAProf is based on CC/PP, its vocabulary can be extended by respecting some rules such as starting all components and attributes with upper case letter, and favouring the use of base profile components. However, it inherits the same limitations and has a relatively rigid structure, which makes it hard to support other types of context information.

2.6.3.2 The task model

The task model describes all the tasks to be carried out by a user in interaction with a system in order to reach a specific goal. Tasks are typically recursively decomposed into a hierarchy of actions (or subtasks), which represent the atomic operations.

Different techniques have been developed to specify a task model, some are more oriented towards identifying the actions and their logical decompositions, whereas others include temporal relationships and other information related to various concepts such as task objects, rules and agents.

Among the task modeling techniques used, the ConcurTaskTree (CTT) notation [55] is undoubtedly the one that has been most widely adopted by practitioners and researchers. In addition, the model specifications and transformation techniques that were developed around CTT have received wide acceptance and are used in numerous experimental studies (e.g. [38, 73]). The notation allows to describe the tasks that have to be performed by the users and the system to reach some pre-set goals.

CTT defines four types of tasks: User Tasks, which are the user cognitive/perceptive tasks; Interaction Tasks, which represent tasks that involve user's interaction with the system; Application Tasks, which are performed by the system; Abstraction Tasks, which refer to complex tasks that are composed of sub-tasks. In CTT, a task model is a hierarchical description of the tasks that the user needs to perform in order to reach a specific goal when interacting with a computer based system. A task tree is usually represented by a graph tree structure (Figure 2.5) where:

- The nodes represent the different tasks and subtasks to be carried out
2.6. Model-Based Approach

- The edges represent a decomposition relation (a task $t_i$ is decomposed into several subtasks $t_{i1}$ to $t_{in}$) between father-children nodes, or a temporal relation (e.g., two tasks can be performed concurrently or sequentially) between sibling nodes. Sibling tasks of the same level can be connected by different temporal operators (by priority): choice (||), order independence (|=|, the siblings can be executed in any order, but not concurrently), concurrency (|||, siblings can be executed concurrently), concurrency with information exchange (|||, siblings can be executed concurrently and exchange information), disabling (|>, the former sibling is disabled by the latter), suspend/resume (|>, the former task is suspended during execution of the latter), enabling (|>>>, the former task starts after the latter starts), and enabling with information exchange (|>>>, the latter starts after the former ends) and enabling with information exchange (|>>>, the latter starts after the former ends and uses information produced by the former node). Node and edges are augmented with icons to give visual clues on their type.

![Diagram](image)

Figure 2.5: Example of task model with CTT notation ([39])

Temporal relationships are used to identify a group of tasks that are enabled to start their performance during the same period of time; this grouping is referred to as Enabled Task Set (ETS). Tasks that are in one ETS will get transformed into a single presentation unit, or Presentation Task Set (PTS) in the presentation model. In addition, this process is supported by at least two tools, ConcurTaskTrees Environment (CTTE) [39] and TERESA [40], which allow the creation, editing and storing of a task model using a machine-readable format, and conversion to XML. The standard CTT does not support the specification of context information with the exception of platform information i.e. device type, screen size and graphics support and modalities enabled.

2.6.3.3 Adding support for context in CTT

The original CTT editing tools (TERESA and CTTE) provide a limited support for context information. In fact, designers can specify for each task the set of hardware platforms (e.g., mobile, desktop,
2.6. Model-Based Approach

etc.) that can support it, and can be ultimately rendered on the device. In this case, the designers are given the possibility to select one platform from those that have been specified within the whole task model, and make visible (i.e. active) only those tasks that support that platform. However, the amount of context information is constrained to pre-defined list of hardware platform categories, and second, it is only beneficial at the design stage where the designer needs to determine which tasks are enabled for a given platform. So, this approach does not allow the designer to specify multiple tasks nodes for different platform categories.

Pribeanu et al. [63] proposed an approach that achieves the clear separation of context-sensitive part of the task model from its context-insensitive part with the use of a context decision tree. The context insensitive part is modelled as normal CTT, whereas a decision tree represented by linking the different contexts-of-use sequentially as optional sub-tasks at the first level with the sequencing operator (\(\triangleright\triangleright\)), then the different contextual conditions are expressed as sub-sub-tasks with the choice operator (\([\])\), and the leaves of the decision trees become the roots of the context-sensitive part. The context-sensitive part, which is formed of a series of sub-trees is modelled in CTTE. The decision tree is represented by a marked structure annotation in the tasks representing its nodes. Each such task is augmented with the current value of the contextual condition considered at that time (Figure 2.6). The application of adaptation is performed manually and the resulting CTT is exported into an XML-compliant file. However, this approach focuses only on the task model and does not present an example of how it can be integrated into a semi-automatic process for generating context-adaptive UI’s. In addition, the introduction of decision tree breaks the compatibility with the existing CTT notation.

Clerckx et al. [15] augmented the CTT notation with a new element, the decision node, which specifies the contexts of use in which the task subtrees should be executed. Instead of collecting decision trees they use decision nodes (represented by corresponding task nodes marked with the letter D) to link context insensitive parts to context sensitive subtrees. The decision nodes are described by a notation where some context parameters are compared with some values, and decision is made to enable or disable the attached task subtrees. Ultimately, the context-specific CTT is generated by replacing decision nodes with subtrees that are suitable for the current context of use. This approach also introduces major changes to the CTT notation which makes the resulting task model incompatible with existing tools. And this requires the pre-processing of the tree by the runtime framework to

![Figure 2.6: Contextual CTT with decision trees ([63])]()}
incorporate context information and render it into the resulting UI.

Subsequently, Clerckx et al. [16] developed a visual tool to support their approach. The tools helps designers visually edit the context-specific task models after applying decision nodes and connecting the corresponding components from the distinct models as a result of model transformation. For the context model, Concrete Context Objects (CCOs), which indicate the source of context (e.g. sensor, preferences), are selected to define what context information influence the interaction. Their abstract counterparts (ACOs) represent abstract context information, and it is down to the designer to link the ACOs to the nodes of the task model by selecting a list of predefined interpretation rules to indicate which information to use and attach ACO’s to children of decision nodes to describe the corresponding task for each type of context. However, this tool has not been made public.

In [81], an extension to CTT notation was also presented, called Contextual CTT (CCTT). A new type of tasks, context tasks, are added which cause a change in the context-of-use of the application. They define four new tasks categories that are contextual versions of categories that are part of the CTT notation: application, interaction task and user task, in addition to a new “environment task/action” which is performed by an entity that is nor the system nor the user (as in CTT). They are represented by a C that overlays the symbol representing each task category.

![Task Categories](image)

Figure 2.7: New task categories defined in Contextual CTT ([81])

The advantage over [15], which uses decision nodes, is that it does not limit the modeling capabilities by supporting dynamic change of context, not just choice of static contexts. CCTT notation allows to describe reactions of the system on all the possible triggers and automatically establishes a connection between the context of use (the condition) and the context of rendering the UI (the result). However, the new notation has only been exemplified for few scenarios and no working demo has been presented.

Paterno and Santoro [56] presented a semi-automatic method to generate multiple interfaces for different contexts of uses. It uses CTTE tool to specify for each task the set of platforms suitable to support it. Then the task model is filtered according the target platform, though the resulting model may need to be manipulated by the designer. This model is mapped to an abstract presentation model after calculating Enabled Task Sets (ETS’s) and applying heuristics to obtain lower number of task sets (for instance, avoid repetition, grouping tasks that have data exchange). Finally, the UI is generated depending on the mapping between AIO’s and CIO’s.

Finally, Bisignano et al. [8] described a client-service UI generation framework that uses CTT notation for task model and some high level UI XML-based description languages (e.g. XHTML) to enable multi-device UI’s. Depending on the user device capabilities (PC, low end mobile, high end mobile), the rendering process occurs on the client or server side. The framework involves some content adaptation as well as event and session management. First a task model is created using the CTTE tool, then an XML description of abstract components of each task is computed, where the location of the processing happen and the processing parameters are specified. Device profiles are then exchanged, and the task model is retrieved and the UI is rendered, serialized and sent to the client.
2.6. Model-Based Approach

Although different types of renderers could be theoretically supported, only HTML-based UI's that are effectively generated on the server have been demonstrated. Besides, as far as UI adaptation is concerned, context information is limited to device hardware characteristics, offering little advantage over the original TERESA transformation process.

![Client-server framework presented by Bisignano et al.][7]

2.6.3.4 The dialog model

There is no agreement on the exact definition of what a dialog model ought to be. The best way to comprehend it is to situate it with respect to the other models. It is in fact an intermediate model between the task model and presentation model. Unlike task model representation, the dialog model reflects the information flow and how navigation between the set of tasks is executed. This model links up the tasks to the presentations units without specifying the internal structure of the dialog. Note that there is a one-to-one relation between a presentation unit and an enabled task set (ETS). This is to ensure that all aspects of the task model are reflected in the final user interface.

It is common to use the term task-dialog model to refer to the combination of task and dialog models. In fact, in throughout our work, we use the term task-dialog model, or simply task model, to refer to the initial task model as well how the task groups are formed (i.e. ETS) and temporally linked, which represent the dialog model.

2.6.3.5 The presentation model

This model extends the dialog model by graphical representations of its elements. It represents the structure of the UI, and can be modelled at three levels of abstraction: abstract, concrete and final.

- **Abstract User Interface Model (AUIM):** It defined a platform-independent structure and functionality, however there is no much consensus about the composition of the abstract model (compared with the task model). An AUIM is populated with Abstract Interaction Objects (AIO's) which are linked by abstract relationships. AIO's are generated based on the analysis of task composition (task model) and transitions (dialog model). AIO’s consist of abstraction of widget elements found in UI toolkits like buttons, and voice commands. An AIO is independent of any interaction modality and any platform. Abstract relationships indicate some spatio-temporal relationships among AIO's. These relationships characterise
the physical constraints between AIO's as they are presented in time and space. Examples of AIO's are 'action trigger', 'range indicator' and 'text input', and examples of relationships include containment, grouping and transition.

- **Concrete User Interface Model (CUIM):** This model allows the specification of the appearance and behaviour of the UI with elements that are directly visible to the user. In other words, Concrete Interaction Objects (CIO's) are potential implementation of AIO, for instance 'range indicator' AIO can be mapped onto a slider in Java for desktop or a gauge when using a Java for mobile devices. CIO's are modality dependent, so the same AIO can be rendered into different types of interfaces.

- **Final User Interface:** This is what the user will see and use to interact with the system or application. The end result depends on the platform characteristics (e.g. modalities, screen size), and UI libraries installed on the device.

### 2.6.4 Model-based Development Environments

One of the early work using a multi-model approach is Mastermind [78]; it used the presentation, application and dialog models to automatically generate the user interface. A similarly early tool supporting the domain model, which was on of the first models to be integrated in UI development, is DON [35]. Its layout mechanism supported a diverse number of screen-sizes, in a time where this diversity was only limited. Other early adopters of multi-model approaches to UI development include the work of Paternò and Leonardi [54], who developed a toolkit that could select UI widgets based on the task they could accomplish. Trident (Tools for an Interactive Development Environment) is another model-based system to create an interactive system by Vanderdonckt and Bodart [85]. It was one of the first design tools that recognised the importance of a clear separation between an abstract representation of the presentation model and a concrete representation thus supporting a multitude of interaction style alternatives for the same functional core. It also integrated task analysis as an important component to create a usable interface. Together with DON, Trident can be considered to be one of the first complete Model-Based User Interface Development Environments that where available.

Tadeus (Task Analysis/Design/End User Systems) by Schlungbaum [67] uses a user model, a task model, a domain model, a dialog model and was later supplemented with a presentation model. It relies on automatic generation of (part of) the dialog model from the other models. A similar framework, Dygimes [86], was developed, where a similar set of models were used, but more emphasis was put on multi-device user interface development, supporting an XML-based User Interface Description Language and a web service-based communication mechanism between the UI and the application's back-end.

The Cicero system [2] is an application-independent interaction manager that performs run time media coordination and allocation so as to adapt dynamically to the context, as to maintain coherent extended UI dialogs. A derived work was presented by Eisenstein et al. [23] in which they describe a set of techniques for the development of a consistent UI for several mobile devices and contexts. They describe a comprehensive UI description language (called MIMIC), which is derived from three models: platform, presentation, and task.

Finally, the CAMLEON (Context Aware Modelling for Enabling and Leveraging Effective Interaction) framework is a conceptual reference model for the development of distributed and plastic
2.7. Proposed Approach

As it can be observed from the above sample of research work, most of them focus on the design stage of MBUID, whereas others go beyond this stage to implementation and deployment. In this respect, our approach aims to support all UI development stages (from a software perspective) i.e. from the design to deployment stage. In the following sections, we will explain the reasons for opting for MBUID approach, and explain in details what models have been used, the transformations processes involved and the novelties introduced. We also highlight our contribution to better support of context adaptation in the task. On the other hand, the aspects of models specifications will be discussed in the subsequent chapters.

2.7.1 Requirements

We use the research objectives set in section 1.2 as basis to formulate the requirements for our approach. These requirements, which are listed in table 2.1, cover the design and implementation perspectives aspects.

<table>
<thead>
<tr>
<th>Objectives</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Enable the definition of user interface characteristics and semantics in a context-independent way</td>
</tr>
<tr>
<td></td>
<td>Allow the incorporation of context information and support for multiple contexts-of-use</td>
</tr>
<tr>
<td>2</td>
<td>Support for a comprehensive yet simple context model</td>
</tr>
<tr>
<td>3</td>
<td>Be easily integrated with existing methods and design tools to facilitate their adoption by UI designers and mobile application developers</td>
</tr>
<tr>
<td>4</td>
<td>Keep the early phases of UI design process free from implementation considerations as much as possible</td>
</tr>
<tr>
<td></td>
<td>Be consistent with the syntax and semantics of existing software notations such as UML and XML</td>
</tr>
<tr>
<td>5</td>
<td>Support model adaptation process</td>
</tr>
</tbody>
</table>

Table 2.1: Requirements for our model-based approach

We argue that a model-based approach is suitable to satisfy these requirements by offering a number of features which can enhance and complement the above-reviewed software design patterns in the following ways:
• **Better separation of concerns.** This is achieved at two levels:
  
  − Between the internal implementation of the business logic of the application and the implementation of the user interface. Component reuse, better scalability by distribution of processes, and easier code maintenance.
  
  − Between the device and interface-specific interactions of the user with the system and the different generic methods by which the user can affect the state and behaviour of the system. It allows the reuse of transformation components.

• **Abstraction.** Using models can take the design of the user interface to a more abstract level. In this way designers with knowledge about cognitive aspects of user interfaces are empowered to design the user interface without having to know or understand implementation specifics of user interface toolkits.

• **Automate user interface generation.** After abstract models are defined, tool support can be provided to assist user interface designers in transforming these abstract models into more concrete models and even a concrete user interface. This feature speeds up the user interface design process. This in turn can minimize the development effort and maximize the flexibility of the application to changes during the software life cycle.

• **Obtaining consistent user interfaces.** User interfaces built with the same MBUIDE maintain consistency. First, the user interface of a system will be consistent with the user interface on another system. Second, if a software update requires a new user interface, the models can be updated, and the resulting user interface that use the same MBUIDE will remain consistent with the previous one. It is also possible to dynamically introduce changes to the models to reflect changes in the context.

• **Application maintenance.** By having models described in a formal way, it is easier to edit and maintain the UI models while being able to visualise their structure using editing tools. Designers can then reuse parts of the UI more easily, and manage the increasing complexity of interactive applications and analyse them both during their development and when they have to be modified.

Since MBUID is a non-traditional approach in the sense that it prescribes the use of abstraction over implementation-specific techniques such as programming, it can be argued that the gap between indicating what the UI does and how to implement it, requires a new set of skills in translating UI behaviour and layout requirements into abstract models using domain-specific non-standard notations, which complicates the direct translation of those models. In addition, there is no consensus between researchers and practitioners on which set of models to use to describe UI's, which are essential and which are optional, and what is the minimum set of models that guarantees a usable interface. Currently, the modelling stage (real life to abstract form) is not fully formalised and it is not clear which aspects to include in the model. As a consequence, UIM's have not been widely used outside the academic arena, and have only been hardly deployed in commercial products, where they have been used for prototyping (e.g. TERESA). Another recognised limitation of MBUID is the unpredictability of the resulting UI because of the automation and the inter-model mappings introduced. This aspect can become more serious if adaptation is taken in consideration in the transformation process.
Specifically, when compared to OOP design patterns, despite their potential to enable the development of adaptive applications, relying solely on patterns means that we cannot reasonably cope with the requirement of context-sensitive adaptivity, because cross-cutting concerns will lead to a proliferation of adaptation code within the whole software, which will increase maintenance complexity. Regarding SOA, it helps in separating the interaction from the business logic to some extent, which could facilitate the support for separation of concerns between the application and the interaction, whereas model-based approach will ensure separation of concerns between the UI and the different contexts-of-use. Furthermore, AOP techniques are unable to specify conditions for dynamic adaptation and do not provide indication of how to incorporate user, application and environment context at runtime. In addition, AOP’s support for the resource-constrained mobile platform is not yet available in AOP frameworks such as AspectJ. That is because mobile execution environments, such as Java ME, do not support reflection, and few authors have discussed how to enable AOP aspects using workarounds on such platforms (e.g. [69]), and others have managed to implement prototypes that use their own unreleased code (e.g. [21] and [100]). However, these approaches do not support dynamic behaviour and runtime adaptation. Finally, Sendín and Lorés [70] have reported in the literature that there is a possible increase in source code size, reduced efficiency and lack of dynamic loading (due mainly to limitation of Java ME platform itself) when using AOP.

### 2.7.2 Transformation-based Technique

To move from early abstract models and generalised interaction descriptions to more concrete and more specialised interaction, transformation actions needs to be put in place. This may consist in the use of one or a combination of the two approaches listed below:

1. **Transcoding**: It focuses on extracting the information from the early models and create an intermediate format (usually using a markup language such as XML) that can in turn be used to produce other views.

2. **Transforming**: It aims to produce a presentation-neutral view of the system, the content can then be transformed to the appropriate views using XML styling techniques (e.g. XSL) or similar technologies.

In fact, both methods are complementary to MVC and PAC user interface design strategies, and applications that use them can enable access by different types of user interfaces for the PAC case (Figure 2.9(a)). In our approach, we have opted for transformation-based techniques since transcoding is a lossy process. More specifically, our approach is a cut-down version of the combination between PAC and transformation techniques, because we abstract the interaction aspects, provide a generic user interface representation (generic presentation), implement a set of software modules to sequentially transform the generic presentation to a specific presentation according to changing context-of-use (transformer), a component to produce the final user interface with which the users interacts and components to facilitate the messaging between the different components (control) (Figure 2.9(b)).

Using a combination of PAC and transformations, it becomes possible to distribute the processing between the end device and the server. Different servers can also be used to implement different transformation mechanisms and final user interface modalities. In addition, by using an XML-based description languages and transformations, it is possible to create a system with different degrees of
distributions, such as having both the presentation and transformer residing on the client device, or having shared processing between the client and server, or have everything on the network and make a device a thin client or browser that supports a lightweight UI description language based on XML.

Figure 2.9: Use of transformation techniques with PAC for mobile user interface generation

2.7.3 General Overview of UI Models

To determine the models that we will use for our approach and formalise the transformation used, we rely on the reference framework for plastic UI’s introduced by Calvary et al. [12]. The authors identified four levels for producing context-sensitive UI’s (Figure 2.10).

1. A Concepts and Tasks Model which connects a task model and a concepts model (which describes the concept of the domain of discourse).
2. An Abstract UI Model which defines a computing platform-independent rendering of the above concepts and relationships as they are required by the task in terms of presentation units.
3. A Concrete UI Model which transforms the above platform-independent rendering into a platform-dependent rendering.
4. A Final UI consists of the generated code required to compile/interpret the source code of the UI from the above concrete UI.

For each context-of-use, $C_1$, each level is subject to an iteration, that implies a redefinition of re-composition of that model at the same level of abstraction to accommodate the design requirements. Reification is the process of transforming an abstract level into a more concrete one, to ultimately produce the final UI. Another context-of-use $C_2$ in the figure can be reached through the process of translation, that is a transformation of a UI description initially intended for $C_1$ but re-used for another description of the same level of abstraction but tailored for the new context-of-use.

Specifically at the task level, Thevenin [79] introduced two notions: decoration, which consists of expressing particular configurations of the task model depending on the logical conditions that represent the current context of use, and factorization which consists in expressing common configurations of the task models depending on the same logical conditions.
2.7. Proposed Approach

These two notions can serve as basic operations for the composition of various approaches to modelling tasks for multiple contexts of use. Thevenin proposed an approach whereby one task model is built for each context of use one after another, combining the resulting separate task models into a large one by performing factorization, and indicating at the key nodes, whether such sub-tree should be implemented for this contextual conditions, using decoration. If we only keep factorisation and bypass decoration, than we can also generate a task model that is most common to all contexts of use, which will result in the minimum set of functions across the different contexts of use. A third approach consists in creating the most comprehensive task model with all sub-tasks for all contexts of use, derive from it a specific task model for each context of use by applying decoration. A last approach consists in creating a task model for contexts-of-use that are considered representative of most cases (e.g. selecting most important nodes, most frequents tasks, most comprehensive one, etc.), and apply decoration when appropriate. The latter is the approach that we have selected since it offers the best balance between generality and specialisation, on the assumptions that mobile devices have more common features and their contexts-of-use are more recurrent.

Our approach is primarily inspired by the work of Paternò [57] and the use of TERESA MBUIDE, in the sense that we follow a top-down approach of refinement, starting from the task model, to concrete presentation model. One advantage of this approach is that all the concrete interface/ languages share the same structure and add concrete platform-dependent details to the abstract language on the possible attributes for implementing the various interaction objects and the ways to compose them. We also use descriptive languages which are based on XML for any abstraction level in order to make them more easily manageable and allow their export/import in different tools.

Another advantage of this approach is that maintaining links among the elements in the various abstraction levels allows the possibility of conveying semantic information which can be exploited to guide and refine UI implementation. A further advantage is that designers of multi-context interfaces do not have to learn all the details that may influence the implementation and behaviour of the UI because designers have control over the design through the logical descriptions and leave part of
the implementation and adaptation to automatic processes. In addition, if a new implementation language needs to be addressed, the entire structure of the environment does not change; only the transformation from the associated concrete level to the new language has to be added. This is not a complex task since the concrete level is already a detailed description of how the interface should behave and be structured. In summary, we propose the following nine-stage approach for the production of a context-aware user interface:

1. **Production of the context model**: This consists of the formal description of the external elements that influence the user’s interaction with the system.

2. **Production of a context-sensitive task model**: The starting point is to create a task model which takes into consideration all sub-tasks that might be needed in a single context of use or in multiple contexts of use. This activity may involve adding, removing or modifying sub-tasks depending on their need in a particular context-of-use. The dialog model would reflect the grouping of the sub-tasks and the information flow between the set of tasks when executed.

3. **Definition of adaptation rules**: These rules specify the conditions and the action to be applied on the different tasks of the task model, and CUI objects of the concrete presentation model.

4. **Application of task-level adaptation rules**.

5. **Generation of a context-specific task model**: This model is generated by applying the adaption rules. It is possible to generate as many task models as there are adaptation rules.

6. **Generation of an abstract UI model**: This generic presentation UI model is supposed to define a platform-independent rendering of the task-dialog model, for instance, subtasks can be grouped and then mapped to Abstract Interaction Objects (AIO’s) at the presentation level.

7. **Application of concrete presentation-level adaptation rules**.

8. **Generation of context-specific concrete presentation model**: This model results from the application of the adaptation rules. Multiple models could be generated as a result.

9. **Generation of a final running UI**: The concrete presentation model can then be exploited at design time to automatically create the code required to run the UI or be interpreted at run-time to produce the expected UI.

To enable each of the steps described above, our work encompasses a number of contributions in model processes, model specifications and software components. Figure 2.11 depicts the areas covered by our work and their inter-relationships. Model processes include inter-model mapping of model components and how transformation rules are applied to the different models. Model specifications concern the notations used to describe the model. Software components concern the software modules implemented on the client or server side to support the model processes and notations as well as context management and UI rendering. Note also that some of these building blocks belong to more than one category, such as model transformation which has modelling and implementation dimensions.
2.7. Proposed Approach

2.7.4 TERESA Environment

TERESA (Transformation Environment for inteRactivE Systems representAtions) [40] is a Java-based transformation-based environment that is based on CTTE [55] and supports the design of an interactive application at different abstraction levels as well as the generation of a user interface for various types of platforms, with support for CTT and XML-based notations. It provides a flexible environment for designers to mix manual and automatic processes of model specification and transformation, using a set of visual tools which facilitate the introduction of changes and the creation of prototypes. The environment supports four main transformations\(^8\) (Figure 2.12):

1. Generation of presentation sets. Derive enabled task sets (ETS) and related transitions from the XML or CTT specification of a task model. This grouping depends on the application of a number of heuristics which are supported by the tool (see section 2.7.6).

2. Generation of abstract user interface. The XML task model and ETS model are used to create the associated abstract user interface. The static structure (presentation part) and dynamic behaviour specifications (dialog part) of the abstract user interface is stored to enable further transformations.

3. Generation of platform-specific concrete presentation. This transformation uses an abstract user interface as input generates the related concrete user interface for the specific interaction platform selected, by mapping each abstract interaction object (AIO) to a concrete interaction object (CIO) as specified by the designer before the execution of this transformation.

4. Generation of the final UI. Based on a number of default configuration settings for the desired platform, the final user interface is generated. TERESA supports XHTML and VoiceXML as markup for the final UI.

\(^8\)Note that we only summarise the different processes, since more details can be found in the original literature e.g. [40]
2.7. Proposed Approach

2.7.5 Task Model

According to the models' matrix presented by Van den Bergh and Coninx [83], which provides a categorisation of the UI models with respect to their dependency on context (i.e. the deployment platform), the (concrete) presentation and dialog models are context-specific versions of the (abstract) presentation and task models respectively (Figure 2.13). The upper half of the figure contains the platform independent models (PIM), and lower part contains the platform specific models (PSM). Besides, the models on the left describes behavioural models, and on the right are those related to structure. While this matrix looks at the UI as being made up of a structural and a behavioural part represented by specific models, our approach looks at what aspects of the UI can potentially change in reaction to a change in the context-of-use, and how it could be applied. Furthermore, our definition of context-of-use goes beyond the limited scope of the hardware and software platform of end user's device.

Our argument is that since the abstract model is generated from the task model (via the dialog model), it is possible to centralise the behavioural and structural transformations on the task model. As a result, it is more effective to generate a context-specific task model as early as possible so that these changes can propagate to the other models, instead of introducing changes at every stage and generating context-specific models. The task model defines the different 'actions' (from the user, application and system perspectives), their decomposition into sub-actions and their temporal relationships, so any change in this model has important repercussions on the structure and behaviour.
of the UI. For this reason, it is important to make the task model the first model to be context-sensitive. As for the modelling notation used for the task model, we have selected the ConcurTaskTree (CTT) because:

- It allows designers to concentrate on the relevant aspects of the interaction that encompass both the user and system-related aspects, without worrying about the low level implementation details early on.
- It allows different levels of granularity thanks to its hierarchical structure.
- It uses a graphical syntax (i.e. icons) which helps quickly grasp the meaning of the notation.
- It has a rich semantics set that cover nodes and temporal operators, which makes its usage scope very large.
- It has been used to create usable UI's using a graphical tool (CTTE) and the model-transformation tool (TERESA), which facilitates its adoption among practitioners.

2.7.6 Dialog Model

The task model is used to derive enabled task sets (ETS), also known as presentation tasks sets (PTS) in the context of TERESA and CTTE. This grouping depends on the application of a number of heuristics which are supported by the task model editing tool to obtain a lower number of task sets. TERESA supports four heuristics to the grouping of tasks into ETS, which can be set from the "presentation tasks sets generation" window:

1. **Joining when Enabling.** If two PTS's differ for only one element, and those elements are at the same level connected with an enabling operator, they are joined together (Figure 2.14(a)).

2. **Single elements Sets.** If a PTS is composed of just one element, it is joined with another PTS (Figure 2.14(b)).

3. **Sharing most elements Sets.** If some PTS's share most element, they are unified (Figure 2.14(c)).

4. **Exchange information.** If there is an exchange of information between two tasks, they are included in the same presentation in order to highlight such information transfer (Figure 2.14(d)).
2.7. Proposed Approach

2.7.7 Presentation Model

In addition to the structural and behavioural aspects of the UI that are described in the task and dialog models, we provide an additional layer of adaptation which is the presentation style of the UI, because it enables the customisation of the interaction and the application of visual enhancements. We will use TERESA’s XML-based notation for the presentation models (i.e. TeresaXML\textsuperscript{9}) in most parts, with slight modifications to the concrete presentation model notation as we have added new objects and amended existing ones to accommodate the UI rendering engine. These changes will be discussed in more details in section 5.3.4. Besides, since we are focused on one modality, namely graphical UI's, there is no need to modify the abstract presentation model.

The concrete presentation model is similar to the abstract model in its structure, but it is composed of more concrete modality-specific interaction objects. Since, the concrete model is the closest to the user, its result (the final UI) is the most visible expression of the interaction's actions and behaviour defined in the task model. For the mobile platform, when using the desktop version of TERESA for the conversion from abstract to concrete UI, the designer is presented with a list of three types of mobile devices, with different screen sizes and different degrees of graphics support, and the designer needs to select one platform, as shown in the Figure below.

\textsuperscript{9}http://glove.isti.cnr.it/tools/TERESA/teresa_xml.html
2.7 Proposed Approach

2.7.8 Context Model

Defining and modelling context is a complex and difficult task. Most of the approaches typically use some elements of the overall context such as location, time and user activity. In this work, we propose a simple, yet extensible, XML-based context model to capture and communicate context information. We will first provide some definitions of the concepts related to the terms context and context of use. Then, we will present the details of our model as well as its serialised version, which uses XML to store and communicate context information.

2.7.8.1 Definition and Classification of Context

The topic of context adaptation is very wide because it encompasses a variety of research areas. For the purpose of this study, we will narrow our scope to themes related to context adaptation which are relevant to UI design and development. In the following paragraphs, we will present a short overview of the context and how it can be specified.

While Context has been the subject of numerous scientific works, there is still no consensus on the exact meaning of context and what it should encompass. The significance of the term “context” has changed following the advances in context-aware applications and the accumulation of experience in them. Initially the term “context” referred to the location and identity of user and objects. Then the term expanded to include the environment assuming three components: computing system, user and physical environment. Instead of listing all the definitions found in the literature, it is sufficient for the purpose of this work to present the most accepted definition. In fact, Dey [22] proposes the following definitions for context and context awareness, which we believe are balanced descriptions of those terms, in the sense that they are general yet practical definitions:

**Definition 1:** Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant...
2.7. Proposed Approach

to the interaction between a user and an application, including the user and applications themselves.

**Definition 2:** A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

2.7.8.2 Requirements

Approaches encountered in the literature, and more specifically in MBUID, such as CUP 2.0, offer UML-based models which are used for design purpose, but not a way to serialise the models so that they can be used to store and communicate context information and trigger adaptation. As a result, we believe that there is a need for a dual-purpose modelling of the context model. Our approach consists in defining a context model using a notation that is familiar to software developers, but at the same time be serialisable into a human-readable format which can be directly exploited for implementation purposes (for instance, as part of the context management middleware). That means that there is no need to reason about context, but rather favour the use of markup language for context data processing, over ontology-based notations. As a result, the context model should have the characteristics listed in Table 2.2, with respect to its ability to be used for design purpose and also for data processing.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Implication</th>
<th>Design</th>
<th>Data processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantics/comprehensive</td>
<td>Richness in describing context data and related concepts</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Generality</td>
<td>Technology-independent context specifications</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Structured/Formality</td>
<td>Need for formalism to ensure its correctness and validity</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Concise/Lightweight</td>
<td>Minimal format and low processing requirements</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Design/Implementation</td>
<td>Ability to use it for design and implementation purposes</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Extensibility</td>
<td>Ability to define new context entities and characterise them</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Standard-based</td>
<td>Based on common software design and also graphical in nature</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Tool support</td>
<td>Support for editors and generator</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2: Features requirements from the context model

We need to represent the context with two representations, graphical and markup-based, to accommodate the need to use it in the design stage of software development as well as during implementation, making each approach most suitable for each purpose, and making one translatable to the other. In our work, we have opted for XML for the data model of the context, and UML for the graphical description of the model. The rationale behind this choice is briefly presented below:

- **Markup language:** The advantage of using XML lies in the fact that it's hierarchical data structure consisting of markup tags with attributes and content that can be recursively contained. Our approach consists in defining context data attribute and values as XML tag
attributes and values respectively. Context elements can then be characterised in terms of their nature, source and value.

- **Graphical language:** In software engineering, the Unified Modelling Language (UML) is the de facto modelling framework for describing the composition of an application, its behaviour, and how it is deployed. Our approach consists in defining an extension to UML to support the semantics and structure used in the markup-based version of the context model. As far as software developers are concerned, the use of a graphical representation makes it easier to comprehend the composition of context information and its sources, which can then be mapped to the UML models used for database modelling (e.g. user-specific context profiles) and middleware development (e.g. context management). The use of UML for context modelling will be discussed in section 3.4.4.

To keep the size of the context profile's vocabulary low, we defined a small set of tags while still being able to describe the context data in a concise manner and making it generic enough to support different types of context data. Regarding its syntax, we used a simple structure so that UI designers can specify and edit context profiles manually by keeping to the simple two-level hierarchy of context information (i.e. atomic and composite elements), as defined in section 2.7.9.3.

### 2.7.8.3 Context Categories and Usage Principles

Calvary et al. [12] categorise contextual elements into platform and environment where, the platform represents the physical and software platforms used for the interaction, and the physical environment is where the interaction is taking place. They refer to the combination of platform and environment as the context-of-use. We argue that the environment as described by Galvary et al. has an open definition as it may encompass anything around the interaction. For practicality, we have decided to split the environment dimension into two context components creating three categories of context information in mobile computing:

1. The **user** context, which includes user’s identity, capabilities, preferences, location, social status and role, etc.

2. The **platform** context, which describes the hardware and software characteristics of the end-user devices to ensure that the runtime system is in place to support a given application. It may include static and dynamic computing contexts. Static computing context includes processor speed, storage capacity, screen size, wireless connections, operating system, application execution runtime, software libraries (e.g. GUI engines, etc.). Dynamic context may include network bandwidth and delay, neighbouring mobile devices, etc.

3. The **environment** context, which includes anything beyond “platform-user” entities that has direct influence on the interaction, such as lighting conditions, noise level, etc. One of the characteristics of the environmental aspect is that it is essentially task-dependent and the boundary of what is included is left to the UI designer to define.

Therefore, we considered the triplet User-Platform-Environment, as we have redefined them, as the building blocks of the context model in our model-based UI generation process. Thus, we formally define the term context-of-use (or simply context) as being the environment in which the user is carrying out an interactive task to fulfil a goal using a specific platform.
Regarding the modelling aspect, we associate a profile with each context category defined above. Therefore, a profile encompasses the different context parameters associated with that particular category and their values at a specific time, as follows:

- **User profile.** This profile describes the characteristics of the user. These attributes can be indicative of user preferences, as well as the cognitive and physical capabilities of the user.

- **Platform profile.** This profile captures the hardware and software characteristics of the device on which the UI is rendered, and which will affect the ‘realisation’ of the UI. These attributes may include screen size/resolution, software support for a particular modality, connectivity, etc.

- **Environment profile.** This profile describes the physical environment in which the interaction is taking place. Attributes may include the user location, and information about lighting, noise level, number of access points in the surrounding area, etc.

### 2.7.8.4 Context properties

To simplify the process of building a unified structure to the context model, we proposed to categorise the different contextual parameters along three axes. From a source perspective, the context profile is composed of three sub-profiles as indicated above: platform, user and environment. Each sub-profile is structured in the same way, containing a list of context parameters which when combined make up the current context-of-use. Each atomic context element also specifies its type, the nature of the context parameter it is representing, its associated value and the UI level(s) that are targeted by adaptation. Each context is time stamped to inform the context processor about the time of creation (or reception in the case of differential profile), and is assigned a unique ID. A context parameter is identified by its name, which should be unique within a given sub-profile, but different sub-profiles can include a context element with similar names, such as time. The properties of the context parameters explained below:

- **Type:** As explained in section 2.7.9.3, the context-of-use can be decomposed into more granular components, and they can be represented using XML tags. A context can be atomic or composite; An atomic context represents a basic context element that can be fully represented using a single parameter describing a uni-dimensional information e.g. sound level, and a composed context is made up of a number of atomic context parameters, each describing an aspect of the captured context information e.g. location is made up of latitude and longitude.

- **Nature of context:** This parameter indicates the change frequency of the context element. *Static* characterises context parameters that do not change over the usage session, like hardware, software characteristics and most of user preferences. *Dynamic* characterises context parameters that do change over the usage time such as network characteristics and location. Static contextual parameters can be incorporated during the design phase of the user interface development or during the initial exchange of the context profile, while dynamic parameters are used during run-time rendering of the UI. Note that the nature of each context is determined by the nature of the composing context parameters i.e. it is dynamic if at least one of the constituent parameters is dynamic.

- **Value:** This indicates the current value of the context parameter. It could be an arithmetic value or string of characters.
2.7. Proposed Approach

- **Level of impact**: As explained in section 2.7.9.3, this parameter is used to indicate what UI models are potentially affected by a change in the context element. The value of this parameter is checked against the corresponding target model associated with the corresponding context parameter in the adaptation rule, and if they match than the rule is applied (see section 4.3.7 for more details).

2.7.8.5 Context model XML schema

In this section, we present the details of a lightweight XML-based notation used for the context model that corresponds to the structure defined above, and is specifically targeted at devices with low processing power and memory. This representation complements that of the UML-based specification which will be presented in section 3.4.4.

At the top, the context profile constitutes the *Current User Interface Context* (CUIC) (indicated with XML tag `<CurrentUserInterfaceContext>`), and which corresponds to the current context-of-use. It is composed of three sub-profiles (indicated with XML tag `<profile>`) namely, user, platform or environment. As per the specifications given above, we associate each context profile with a unique ID identifying the combination of user-platform-environment. We also timestamp each context profile to enable the server to keep track of the versions of context profiles. The latter is relevant when using differential profiles since the context management system can keep track of the latest version of the full context profile. The use of differential profiles is relevant when updating dynamic context properties and when requiring to override the composition of the complex context element. Note that differential profiles can be used to add new context parameters and change the values of existing ones, but does not support the deletion of context parameters, which means that the content of context information can only increase incrementally.

In practice, the profile should be attached to the current client’s session which is managed by the adaptation server. The context profile is initially assembled at the client and transferred to the server at the beginning of the session. However, this does not mean that all context information must be located at the client and transferred via the network. The context profile may rather contain references to external profiles, such as the environment profile, which could be stored in a local processing using in the vicinity of the client, or the user profile which could be provided by the service provider (Figure 2.16).
The decomposition of this profile into three sub-profiles also allows the partial update of the CUIC, which is an effective way to keep the UI adaptation server up-to-date of the current situation of the mobile user. It also allows the specification of two different context parameters that might share the same name but are associated with two different categories, such as time (by contrast, CC/PP does not allow to do that). After the first transfer of the context profile, the client does not need to re-send the full context profile during the lifetime of the session but simply updates the existing context parameters by sending a differential profile which contains only the information that has changed since the last exchange of profiles. The default mechanism is similar to CC/PP in that it overrides attribute values that have changed, removing the need to change the full sub-profile during the lifetime of the session. New parameters can also be inserted in the suitable sub-profile by indicating the context type, name, nature and value. It is also possible to add new parameters to a composite context proving more granularity to the definition of particular type of context\textsuperscript{10}.

Each sub-profile is in turn composed of a number of context descriptions. A context element (indicated with XML tag <context>) can define an atomic context, or a composite context which is made up of a number of context parameters (indicated with XML tag <ctxParameter>). Each atomic context or context parameter is associated with a name, value, affectedUILevels and nature. Note that the nature of the context and its affected UI levels are defined at the individual context parameter level.

An example of a context profile is presented in Listing 2.2. This example demonstrates the expressiveness of the schema, and illustrates the different approaches to defining context information. It combines a user, platform and environment sub-profiles and describes the network conditions, presentation characteristics and location. Some of the context elements are atomic, others are composite with one or multiple context parameters (e.g. location or network conditions). The full syntax and semantics of the Current User Interface Context is presented in Appendix A.1 using XML schema.

\textsuperscript{10}However, it is not possible to delete an atomic context element or a context parameter of a composite context once it has been defined in the original context profile.

Listing 2.2: Example of Current User Interface Context
2.7. Proposed Approach

<?xml version="1.0"?>
<CurrentUserInterfaceContext xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="current_context.xsd" ctxEntityID="Con04" timeStamp="12:20:46.275+01:00">
  <profile type="user">
    <context name="preferences">
      <ctxParameter name="Navigator Choice" nature="static">
        <value>Graphic Link</value>
      </ctxParameter>
    </context>
    <context name="presentationCharacteristics">
      <ctxParameter name="backgroundColour" nature="static">
        <value>white</value>
      </ctxParameter>
      <ctxParameter name="fontName" nature="static">
        <value>helvetica</value>
      </ctxParameter>
      <ctxParameter name="fontSize" nature="static">
        <value>12</value>
      </ctxParameter>
      <ctxParameter name="displayOrientation" nature="dynamic">
        <value>horizontal</value>
      </ctxParameter>
      <ctxParameter name="inputModality" nature="dynamic">
        <value>keyboard</value>
      </ctxParameter>
      <ctxParameter name="outputModality" nature="dynamic">
        <value>graphical</value>
      </ctxParameter>
      <ctxParameter name="speedOfNavigation" nature="dynamic">
        <value>4</value>
      </ctxParameter>
    </context>
    <context name="activity">
      <ctxParameter name="latitudemodality" nature="dynamic">
        <value>101</value>
      </ctxParameter>
      <ctxParameter name="longitudemodality" nature="dynamic">
        <value>64</value>
      </ctxParameter>
    </context>
    <context name="platform">
      <ctxParameter name="JVM" nature="static">
        <value>Java</value>
      </ctxParameter>
      <ctxParameter name="screenSize" nature="static">
        <value>418x128</value>
      </ctxParameter>
      <ctxParameter name="networkCondition">
        <ctxParameter name="bandwidth" nature="dynamic">
          <value>128</value>
        </ctxParameter>
        <ctxParameter name="delay" nature="dynamic">
          <value>100</value>
        </ctxParameter>
      </context>
    </context>
  </profile>
</CurrentUserInterfaceContext>
2.7. Proposed Approach

As explained above, our approach to context adaption uses a set of rules which are applied onto the original task model and concrete presentation models. In our exploration of how to best implement this feature, we have initially adopted the use of choice nodes, after which we have used external adaptation rules. In the following chapters, we provide details on both approaches.

2.7.9 Context Adaptation

Our first approach to supporting contextual changes of tasks, was to augment the XML-based notation used in TERESA for the task model to implement a logic-based selection of task trees. We define choice nodes, which are based on the concept of decision nodes introduced by Clerckx et al. [15], with the difference being the way the conditions are defined. In addition, active attributes were added to the task definition [97].

Our approach offered few other improvements over decision nodes. It combines the pre-processing of the task model (for structural changes) at the node level, with the post-processing of the presentation model after its derivation from the modified task model. By contrast to work presented by Souchon et al. [73], where they created multiple variations of the task model based on the current context of use and merged the different parts into one large task model, our approach supports a maximum of two variations by node, simplifying the structure of the task model. In addition, Souchon's case, they do not provide a runtime (or tool) that can be used to generate a final and adaptive UI.

In our approach, each choice node is associated with a contextual condition. The latter is composed of a context parameter, an arithmetic operator and a preset value. When the statement is evaluated to true, the right child of the node is selected, otherwise the left child is selected, and the node is replaced by one of its children. Each node also has an active attribute to indicate whether the associated task is active or not. When a task is inactive, the task tree is recomputed to reflect the current state of the task model, without deleting it completely from the tree. This entails the recomputing of the attributes of the temporal relationships for neighbouring tasks whenever a task node changes its activity state. When a node's activity status is set to false, the node's temporal relationship is checked before removing it from the contextualised task tree, while converting it to a standard task tree. If temporal relationship is one among choice, order independence or concurrency then the check succeeds, otherwise the check fails. The adaptation rules in response to context change are applied to the tree first followed by restructuring of the tree as a result of the active attribute change.

Figure 2.17 depicts the information flow diagram of the approach using choice nodes. An embedded action is first applied on the large task model (1), then the process of transformation and derivation is executed (2), followed by an external adaptation action applied on the resulting concrete presentation model (3), to finally generate a context specific concrete presentation model (4).
2.7. Proposed Approach

The introduction of the choice nodes enabled a localized task modification, which was sufficient to support adaptation to simple context information, such as platform type, and not specify more complex contextual conditions that contained logical operands within choice nodes. However, this approach had a number of shortcomings:

- **Need to use choice nodes to make tasks context-sensitive**: That means that task nodes which have not been replaced by choice nodes at the beginning of the model definition stage cannot be made context sensitive afterwards.

- **Each choice node is defined with a condition and associated branch**: That means that we cannot use a single contextual condition to trigger changes on multiple tasks and sub-trees.

- **Limited operations**: During pre-processing of the initial task model to generate the contextualised task model, only a choice of the child nodes.

- **Choice nodes add unnecessary complexity to the task model**: since the adaptation mechanism is incorporated in the task model, the initial task model needs to incorporate all possible tree compositions, rendering the tree very complex.

- **Cannot create complex conditions**: It is not possible combine conditions with logical operators to create nested conditions.

- **Incompatibility**: Like the decision trees and decision nodes found in the literature, this approach breaks the compatibility with existing CTT editing tools.

### 2.7.9.2 External adaptation rules

To simplify the specifications of adaptation rules and the execution of the adaptation process, we separated the adaptation rules and adaptation mechanisms from the task model. We then grouped the specification of contextual conditions and adaptation actions for the task and the concrete model, hence circumventing the need to have choice nodes [98]. In the improved version, actions cause a change in the structure as well as in the attributes of the task model, and different actions can be applied on different models for the same contextual condition. An additional advantage of this
2.7. Proposed Approach

approach is that the initial task model does not need to include the anticipated context adaptation specifications, making the task model less complex.

Notably, our approach involves the post-processing of the task model which has the advantage that applications that are not designed with context adaptation in mind can be augmented to support context adaptation provided that they have been modeled using CTT in the first place. This way, the preparation of adaptation rules can be carried out after the task models associated with the service have been defined, giving room to modify the graphical user interface after deployment of the service.

Figure 2.18 depicts the information flow diagram of the approach using external adaptation rules. In comparison with the choice nodes approach, an external action is first applied on the default task model followed by the process of transformation and derivation.

![Information flow diagram of the external adaptation rules approach](image)

Figure 2.18: Information flow diagram of the external adaptation rules approach

### 2.7.9.3 Context adaptation process

To formulate adaptation rules, we need to classify the context elements according to their potential impact on the different UI models by providing a taxonomy of context information. Next, we have to quantify and link the effect of context change on the UI to the different models. In the following paragraphs, we will illustrate our approach to defining the context of use and describing the adaptation rules.

Within the e-Sense European project\(^{11}\), we have created a taxonomy of context information for sensors-based context-aware services [24, 99]. We proposed a hierarchical structure of context information, where the atomic context elements are used to construct composite context information, and the different types of context information are then combined to create a unique context-of-use which describes the external factors that influence a service. We referred to these high level context elements as context building blocks (CBB) because they are essential to the realisation of the sensor networks.

\(^{11}\)This European project aimed to develop the networking and application stack for services that make use of sensor networks
2.7. Proposed Approach

proof-of-concept demonstrators and recurring across the different scenarios that have been defined within the project. Three types of application 'domains' were considered: lifestyle (for individual applications), community (for group-level applications) and industrial applications.

The process by which user requirements are extracted from the scenario is illustrated in Figure 2.19. By analysing the scenario, we derive the high level context elements or CBB's (1). Examples of context decompositions of typical context elements considered in the e-Sense project is shown in Table 2.3. Starting from these CBB's, we identify the types of information needed to capture this context. Once the informational requirements are identified, we look for the types of data that lead to the definition of these types of information (2). Each data type hints to the sensor payload type (3). Indeed, the results of analysis at each stage are summarised in a table where each context building block is associated with information and data types provided by the sensors. The sensors are grouped according to their location (e.g. on the body, in the environment, etc.) and are characterised in terms of quantifiable device characteristics (e.g. sensor lifespan, reliability, etc) (4).

![Figure 2.19: Derivation of user requirements based on the decomposition of scenarios (adapted from [99])](image)

<table>
<thead>
<tr>
<th>Context information</th>
<th>Atomic context</th>
<th>Type</th>
<th>Sampling rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physiological context</td>
<td>Skin Conductance</td>
<td>Physical</td>
<td>125 Hz</td>
</tr>
<tr>
<td></td>
<td>Heart Rate, Heart Rate Variability and Pulse</td>
<td>Physical</td>
<td>250Hz</td>
</tr>
<tr>
<td></td>
<td>Breathing Rate</td>
<td>Physical</td>
<td>125 Hz</td>
</tr>
<tr>
<td></td>
<td>Facial Muscles</td>
<td>Physical</td>
<td>250 Hz</td>
</tr>
</tbody>
</table>

Table 2.3: Examples of context decomposition into atomic elements

In the current work, we have followed the same approach in identifying and decomposing the context, by linking the CBB's with their potential impact on the final user interface. In Table 2.4, we present an example list of context building blocks, their nature and their potential impact in the UI of mobile
2.7. Proposed Approach

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters (type)</th>
<th>Nature</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>name of OS (string)</td>
<td>static</td>
<td>modalities, look-and-feel</td>
</tr>
<tr>
<td>Lighting</td>
<td>luminosity (integer)</td>
<td>dynamic</td>
<td>modalities, brighten up screen</td>
</tr>
<tr>
<td>Font size</td>
<td>pt (integer)</td>
<td>static</td>
<td>widget attribute</td>
</tr>
<tr>
<td>Connectiv</td>
<td>bandwidth, latency (integer)</td>
<td>dynamic</td>
<td>structure (e.g. disable view)</td>
</tr>
<tr>
<td>QoS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>latitude, longitude</td>
<td>dynamic</td>
<td>structure (e.g. enable a new view)</td>
</tr>
</tbody>
</table>

Table 2.4: Examples of context element impact on mobile UI

Application. Such list could serve as guidelines to the UI designers prior to the elaboration of the adaptation rules. This is obviously not an exhaustive list of context elements and the relationship stated depend greatly on the usage scenarios and user requirements of the application, which means that such mapping need to be defined on a per application-domain basis.

As can be seen from above table, each change of context can lead to a different type of change that affects the UI. We can categorise these changes broadly into three groups, according to the scope of their impact:

- **Structural**: This relates to the composition of the final UI. Context change could lead to the addition or removal of a set of widgets, change in the order of the tasks, change in grouping of widgets, nature and type of widgets used, etc.
- **Behavioural**: This relates to the functions exposed by the final UI. Context change could affect the transitions between UI screens, addition or removal of a UI function, etc.
- **Style**: This relates to the visual features of the final UI presentation. Context change could affect background colour, font size, placement of widget on window, text field length, etc.

A transformational action is associated with the atomic context component which is identified to trigger a change in the UI structure, behaviour or visual style. The change in the value of a context parameter (or a combination of parameters) act as triggers for the action. To enable adaptation, the context parameter is evaluated against the contextual condition set in the adaptation rule using a comparison operator such as = or >. The triplet (context parameter(s), operator(s), context value(s)) constitutes the condition part of the adaptation rules, whereas the action part is made up of a series of transformations to be applied onto the UI models. These are described inside the adaptation model. This model will be discussed in more details in section 4.3.7.

Based on the model matrix presented in Figure 2.13, we define the primary models that need to be altered during the UI generation process. In this respect, we argue that the structural change can originate from a structural change to the task model, whereas its behaviour can be controlled from the task model through the attributes of the task nodes and their temporal relationships. Structurally, a UI can change by adding, removing and substituting task nodes or task subtrees, and ordering the nodes in a particular way, or changing the hierarchy. Finally, the presentation style of the final UI interfaces can be manipulated in the later stages of transformation, via a change or extension of the attributes of the concrete presentation model. Table 2.5 provides a matrix that depicts the potential combinations of transformation actions that can be applied onto the different models and their effect on the final UI.
2.8. Conclusion

In this chapter, we have presented a literature review of the different software methodologies and patterns proposed for the design and implementation of user interfaces, with some examples of frameworks that have adopted such approaches. However, they are characterised with inherent complexity, and offer limited support for resource-constrained devices like mobile phones. This is further complicated when support for context adaptation is added to the set of requirements. In this situation, model-based user interface design, which constitutes the foundation of our work, is an adequate solution for this task since it clearly separates the interaction concerns from the application business logic and contextual aspects, and relies on abstraction and visual notations. This approach is also characterised by the use of semi-automated transformation and processing tools which help the designer to focus on interface design rather than worry about implementation issues early on.

In addition to the standard task/dialog and presentation models generally found in the literature, we have introduced the context model, used to capture the different usage situations, and the adaptation model, which specifies the when and how dimensions of user interface adaptation. For this, we have provided definitions for the terms context and context awareness, and presented a review of the literature on the subject of context modelling for UI development, with a focus on markup-based

<table>
<thead>
<tr>
<th>Task (structure)</th>
<th>Task (attribute)</th>
<th>Concrete (attribute)</th>
<th>UI Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action 1</td>
<td>x</td>
<td></td>
<td>Structure</td>
</tr>
<tr>
<td>Action 2</td>
<td>x</td>
<td>x</td>
<td>Structure + Behaviour</td>
</tr>
<tr>
<td>Action 3</td>
<td></td>
<td>x</td>
<td>Behaviour</td>
</tr>
<tr>
<td>Action 4</td>
<td></td>
<td></td>
<td>Visual Style</td>
</tr>
<tr>
<td>Action 5</td>
<td>x</td>
<td></td>
<td>Structure + Visual Style</td>
</tr>
<tr>
<td>Action 6</td>
<td></td>
<td>x</td>
<td>Behaviour + Visual Style</td>
</tr>
<tr>
<td>Action 7</td>
<td>x</td>
<td>x</td>
<td>Structure + Behaviour + Visual Style</td>
</tr>
</tbody>
</table>

Table 2.5: Different transformation actions and the nature of their impact on the UI

Table 2.6 shows the possible structural, behavioural, and visual layout operations that can be applied onto the different task node types (as defined in the CTT notation) and corresponding concrete interactor objects (as defined in TERESA). By knowing how the UI widgets need to change, it is possible for the UI designer to formulate the action part of the adaptation rules. At this point, the UI designer should be able to establish the relationships between the context-of-use that trigger these adaptations (they are inferred from user requirements) and the adaptation actions.

The combination of the different actions results in the cumulative action of each of them. It is possible that a particular context-of-use triggers a number of transformational actions, at the task model (structure and attribute) and concrete model (attribute only). The actions are executed in this order: structural action of the task model, then attribute change of the task model and finally attribute change of the concrete presentation model. In section 3.5.2.4, we will present more details on the process of specification and evaluation of adaptation rules to be used for implementation purposes.

2.8 Conclusion
### 2.8. Conclusion

We have also explained how the designer could establish the specifications of the context model in terms of atomic contextual parameters, and formulate adaptation rules by indicating when these parameters should trigger adaptation and how they impact on the user interface. Notably, by introducing the context and adaptation models, we also ensured that compatibility with existing notations used for task and presentation representations are maintained.

<table>
<thead>
<tr>
<th>Task</th>
<th>CIO</th>
<th>Structural operations</th>
<th>Behavioural operations</th>
<th>Presentation Style Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract node</td>
<td>presentation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>single_selection: radio_button</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>single_selection: drop_down_list</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>multiple_selection: checkbox</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>text.edit, numerical_edit: textfield</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>application</td>
<td>text: input.text, text_file</td>
<td>Add, remove, order task</td>
<td>Change task parameters and temporal operators</td>
<td></td>
</tr>
<tr>
<td>application</td>
<td>object: image</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>application</td>
<td>navigator: button</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>application</td>
<td>description: image, text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>application</td>
<td>navigator: text.link</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>application</td>
<td>navigator: image.link</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>application</td>
<td>activator: reset_button</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>application</td>
<td>activator: button_and_script</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.6: Structural, behavioural and style transformation operations per task type and corresponding CIO

notations given that they offer a number of benefits such as the ability of separating the context profile into sub-profiles enabling efficient exchange of context data with the UI adaptation server and making it easier to modify context elements.
Chapter 3

UML-Based User Interface Design

3.1 Introduction

Since its inception, UML has been mostly used in modelling business processes, domain models that represent business logic, and other types of object models internal to the operations of a system. The motivation of this effort has been mainly in creating a seamless software development process around UML. Our objective in using UML is primarily to model user's interaction with a system, create a uniform method for documenting user interface design and implement it in a way that is independent of the technologies used. In particular, we are interested in the visual representation used in UML, since it can provide an instant access to the details of the UI models and make it easier for developers to assimilate.

In this chapter, we present a UML profile that extends the standard UML notation to support the modelling of task model, context model and adaptation model. This approach complements the other methods proposed throughout this thesis and which uses other notations such as CTT and XML. We also review the different approaches that use UML to model interaction aspects in addition to context information. Finally, we present a methodology on how to convert a scenario that describes an application into the different abstract UI models, which are then used to generate the UI and enable its adaptation.

3.2 UML Notation

The Unified Modelling Language (UML) [48] has become the de facto standard notation for software modelling. In fact, UML is a family of diagrammatic languages used by system architects, software engineers, and software developers with tools for the analysis, design, and implementation of software-based systems as well as for modelling business processes.

The specifications of the notations have been coordinated through the Object Management Group (OMG). The first version UML 1.1 was released in later 1997, followed by several smaller revisions, principally UML 1.4.2. In later 2005, UML 2.0 [49] was introduced, and provided new specifications which can be split into two parts, in addition to a set of related specifications. The first part of UML 2.0 is the UML Infrastructure, which is used to create features considered to be required for
3.3. UML for User Interface and Context Modelling

UML at the metamodel level and it also indicates how to extend it. The second part is the UML Superstructure, which is used by most modellers to create models (i.e. various diagrams). The Object Constraint Language (OCL) [50] and XML Metadata Interchange (XMI) [51] are not part of the UML per se, but are important related specifications. The OCL is used to add extra details to UML models and specify constraints on one or more values of a model, while XMI determines the serialisation format used for the mainly graphical UML specification. A short summary of the main UML diagrams and related notation is included in Appendix B.

UML provides two extension mechanisms that modify the properties of a diagram, change its semantics and augment the notation with new building blocks to suit the needs of specific application domains. They can be applied at two levels:

1. **At the model level**: This approach uses stereotypes to extend the semantics of a default UML Element such as Class and Activity or restrict the way it is used. Stereotypes may introduce additional Values (which are the attributes of Element, also known as tagged values) and attach Constraints (conditions or restrictions on the attributes and associations of Element, and defined using OCL) with a new graphical representation.

2. **At the metamodel level**: This stipulates the definition of new UML Elements or adaptation of existing ones and change the default formalism of UML (in terms of syntax, rules and semantics), which can only be introduced at the metamodel level.

However, the more changes from the standard form of UML, the more issues there are with maintaining interoperability, especially when the meta-model changes. In addition, adding new constructs and semantics could make it harder for designers to comprehend their meaning and effectively use them. For these reasons, it is always important to weigh the benefits of extension against its risks, and carefully introduce extensions. In the next section, we will examine the use of UML in modelling different UI models (task and context models in particular), and also review the more comprehensive frameworks that cover more than one model.

### 3.3.1 UML for Task Modelling

One of the most important aspect in UI modelling is certainly task modelling, because designers view a system in terms of activities that users need to perform, how the system can support them, and how they should interact with the application. However, the major problem is that modelling techniques and notations used for tasks are performed in isolation. The resulting task model does not usually provide indication of the user interface structure, and it does not reveal interconnections between the dialogue components during the execution of tasks. One exception though, the CTT notation already supports the specification of references between tasks and presentation objects, including their corresponding classes and identifiers, but these relationships are not visually apparent, and UML modelling could help in highlighting these links and help in providing hints about its structure. However, using the standard UML is not recommended as the notation has not been designed to support the modelling of the aspects of user interfaces [53]. For this reason, there is a need to augment the notation to enable the support of task model concepts. From the literature review, we
have identified two main approaches to integrating the task model (and in particular CTT-based models) into UML:

1. Use existing UML constructs using profiling\(^1\) to represent elements and operations of the task model, as demonstrated by UMLi [20] and WISDOM [47], where specific UML stereotypes, tagged values (i.e. stereotypes' attributes) and constraints are used to represent properties and constraints of CTT elements such as task parameters, and whether they are optional or repetitive. However, this approach takes considerable time to specify and standardise, as different authors can select different base UML diagrams and differ in how to represent the structural and behavioural information contained in CTT.

2. Extend the UML meta-model, introduce a separate user task model and establish a mapping between CTT concepts (i.e. structure and semantics) and existing UML elements (e.g. Nobrega et al. [45]). This approach maintains a high fidelity in communicating task information as it integrates CTT notions into the UML notation. While this is technically feasible, the application of this method may compromise the comprehension and readability of the models by designers as diagrams can grow in size and complexity.

In the following sections, we will review the different approaches that use UML to model the task and context models, and present a number of examples of frameworks that model more than one UI model.

### 3.3.2 Key UML-based Frameworks

#### 3.3.2.1 WISDOM

Nunes [46] proposed a UML profile for the design and implementation of an adaptive system by introducing the presentation and dialogue dimensions to UML. The presentation dimension corresponds to the abstract presentation model in MBUID, and the dialogue dimension describes the atomic interactions between the user and the system.

For the presentation model, they define <<interaction space>> to model interaction between the systems and human actors. An interaction space class represents the space within the user interface where the user interacts with the functions and information needed for carrying out some particular tasks. The spaces are responsible for defining the output of the system, and how to handle events produced by the user. In the model, two associations are defined: <<navigate>> to move from one interaction space to another and <<contain>> for containment of spaces; two attributes (<<input>> and <<output>> elements), and <<action>> stereotype to denote change in the physical UI which is reflected in the internal state of the system. The dialogue model specifies the dialogue structure of the application using a UML based adaptation of the CTT notation. Temporal relationships are modelled as shown in Figure 3.1. Note that the icon circle with a sticky man and a stylized computer inside denotes a UML stereotype <<task>>. Also, all the associations between <<task>> classes are stereotyped <<refine task>> associations (sub-tasks). Finally {xor}, {sequence} and

\(^1\)This one of the extension mechanisms supported in UML. It allows designers to extend the vocabulary of UML so that new model elements (known as stereotypes) can be derived from existing ones, but that have specific properties that are suitable for a particular problem domain such as task modelling.
### Table 3.1: WISDOM class and association stereotypes

<table>
<thead>
<tr>
<th>Model</th>
<th>Class Stereotypes</th>
<th>Purpose</th>
<th>Association Stereotypes</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dialogue</td>
<td>&lt;&lt;task&gt;&gt;</td>
<td>Model dialogue between the user and the system</td>
<td>&lt;&lt;refine task&gt;&gt;</td>
<td>Specify that target task is at lower level of detail wrt. source task</td>
</tr>
<tr>
<td>Presentation</td>
<td>&lt;&lt;Interaction space&gt;&gt;</td>
<td>Model interaction between the system and human actors</td>
<td>&lt;&lt;navigate&gt;&gt;</td>
<td>Denote a user moving from one interaction space to another</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;&lt;contain&gt;&gt;</td>
<td>Denote that source class contains the target class</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;&lt;input element&gt;&gt;</td>
<td>Denote information received from the user</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;&lt;output element&gt;&gt;</td>
<td>Denote information presented to the user</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;&lt;action&gt;&gt;</td>
<td>User activity that result in significant change to internal state of the system</td>
</tr>
</tbody>
</table>

{deactivate} are UML constraints defined in OCL. Table 3.1 summarises the different class and association stereotypes defined for the presentation and dialogue models.

![Figure 3.1: UML notation for CTT in WISDOM ([46, , pp. 151])](image)

In addition, Nunes proposed three other models:

1. **User Role Model:** This model is represented by use case diagrams and shows the responsibilities of actors and the high level interactions of actors with use cases.

2. **Domain Model:** Use case diagrams to indicate the interactions between the different users and the system as represented by various internal components (by contrast to the user role model where it is shown how a user sees the system).
3. Analysis Model and Interaction Model: The former shows the interactions between the internal components of the system and the activities enabled by the system, and the latter focuses on how the user interacts with the different parts of the system.

WISDOM's general approach consists in starting from the use case structure of the scenario, then expanding it into the interaction model with interaction space and task classes, which accommodate the dialogue and guarantee the consistency between the interaction spaces (i.e. each interaction space is associated with a set of high level tasks). The task is then decomposed into sub-tasks including the relationships, while the presentation model indicates the presentation spaces and association stereotypes. Then, there is the internal analysis model where controls (used for processing), boundaries (used for interaction with external systems) and entities (used for information flow) are shown. However, from the demonstrators that have been presented, it seems that this approach presupposes a particular user interface technology and style, and strongly binds the UI to the logic of the application, making it less adaptable and reusable.

3.3.2.2 UMLi

Da Silva and Paton [20] presented an extension to UML to create a representation of the structure and behaviour of the user interface. The authors maintained the original UML constraints and semantics, and used class diagrams to represent the structure of the user interface (presentation models), and activity diagrams for its behaviour (task model). The authors have also developed a customised version of the ArgoUML\textsuperscript{2} toolset to support their notation with icons.

In their UML profile, interaction objects can be either abstract (e.g. action invoker, inputter) or concrete (physical widgets e.g. menu, check box). Instead of specifying the UI using concrete objects, they show containment and interaction between abstract object and associate them to a collection of widgets and visual styles. Their approach is different from standard MBUID since they set explicit relationships between the different models (e.g. application and interaction models, which are both modelled in UML), instead of relying on transformation and derivation to create one-to-one model mappings. UMLi introduces six interface constructors to model the structure of abstract presentation model, indicating the type of interaction implied and their containment relationships:

- **FreeContainers**: top level interaction level
- **Containers**: grouping of sub-constructors
- **Inputters**: receives information from users
- **Editors**: two way exchange of information
- **Displayers**: sends information to users
- **ActionInvokers**: received direct instructions from users

An example of a UI representation that use UMLi is shown in Figure 3.2 where they used a new diagram type that resembles the deployment diagram. It shows the specification of a dialog for searching books. It consists of a query form, which contains an area to specify book title, author or year of publication, an area that allows to start two kinds of searches, approximate or exact, and an

\textsuperscript{2}http://img.cs.man.ac.uk/uml
area that allows to search in the database or in the previous search results. The second area contains the search results and controls to select a book or search for other books. The third and final area contains OK and Cancel controls. One of the disadvantages noticed by the authors in using a UML extension is that the final UML model is larger than if it were described using only standard UML notion (up to four time larger).

![Figure 3.2: Example of presentation model in UML of a search book user interface ([19, pp. 72])](image)

### 3.3.2.3 Context-sensitive User Interface Profile (CUP)

As part of his work on Context-sensitive User Interface Profile (CUP), Van den Bergh [82] presented a dichotomy of context integration in the UI, which can be context dependent (i.e. specific to a given context) or context sensitive (i.e. adapts to changes in context). A further split is made between static UIs (context is consulted before the presentation of the UI is generated, but no further adaptation is possible) and dynamic UIs (changes in context are reflected into the UI when appropriate), as illustrated in Figure 3.3.

![Figure 3.3: Context integration in UI modelling according to CUP ([82])](image)
3.3. UML for User Interface and Context Modelling

CUP extends the UML metamodel in three areas: deployment diagram to describe the static structure of the UI (Figure 3.4(a)), class diagram to describe the context (Figure 3.4(b)) and activity diagram to specify the dynamic structure of the UI (Figure 3.4(c)). These diagrams are used to model the Task/Dialog Model, Abstract and Concrete Presentation Models, Context Model and Activity Model.

CUP was subsequently upgraded into the second version, known as CUP 2.0 [84], which integrates service aspects and enables the creation of interactive low-fidelity prototypes that can be used for evaluation. CUP 2.0 adds two new models by comparison to CUP, having five models in total:

1. **The application model** (new): It specifies the data structures and functionality that can be accessed through the UI. This includes the data structures and functionality that is used to provide relevant info (including context information) to the application. It is used by the system interaction model and the abstract user interface model to provide details of the data structures which are respectively used in the interaction with the modelled application.

2. **The system interaction model**: This corresponds to the user task model in CTT and the activity model in CUP. It is a hierarchical specification of the user's tasks, which uses flow based notation of the activity diagram instead of a tree-based notation, and supports all CTT's temporal operators, which are enhanced with context-awareness.

3. **The abstract user interface model**: It represents a UI structure that is shared between multiple contexts and on multiple platforms.

4. **The deployment model** (new): It provides information on how an abstract UI model is mapped onto a certain platform by linking the model's nodes to a specific context-of-use.

5. **The context model**: It specifies the different situations (or contexts-of-use) in which an application can be used, and links to the classes defined in the application model which provides context information.

CUP, and its successor CUP 2.0, provide a very complete framework for the use of UML to define various UI models. Details of how each model is modelled using UML profile in each version is provided in appendix C.1 and appendix C.2, respectively.
3.3.3 Model-specific UML-based Frameworks

3.3.3.1 Nobrega’s approach

Nobrega et al [45] proposed an extension to UML 2.0 activity diagram to fully support CTT’s concepts and provide an adapted graphical notation for an UML-like representation. It takes advantage of new UML 2.0 features such as the separation of state charts and activity diagram, which enables a better definition of temporal operators, without compromising the usability of the notation.

The key assumptions are that each task signals its own start and termination of execution. Hence the composed tasks should take into account the precedence of temporal operators. For instance, using CTT notation, $T_1|||T_2\triangleright T_3$ must evaluated as $(T_1|||T_2)\triangleright T_3$ and $T_1||(T_2\triangleright T_3)$. Consequently, they considered the following sequence of precedence: $>>$, $|>$, $|$, $|=|$, $|||$. They then mapped CTT task into an Action when it’s atomic, otherwise it’s a Call Behaviour Action. The complete listing of UML’s correspondences to CTT temporal relationships is shown in Table 3.2. However, using UML 2.0 notation to represent tasks may be impractical as simple task models can quickly grow in complexity. This is a well known problem with statechart-like notation. Although semantically correct, the UML mappings to CTT’s temporal operators will become counter-productive even for simple modelling activities. To illustrate this point, we present a depiction of the choice operator with CTT notation, WISDOM and Nobrega’s approach respectively, in Figure 3.5.

<table>
<thead>
<tr>
<th>Temporal relationships</th>
<th>CTT notation</th>
<th>UML 2.0 representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independent concurrency</td>
<td>$T_1</td>
<td></td>
</tr>
<tr>
<td>Choice</td>
<td>$T_1</td>
<td></td>
</tr>
<tr>
<td>Concurrency with information exchange</td>
<td>$T_1</td>
<td></td>
</tr>
<tr>
<td>Order Independence</td>
<td>$T_1</td>
<td>=</td>
</tr>
<tr>
<td>Deactivation</td>
<td>$T_1</td>
<td>&gt;</td>
</tr>
<tr>
<td>Enabling</td>
<td>$T_1</td>
<td>&gt;&gt;</td>
</tr>
<tr>
<td>Enabling with information passing</td>
<td>$T_1</td>
<td>&gt;&gt;</td>
</tr>
<tr>
<td>Suspend-Resume</td>
<td>$T_1</td>
<td>&gt;\cdot T_2$</td>
</tr>
<tr>
<td>Iteration</td>
<td>$T_1^\ast$</td>
<td>The start signal occurs at first execution of task T1 and a flow loop is created for task T1</td>
</tr>
<tr>
<td>Finite Iteration ($T_1(n)$)</td>
<td>$T_1(n)$</td>
<td>Use of counter which triggers Finish T1 signal after n occurrences</td>
</tr>
</tbody>
</table>

Table 3.2: UML 2.0 notation mappings for task temporal relationship according to Nobrega [45]

\(^{3}\)A call action is an abstract class for actions that invoke behaviour and receive return values
3.3. UML for User Interface and Context Modelling

3.3.3.2 LEAN CUISINE+

Scogings and Phillips [68] presented some enhancements to UML called Lean Cuisine+, which is capable of modelling tasks within the dialogue structure of the user interface, by contrast with UML which proposes two separate diagrams: one for the dialog and another for the tasks.

Lean Cuisine+ has been developed to enable tasks sequences to be represented in the context of the structure of the interface dialogue. The interface is represented as a dialogue tree, and its behaviour is expressed in terms of the constraints and dependencies. Its components are called menemes which are objects or actions available in the UI, with two possible states: “selected” and “not selected”. An example of a UI for a library catalogue is shown in Figure 3.6. Some menemes act as header of sub-dialogues, and a virtual meneme is where the name appears between braces (which indicates that it is not available for selection). Selection Triggers (shown as an arrow) are indicative of a binding between two menemes during a selection of one of them. Conditions can be added to the diagram in between brackets after the header. A monostable meneme (⊥) can be selected by the user, then it reverts back to unselected state on completion of the task. A passive meneme (⊙) cannot be selected or unselected by the user (and only by the system).

The task sequence is then superimposed on the dialogue diagram, which links up menemes with solid arrows (for user action) and dashed arrows (for system action). In practice, the designer starts with the use cases and class diagrams of the application in UML. Then the task sequence is generated for each use case, and are then modelled with Lean Cuisine+ to finally generate a matching user interface.
3.3. UML for User Interface and Context Modelling

3.3.3 XIS

Da Silva et al. [18] presented a use case-driven approach UI design framework called XIS (eXtreme modelling Interactive Systems), which was based on the MVC pattern, but had flaws related to its complexity and inability to specify how generated UI should look like. XIS presents a UML profile and provides a set of guidelines and model-to-model transformation templates for designers, and a model-to-code transformation templates to produce software documents (e.g. source code) from the models for programmers. XIS exploits abstraction and isolation to enable the separation of concerns between the functional and non-functional aspects of an application. For this, XIS provides multiple views and minimize inter-view dependencies.

Currently, there are three main views (or concerns) that are captured: entities, use-cases and user-interfaces views, as shown in Figure 3.7. The Entities View is first specified and consists of the domain view (classes and relationships that correspond to the problem domain) and the business view (business entities with higher level of granularity). The Use-Cases View is used to define actors (or roles) and establish the corresponding permissions. This encompasses the Actors View (specifies the entities that can perform operations) and UseCases View (relationships between the actors defined in the Actors View and the operations they are allowed to perform over the business entities). Finally, the User-Interface View is used to define the interactions spaces (i.e. abstract screens the receive and present information to end users during their interaction with the system) and the navigation flow between them. It consists of the NavigationSpace View (define the navigation flow that can occur between any of the interaction spaces) and the InteractionSpace View (defines the UI interaction elements that are contained in each interaction space, it also specifies access control between actors and UI elements).
3.3. UML for User Interface and Context Modelling

3.3.4 UML for Context Modelling

3.3.4.1 ContextUML

Sheng and Benatallah [71] proposed an UML-based techniques to design context-aware services for model-based development of web services to generalise context provisioning and formalise context awareness mechanisms and their usage in Context-Aware Services (CAS’s). The meta model of ContextUML is depicted in Figure 3.8.

At the top, we find the Context class used to model context information. It has two sub-types: AtomicContext, a low-level context that does not rely on other contexts and can be provided by context sources, and CompositeContext, which aggregates multiple context elements, either atomic or composite. ContextSource models the resources from which context is retrieved. It has two categories, ContextService, which is provided by an autonomous organisation collecting and refining
context information, and ContextServiceCommunity, which aggregates multiple context services, with a unified interface, so that when invoked it selects the most appropriate context service. The selection of the context service is based on a multi-criteria utility function and the criteria is a set of QoC (Quality of Context) parameters which have the following parameters: precision, correctnessProbability and refreshRate.

Context awareness is formalised using the CAMechanism class, with two sub-types: ContextBinding and ContextTriggering. Mechanisms are assigned to context-aware object CAObject by the relation MechanismAssignment. CAObject has four sub-types: Service, Operation, Message and Part. Each service offers one or more operations and each operation belongs to exactly one service. Each operation may have one input and/or one output messages. Furthermore, ContextBinding is a subtype of CAMechanism that models the automatic binding of context and context-aware object. ContextTriggering models the situation of contextual adaptation where services can be executed or modified based on context information. It contains a set of context constraints and a set of actions which are executed if constraints are evaluated to be true. A constraint is modelled as a predicate with one operator and two or more operands. The issue with ContextUML is that it focuses on modelling context for web services, in addition to the fact that the authors provide a heavyweight notation by modifying the metaclasses, which means that it cannot be used by standard UML tools.

3.3.4.2 Simple Mobile Services (SMS)

Broll et al. [9] presented a UML-based modelling of the context as part of Simple Mobile Services (SMS) project which aimed to simplify the creation of context-aware mobile services. In SMS there are two levels of service authoring which differ in their level of expressiveness and representation: one is directed to the expert programmer and uses UML profile, and the other one is directed to end users with minimal technical expertise by providing templates that can be used to compose a service out of service elements.

At high level modelling, UML component services can be composed using UML Activity diagram, where each activity corresponds to a component operation. The UML model is then supposed to be converted into a programming, scripting or markup language for the implementation of the UI. It resembles ContextUML’s approach in that it uses two dimensions of quantity and quality, which can be described by two pairs of terms: atomic vs. composite and low-level vs. high-level.

In SMS, context is modelled in two distinct phases (Figure 3.9), starting from the abstract definition of a generic, high level information model of key entities (at the model level M1) where it defines a model of classes and subclasses of context information used for an application/service on an abstract level, and proceeding to its translation to a concrete context model (at the instance level M0), where the values of the different parameters are associated to concrete instances of context information, and context information can then be transformed to implementation-level language e.g. XML. At the very top, there is the meta level (M2), where ContextUML’s classes are used e.g. Context, AtomicContext and CompositeContext, and adds its own class for describing quality of context parameters, either low-level or high-level context, and the meta-model can be further extended to include other aspects of context.
3.3. UML for User Interface and Context Modelling

3.3.4.3 Ayed and Berbers’ Approach

Ayed and Berbers [4] proposed a general context model for context-aware mobile applications, where UML is used to specify contexts that impact on the application as well as its variability in structure, behaviour and architecture in response to change of context. Although the focus has been put on separating context management and processing concerns from the development of the application, they introduced several dependencies on the application that reduces its applicability.

The following UML class stereotypes have been introduced to model context information (Figure 3.10a):

- <<Context>>, which describes the context type e.g. location of the user, network bandwidth and user preferences, as well as their sources.
- <<CollectionProcess>>, which represents the elements necessary to collect the context. It defines the common tags of two types of context collections: event-based collection (<<EventCollection>>) and periodic collection (<<PeriodicCollection>>). The event tag indicates the condition that must be satisfied by the context before a new context value is returned. The period tag indicates the rate at which context data should be collected.
- <<ContextQuality>>, which represents the quality attributes that must be satisfied by the context such as the accuracy, the precision, the correctness, and the level of trust.
- <<ContextState>>, which is used to specify the context states that have a specific impact on a given application. It is constructed by combining elementary context states (defined with <<ElementaryContextState>>) with logical operators (<<AND>> and <<OR>>).

The effect of the context on the applications is modelled using an extended sequence diagram to represent three types of adaptation (Figure 3.10b):
1. Structural adaptation, which consists in extending the objects' structure e.g. adding or deleting methods or attributes to the objects. For this, `<VariableStructure>` abstract class was defined, as well as a set of `<version>` subclasses which represent its different versions. `<contextStateIds>` are set of identifiers of context states specified by the designer which are associated with the change. All subclasses should have one and only ancestor from `<VariableStructure>`. If several context states associated with different versions are satisfied, a new subclass that represent an OR of the attributes and operations is dynamically defined.

2. Behavioural adaptation which concerns the adaptation of behaviour of the applications' objects. For this, they defined `<VariableSequence>` sequence diagram stereotype which is associated with the interactions that are variable depending on the context. Each interaction variant is represented with `<SequenceVariant>` and tagged with `<contextStateIds>`. The `<SequenceVariant>` should be enclosed in one and only one interaction stereotyped with `<VariableSequence>`.

3. Architectural adaptation consists in adding and deleting objects according to context. UML class diagram is extended with the `<Optional>` stereotype. Similarly, each optional object is associated with `<contextStateIds>` which indicate when it is instantiated.
3.3.4.4 The Context Modeling Profile (CMP)

Simons [72] proposed a lightweight UML extension for context models in mobile distributed systems, and in particular for the sharing of context information among users. The resulting models visualise meta information of the context i.e. source and validity of context information (i.e. how frequently the context changes) and reflect privacy restrictions (Figure 3.11).

The context model provides type definitions for small autonomous context items. The model consists of small classes each representing a context item type. The end names of the associations are used to access linked context items. When exchanging context items, only the atomic item is transferred. Links between the items are mostly time variant, and their validity is specified as a property of the association.

The source of the context item is a measure of quality and context information that is generated by the users is generally considered more reliable than sensors date. Context information can be derived from other context items too. Interestingly, privacy is used for some context information, and several access rights are specified to reinforce it. The latter can be added to the class model using comments, and derivation rules can be specified by adding constraints to model elements, and derived context items (like derived activity of a person) can be notated using a preceding “/”. However, comments may be misused, resulting in models that can be potentially invalid.

![Figure 3.11: CMP UML Profile (72)](image)

3.3.5 Discussion

It is clear that the use of UML for modelling UI models and integrating them into a full-fledged user interface design and generation framework has only been recently explored by researchers and practitioners. We also find that the desktop and web platforms have been the primary targets of these approaches, and that mobile platform is only implicitly supported. This also highlights the difficulty of using software design-centred notations, such as UML, to specify the different models used in mobile user interface design. Despite the immaturity of mobile application development tools, UML has a key role in current and future development tools to facilitate the process of developing mobile applications.

In comparing between the different approaches, we notice that Nobrega’s approach [45] is relatively exhaustive in modelling the task model using activity diagram but its complexity makes it less usable.
3.3. UML for User Interface and Context Modelling

than the other lighter UML extensions albeit they may sacrifice compatibility with standard UML tools. This is the case for UML+ [20], where they extended activity diagrams and used object flows to model UI behaviour. The introduction of new diagrams to indicate nesting and containment breaks compatibility with standard UML, so a specialised editing tool needs to be used. In WISDOM, Nuns and Cunha [47] used case and class diagrams to define semantical equivalent of CTT notation in UML 1.x. Despite the existence of a presentation model, the latter has no relation with the way UI's are constructed at the end, which places limitations on its adoption. More importantly, the three above-mentioned approaches have no support for modelling context-sensitive UI's.

To model the relationships between the different classes of presentation objects, UML+ defines a new diagram-type similar to a UML deployment diagram. CUP uses stereotyped deployment diagrams to define the presentation structure, whereas WISDOM uses stereotypes class diagrams to highlight the different types of components that the user interacts with, such as input and output. In the second version, CUP 2.0, deployment diagram is replaced with class diagram.

XIS is similar to UML+ to some extent in the Interaction Space View, and to WISDOM in terms of representing the navigation aspects, though the latter does not represent each node of the abstract UI. In addition, XIS considers the trade-off between simplicity (i.e. keeping models at a very high-level in terms of business model, roles and presentation views) and productivity (enabling the specification of model-transformation templates to generate functional UI's). Regarding Lean Cuisine+, the fact that the application is the starting point for the UI task model, means the model is confined to showing the GUI widgets associated with the input and output of the application, which is limited to system and user tasks. And those tasks that do not have a visual representation on the GUI are not included in the task model. In addition, the lack of a presentation model make it more difficult to specify a mapping to abstract and concrete interactor objects.

Looking more closely at CUP, we remark that the abstract presentation model employs UML to show the structure of the UI visually, with additional information about the UI components such as data types used and description of functionality. It also uses deployment model with a set of stereotypes to indicate input, output, group and action executed by the user. Meta information is specified as attributes of the node, data type and class manipulated by UI is specified by an association, whereas the type of relation (select, trigger, interact) are specified by labelling it. The abstract connectors are mapped to multiple concrete instances using artefacts (libraries, executables, etc.). On the other hand, CUP 2.0 has a slightly different objective in that it was meant to be used for (semi-)automatic creation of interactive low-fidelity prototypes that can be used for evaluation purposes. For this, it supported additional stereotypes for associations between abstract UI components to express relationships other than containment, to indicate constraints on the structure of the UI. In this regard, CUP 2.0 represents the most exhaustive and ambitious approach to date to enable model-based UI development, although the way it supports context adaptation is limited to design stage and does not extend to implementation.

Compared to UI modelling, UML has been more widely used to model context information and its impact on service/application design. This is likely due to the fact that context adaptation has become an essential dimension to cater for in software design. From the representative sample of approaches reviewed above, we notice that there are as many UML profiles presented as there are application domains. ContextUML has been specifically designed for context-aware service delivery, and offers a hierarchical decomposition of context information, in addition to the support for Quality of Context. The notation is generic enough that it can be adapted to user interface domain, for
instance, by considering UI generation as a service in a distributed system. However, its context adaptation stereotypes do not offer a sufficient level of granularity that it can be used for the execution and modifications of UI models for instance. SMS, which is related to ContextUML but starts at a higher level of abstraction, simplifies the modelling of context sources and adds support for quality of context. However, it suffers from the same limitations as far as the integration of context information into UI models is concerned. Ayed and Berbers' approach is fairly exhaustive in the sense that it supports structural, architectural and behavioural adaptations of service design based on the context of use. However, it is more geared towards helping to show how context information can have an effect on the design of a context-ware middleware (presumably, capable of runtime adaptation, using aspect oriented programming (AOP) for instance), and not so much for storing context data for use at runtime. It achieves that by associating different versions of a software class with context states. Finally, as for CMP, we believe that it is limited in scope as it focuses only on how to communicate context data between different users. For this, it stores the source of context, its change frequency, its quality and who has the right to access it. The lack of hierarchical structure of context information and support for adaptation make it unsuitable for model-driven UI generation and adaptation.

3.4 Proposed Approach

3.4.1 Introduction

Our goal has been to support the efforts to create user interfaces and describe their usage situation by modelling the interaction between the mobile user and the applications at different levels of abstraction. UML is a suitable standard notation for this task since it ensures easy integration with existing modelling practices, tools and workflows used in software development, so that some of these models can be combined with other UML based models used for application development. It is also suitable to describe the design concepts as well as implementation and deployment. In addition, since UML is a visual modelling language that uses abstract concepts, metaphors and diagrammatic notations, it is ideal for software developers, UI designers and end users to communicate ideas and concepts. As for the approach to extending UML, we opted for creating a UML profile, since it maintains compatibility with existing UML modelling tools, while being extendible if needs be, and it better matches the expectation of existing UML modellers.

UML is particularly suitable to represent structural information and relations (e.g. containment, composition) because it adds a visual dimension, making it easier to comprehend. However, it is less effective in showing UI's behaviour as diagrams can rapidly grow in size and complexity. For these reasons, our approach consists in using only one structural diagram, namely Class diagram to indicate how entities relate to each other, and one behavioural diagram, namely Use Case diagram to present a graphical overview of the functionality provided by the user interface prior to the elaboration of the task model.

Note that the presented UML profile complements the TERESA [40] standard model-driven approach in the sense that UML models are only produced for the models newly introduced, namely Context and Adaptation models to specify data used at runtime, and to enhance existing ones such as the Task model. We believe that there is no need for UML modelling for Abstract and Concrete presentation models in our case because they are automatically generated in TERESA and developers have little influence on the mapping between task nodes and presentation interactor objects. Our primary
objective has been to keep the UML profile relatively simple to use by both user interface designers and software engineers since they need to collaborate in order to create the interactive application and its UI.

### 3.4.2 Modelling Tool

To support the work of UI designers and ensure the adoption of a new UML-based representation, it is important to provide the potential users with a software tool that can be used to view, edit, and create these models using suitable visual icons and supporting basic operation found in other UML editing tools. The tool needs to exhibit the following features:

- Support for UML 2.0 notation.
- Ability to serialise the UML models to XMI to inter-operate with other software CASE tools.
- Support the specification of UML profiles.
- Be incorporated within a CASE toolset that is used for the modelling of software, to enable the UI designers and software developer to communicate and integrate their perspectives on UI design.

The support for UML 2.0 is needed because it is the version that is most widely used nowadays, and our models rely on the new additions introduced since UML 1.x e.g. profiling. The XMI support ensures that models created with this tool can be opened and edited by other UML editing tools. In practise, different tools have incompatible XMI implementations, but this is the best that can currently be achieved in terms of inter-operability. XMI can also be transformed into source code or concrete UI presentation models (which are generally XML-based) using external tools, although this is not considered in our approach since we rely on TERESA’s notations and processes to transform models. The support for UML profile is essential because we rely on this technique to extend UML. The tool should also be integrated in a toolset that support the traditional use of UML for software design.

After a review of current free CASE tools, the list of possible options was narrowed to three: ArgoUML⁴ (used in [46]), MagicDraw⁵ (used in [84]) and Poseidon⁶ (which is based on an earlier version of ArgoUML). The three UML tools are compared in the table below.

From the table above, MagicDraw seems to provide the most suitable set of features that match our requirements. It serialises models to XMI 2.1, allows the creation of UML profiles, and offers easy creation of custom diagrams and addition of new symbols. There is a free and and commercial version of MagicDraw with different features and limitations (such as the number of diagrams that can be used for a single model). Nonetheless, the free version was deemed capable enough to be used for this work. MagicDraw also shows tagged values within the diagrams for all stereotypes, which increases the visibility of the information. A screenshot of the application is provided below.

---

⁴[http://argouml.tigris.org/]
⁵[http://www.magicdraw.com/]
3.4. Proposed Approach

<table>
<thead>
<tr>
<th>Features</th>
<th>ArgoUML</th>
<th>MagicDraw</th>
<th>Poseidon</th>
</tr>
</thead>
<tbody>
<tr>
<td>License</td>
<td>Open source</td>
<td>Proprietary: free or commercial</td>
<td>Proprietary: free or commercial</td>
</tr>
<tr>
<td>UML support</td>
<td>UML 1.4</td>
<td>UML 2</td>
<td>UML 2</td>
</tr>
<tr>
<td>XMI support</td>
<td>XMI 1.2</td>
<td>XMI 1.2</td>
<td></td>
</tr>
<tr>
<td>OCL support</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Extension support</td>
<td>No easy way to customise the menu and models, need access to source code</td>
<td>Some diagrams are limited to 20-25, customise UML diagrams, insert actions into menus, Java plugin, support model transformations (in paid versions).</td>
<td>Not possible to add new symbols</td>
</tr>
<tr>
<td>Code generation</td>
<td>C++, C#, Java, etc.</td>
<td>No</td>
<td>Java</td>
</tr>
</tbody>
</table>

Table 3.3: Comparison of UML editing tools

Figure 3.12: MagicDraw main window

We define a number of stereotypes that extend the existing UML Class metaclass. Using MagicDraw we have created a UML profile that incorporates all the required stereotypes for each of the models under consideration:

- The task model
- The context model
- The adaptation model

By creating these visual models, it is possible for these UML models to be exported into other CASE tools. In the following sections will provide more details on each model and their composition, supported with illustrations created with MagicDraw.
3.4.3 Task Model

One way to support the design of interactive applications is to describe the different actions carried out by the user (be them physical or cognitive in nature) and the system (e.g. background processes), which can form the basis for the analysis of user tasks and the evaluation of their performance. Task-based approaches have been precisely developed to achieve that, although they do not give specific guidelines for the development of user interfaces.

In some respect, UML shares similar objectives with task-oriented approaches, such as CTT, in the sense that they both describe activities and objects, and they only differ in the focus and the notations used to represent them. Task-based approaches first identify activities and then the objects that they have to manipulate. UML follow an inverse process as it focuses on modelling the objects composing the system. At the same time, when using UML alone to specify the interactive part of a computational system, the resulting model tends to ignore some specific aspects related to user interaction.

Through their differences, we believe that they are complementary to offer a complete solution. While CTT is more suitable to design user-oriented interactive applications since it offers an effective and efficient way of supporting user activities, UML can help in providing the design of the software objects through which interaction is enabled. We propose to integrate the two approaches to offer a complete solution that tackles the issues related to the analysis of the user tasks, and cater for the development of user interfaces. The need for a new approach is further highlighted by the fact that CTT helps specify user and system interactive tasks without having to deal with the low-levels details, while relying on UML to provide details of how to link with the UI implementation aspect. Ultimately, the combination of CTT and UML notations and concepts allow different stakeholders in the development process to take part in the design phase, not just by understanding models but also contributing to them.

As we have seen in the previous section, existing attempts to integrating the two approaches have centred around trying to represent task models completely using both ConcurTaskTree and UML, exploiting the extensibility mechanisms built into UML such as profiling. However, supporting both the structural and behavioural characteristics of the model made the result to be cumbersome, and takes considerable time to specify, while not all temporal operations may be representable in UML. Our argument is that while a notation can exploit the expressive power of UML (e.g. UMLi and WISDOM), the resulting diagrams might not be the most effective way to support designers in their work. We believe that it is important not to complicate the model’s representation in order to keep them useful for the design process and comprehensible by software engineers.

As a result, we opted to only represent the structural aspects and properties of the task model using Class diagram. This is to avoid completely replacing the CTT notation with its UML equivalent, since the task model is an integral part of our UI design, generation and adaptation framework. To further simplify the UML profile, we have used a simplified representation of CTT notation in UML, yet it is compatible, as shown in Figure 3.13.

*We have chosen to reduce the number of stereotypes used compared to the elements defined in the original CTT schema, yet being complete and consistent. We have also made visible all possible values for stereotype attributes (i.e. tagged values) to guide UI designers.*
Based on the simplified CTT notation, we defined a number of corresponding class stereotypes, of which the most important are listed below.

**Stereotypes**

- **TaskModel**: A container for a set of instances of the Task stereotype.
- **Task**: It represents a task node with a number of attributes, such as category, description, etc. A task instance is associated with a set of instances of the SubTask stereotype. A Task stereotype is also associated with a Parent, SiblingRight and SiblingLeft.
- **Object**: It defines the type of abstract input/output interactors and transitions to which the task is mapped at the presentation level, using the InputAction and OutputAction stereotypes.

To illustrate how such a profile can be used to model tasks, we present an example of a very simple task subtree. Figure 3.14a depicts the subtree represented with CTT notation (using TERESA editor), and Figure 3.14b shows the same model using our UML profile. As it can be seen, the advantage of UML is that it provides instant access to all the properties of the task and increases the visibility of information, whereas in TERESA the designer needs to double-click on each node to view its attributes. However, the use of UML results in a relatively cumbersome and complex representation whereby task types and relationships (e.g., hierarchy and temporal relationships between tasks) are shown as attributes textually. By comparison, CTT is based on a hierarchical structure and uses appropriately designed icons. It reflects the logical approach that would be used by most designers, allowing the description of a rich set of possibilities that is both highly declarative and generates compact description. For this reason, we will maintain CTT as the primary modeling notation for task models (as part of our model-based approach). In addition, we supplement the CTT-based
3.4. Proposed Approach

model with system behaviour model using a use case diagram to create complementarity between the two notations, as it will be shown in section 3.5.2.1, where we discuss mobile UI development cycle. We believe that use cases could be useful in identifying the tasks to perform by the different 'actors' and extracting related requirements. It is then possible to switch to CTT notation to model those tasks and their temporal interdependencies. In addition, CTT notation is supported with authoring tools (e.g. CTTE [39], TERESA [40]) which further facilitates the design, simulation and prototyping of UI's.

Nonetheless, UML notation remains useful in the sense that it could be used by UI designers who opt to use CTT semantics and syntax as part of another model-transformation framework. For instance, it could be integrated into existing UI tools that can convert UML models into prototypes which can be more flexible and customisable than with CTT editing tools, and offering a wider choice of
3.4. Proposed Approach

widgets.

3.4.4 Context Model

As we have seen from above, UML has only recently been used to model context information as part of an overall model-driven UI design and generation process, such as CUP 2.0 and WISDOM. Indulska et al. [31] gave a rather negative evaluation of UML as notation language for context modelling in comparison to Ontology-based and Object-Oriented models. However, we believe that UML can be adapted to support context modelling using a lightweight profile. The rationale behind our approach is outlined below:

- UML can be used to derive descriptions of context, which can be manipulated and processed by standard UML tools.
- The structured nature of UML, allows the modelling of all context information for an application on different levels of abstraction from meta to instance level.
- UML class instances that represent context elements can have relationships, attributes and constraints.
- While UML model is rather conceptual, it can be used as a template or blueprint from which to derive descriptions of context using other languages that are closer to the actual implementations of applications and services such as XML. In addition, it can be easily assimilated by system engineers as well as UI designers.
- It allows to further extend and include other aspect of context, meet the requirements of new application domains, and support additional context properties.
- The use of UML can help in implementing a suitable context management software and formally represent the objects and data flows related to the context information.

Based on the context categories 2.7.8.3 and context properties 2.7.8.4, we define a number of corresponding class stereotypes, attributes and relationships (Table 3.4). The assumptions made is that each combination of context elements represent a unique context-of-use, that context information representation is independent from implementation considerations for how it has been captured and analysed, and finally that context information is free from ambiguities which require data conversion and normalisation.

Since the context can be described depending on its levels of granularity, it is therefore possible to represent context as a hierarchical structure where higher-level context (complex) can be deduced from more basic context components (simple). Furthermore, the nature of a complex context is considered dynamic if any one of the constituent simple context element is dynamic. A Context is a class that models the overall context information. It has two sub-classes: SimpleContext, which represents a low-level context information, with one parameter and value, and ComplexContext, which aggregates multiple instances of context, either simple or complex. A ContextProfile aggregates multiple ComplexContext's which share the same characteristic (e.g. device, user, environment). The UML profile is shown in Figure 3.15.
3.4. Proposed Approach

<table>
<thead>
<tr>
<th>Stereotypes</th>
<th>Attributes</th>
<th>Type (values)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentUserInterfaceContext</td>
<td>timestamp</td>
<td>Date</td>
</tr>
<tr>
<td>ContextProfile</td>
<td>type</td>
<td>ContextSource (platform, user, environment)</td>
</tr>
<tr>
<td>SimpleContext</td>
<td>name</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>value</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>nature</td>
<td>ContextNature (static, dynamic)</td>
</tr>
<tr>
<td></td>
<td>impactLevel</td>
<td>ContextImpact (task presentation)</td>
</tr>
<tr>
<td>ComplexContext</td>
<td>name</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>nature</td>
<td>Implied from nature parameters of its context constituents</td>
</tr>
</tbody>
</table>

Table 3.4: Current User Interface Context UML Profile

While UML is an abstract language that provides a quick overview of the structure and content of the context profile, this representation is mapped to XML to be used as data input to the UI transformation engine. This ensures a clear distinction between the modelling activity (carried out by UI modellers and designers) and the implementation activity. The mapping has been shown in section 2.7.8.5 along with an example of current context profile.

3.4.5 Context Adaptation Model

Given the lack of a UML notation that is specifically designed to support adaptation concerns for a model-driven UI design approach, we proposed to create a notation that corresponds to the adaptation mechanisms introduced in section 2.7.9. The initial approach consisted in modifying the CTT notation so that choice nodes can be supported. The new stereotypes added to the task model are illustrated in the UML model shown in Figure 3.16 in red. The choice node has a condition part and two sub-tasks. Conditions are constructed by comparing a context element to a threshold value using an arithmetic operator (i.e. <, >, =, !=). At its most basic form, a condition is made up of a single context parameter and a value, which is compared to the current value extracted from the context profile. This structure allows the local evaluation of the context and depending on the result, the right or the left child node is selected to replace the choice node.
In the second iteration, there is a standalone adaptation model which comprises a number of adaptation rules. The latter is composed of a contextual condition and one or more actions. Rule specifications determine the actions to be taken when certain contexts are met on the affected models which are specified at design time. The list of affected models is compared with the UI models (task or presentation) associated with the matching atomic or composite context element from the context profile. This check ensures that the change in context is permitted (by the context model.
3.4. Proposed Approach

designer) to trigger the adaptation actions (which could have been specified by another person). In addition to the ability to compose contextual conditions using arithmetic operators, it is possible to create more complex conditions by nesting a number of simple conditions using the logical operators (and, or). The action part describes what transformations to be applied onto a list of objects, representing either task nodes, or presentation nodes and referred to with their ID's. The action specifies the level(s) affected (task and/or presentation), the nature of adaptation applied on them (attribute or structure), the type of operation (sort, add and delete), and a list of objects' parameters that need to altered.

Contextual conditions contain an expression comparing context parameters with given values using an arithmetic operator. Two types of actions can be specified along with operations (e.g. add, remove, etc.). The priority of an action is also specified, to force their execution in a particular order. If no priority is specified then they are executed in the order they appear in the serialised version of the adaptation model. Note the details of how conditions are evaluated and actions executed is explained in section 4.3.7 when we present the serialised version of the adaptation model. Also, given the similarities between the context model and the action part of an adaptation rule, the UML model re-uses two stereotypes found in the context model, namely the SimpleContext and ComplexContext. The full list of stereotypes and their attributes is shown below. The associated UML profile is shown in Figure 3.17.

Stereotypes

- AdaptationRules: A container for a set of instances of Rule class stereotype.
- Rule: A container for a set of instances of Condition and Action stereotypes.
- Condition: A container for set of instances of ComplexContext. A Condition instance can recursively contain a set of conditions.
- Action: A container for a set of instances of Object, which identifies task id's or presentation units.
- SimpleContext: Similar to the stereotype defined in the context model.
- ComplexContext: Similar to the stereotype defined in the context model.
- Object: A stereotype which holds the ID's of the task nodes that are affected by the adaptation actions.

Attributes (for Rule)

- priority: It indicates the orders in which different rules are executed.

Attributes (for Condition)

- type: It indicates what operator is used to evaluate the current context with respect to the condition specified, and how nested conditions should be combined. Two logical operators (AND, OR), and three comparison operators (==, !=, <, >) are supported.

Note that rule priority has only been introduced in the model specification, but it not taken in consideration in the adaptation process implemented in our work. The default mode is that rules are executed sequentially as they appear in the adaptation model.
Attributes (for Action)

- **level**: It indicates the model on which adaptation is applied i.e. task or presentation model.

- **change**: It indicates what type of modifications are required, be it structural or at the attribute level.

- **operation**: It indicates what operation is applied on the task or presentation representation i.e. sort, add or delete one node or multiple nodes.

- **param**: It lists the (task or presentation) nodes that need to be modified and/or the attributes that are affected by the adaptation rules.

![Diagram showing Adaptation Rules Profile with attributes and operations](image-url)

Figure 3.17: Context Adaptation Model UML Profile
3.5 Mobile User Interface Development Process

3.5.1 Introduction

The standard approach to developing a stationary application is to start with gathering requirements, lay out an architecture and design, select tools to help implement the application, develop the application, test, and finally deliver it. Since we are developing for the mobile scenario, the dimensions of mobility and the mobile conditions of the user make the process of developing mobile applications different from the process of developing stationary applications. Traditional approaches cannot be used directly because of the assumptions made with regard to the stationary use cases, and hence, they need to be modified to accommodate the development of mobile applications. This distinction is also present in the development of user interfaces.

In the following sections, we will outline one suggested method on how to build mobile user interfaces that integrates the dimensions of mobility and adapt to the usage conditions, using a combination of UML and UI model-based techniques for the development process. We need to keep in my mind that this approach is rather a set of recommendations to point out a roadmap to building mobile UI’s. It does not exclude other methodologies which may be used to develop the other parts of the mobile application such as the back-end and business logic, as long as the principles explained here can be integrated into the overall development methodology. Our goal is primarily to augment existing methodologies to address the dimensions of mobility.

Figure 3.18 depicts the models under consideration and how the various tools can exploit them in the development process. There are tools that help in developing the models (modelling tools), others that aim to analyse their content (analysis tools), and others more that use them in order to generate the user interface, the context of use and adaptation rules (development tools). Note that tool in this context refers to the set of notations, procedures or software packages used to achieve our goal of modelling, analysis and development of the UI.

![Diagram](Figure 3.18: Models and related tools in the user interface development process)
3.5.2 Software Development Cycle for Mobile User Interfaces

3.5.2.1 Mobile use cases

Mobile applications are considered to be a superset of their stationary counterparts, so we need to introduce additional use cases as necessary. To illustrate our approach on how to design a mobile user interfaces with mobility aspects, we use the example of a touristic guide application that delivers information about the town’s main attractions. The initial step involves non-technical participants such as users, UI designers, etc. describing the scenarios in an informal fashion. The next step is for the software designers to translate those scenarios into UML-based use cases, with more details and formalism, but they remain visually readable by the different participants. The early material could also include documentation concerning existing applications, notes from meetings with users, requirements provided by users, etc. This material is refined to identify the task structure underlying the new application to design.

Scenarios are a well-known technique in HCI often used during the initial informal analysis phase. They provide informal descriptions of a specific use in a specific context of an application. Let us assume the following use cases have been defined after an analysis of the scenario:

- **Use Case 1**: User looks for interesting parks in her vicinity, requests directions and info.
- **Use Case 2**: User looks for shops in her vicinity, requests directions and info.
- **Use Case 3**: User look for a particular address, enters destination, requests directions.
- **Use Case 4**: User wants to make a booking for a hotel, selects a hotel, makes a booking.
- **Use Case 5**: User asks for bus map of the town.

At this point, we do not really know anything about the mobile users of the application or the consequences of the application being usable by mobile users. For each of the use cases we need to associate additional contextual information that surrounds the execution of these use cases and the adaptation that should occur on the user interface to reflect this change of context-of-use and could offer additional benefits to the user. Possible mobility dimensions that could affect or improve how the tourist guide application is used include:

- Location which is automatically detected, thereby avoiding the data entry for the user’s current address.
- If the user crosses an area of degraded or erratic network conditions, the graphical map is disabled (i.e. no display of points of interests) and instead a textual listing is provided.
- If the user crosses an area where there is an continual disconnection from the network, the functionality to book a hotel is disabled for security reasons.
- Notifications about shops in the vicinity that match the user’s interests (or shopping patterns) can be enabled/disabled according to the user’s personal preferences .

It is assumed that the infrastructure for the mobile application provides reasonable level of functionality such as location information, network condition, etc. While this assumption may not hold true at present for all mobile devices, however, the rapid evolution of technology is likely to integrate such functionality in the future. At this stage, it is important to ask further questions regarding the use of the application in a mobile setting:
3.5. Mobile User Interface Development Process

- Has the user considered all of the scenarios under which the system may be used by a mobile user (e.g. connected, degraded connectivity, change of device)?
- What is the impact of a change in device? chance of a user profile?
- Are there any scenarios for notifications that the user has not considered?

These kind of questions help bringing up answers which can be used to revisit the list of use cases and come up with additional ones that may be required to have a better understanding of the requirements. We refer to the new set of use cases as mobile user cases. They are a superset of the original ones, though the user may decide to eliminate some of the typical use cases after considering the mobile condition of the user and the additional functionality available based on the infrastructure and devices. Given that this exercise could yield a large number of use cases, there must be a prioritisation of the use cases that should be delivered first.

Now we should have a set of use cases that must be implemented by a development team, preferably where the UI is implemented separately from the business logic. In this work, our focus is on the translation of these use cases into a working user interface. It is assumed that a similar exercises is undertaken for the business logic of the application using the part (or all) of the use cases. For instance, the use cases that are most relevant to the user interface are those that describe activities that concern the direct action of the user on the application and response of the system, and to lesser extent, the processes that occur in the background. On the other hand, the use cases that are most relevant to the business logic are those that concern back-end process and methods that deal with the interaction between the UI and service provider.

We also propose a mobile use case evaluation matrix which extends the mobile use case evaluation matrix proposed in [6]. The rows of the matrix table provide a taxonomy of functions, adaptation features of the mobile UI due to change in the conditions and the dimensions of adaptation of the application which adds value to the user. This matrix table helps us to categorise and prioritise the use cases, by favouring those that offer the best value for the user. The columns indicate the mobile cases that were generated in the previous step. The content of matrix is to be filled with a number between 0 and 1 (or alternatively 0% to 100%) which give quantitative summary of a number of metrics, such as the relevance of the adaptation in a given context-of-use, its importance to the user, its usage frequency and complexity of integration.

To derive the importance of the use case to the user, it is possible to survey a sample of users on the predicted usage of each use case. The number indicates the fraction of the users who will use the adaptation functionality for a given use case. For each functionality, we need to incorporate the evaluation of how frequently the user will be using such feature in a given use case. This can be represented with a number between 0 and 1, indicated by the usage proportion in the surveyed population. Note that the goal behind this survey is not just to have an estimate of what the user perceives as useful and essential but rather to reveal possible usages of mobile functionality that had not been covered before.

Another aspect that may be considered is the level of complexity of integrating the different infrastructure technologies to enable the application. So we need to subjectively evaluate the use case for each infrastructure. This is however difficult since it is not possible to know the relevance of a use case to the infrastructure until the application is implemented. The idea is to use numbers to quantify the relationship between the requirements and various aspects of the design and implementation of the mobile application based on some assumptions.
We can then average the indices in each matrix (value, usage, simplicity, etc.) to determine the priority of the use cases in delivering the application. The information on any instance of the mobile use case matrices should be updated in an iterative manner throughout the development process because further requirements of the system could be discovered as the system is designed and implemented.

We illustrate this approach by using the scenario described above, and making some assumptions on the value of the application, usage frequency, and simplicity (or complexity) of integration\(^{\text{10}}\) to create a listing by priority of use cases to consider for development. First, the adaptation functions are determined, then for each use case/function combination, a corresponding index is inserted. The table below shows an example of the mobile use case evaluation matrix for each of the dimensions concerned. The last table represents the average of those evaluations matrices. Based on that, we can conclude that the most critical functions to implement are UC1 and UC3 with support for location detection; UC2 with support for user preferences; UC4 with support for network detection and user preferences. UC5 on the other hand has a low index, which makes it a good candidate for review.

\(\begin{array}{|c|ccccc|}
\hline
\text{Functions/UC} & 1 & 2 & 3 & 4 & 5 \\
\hline
\text{Location detection} & 0.8 & 0.8 & 0.8 & 0.4 & 0.2 \\
\text{Network detection} & 0.6 & 0.4 & 0.5 & 0.8 & 0.5 \\
\text{User preferences} & 0.6 & 0.8 & 0.3 & 0.8 & 0.3 \\
\hline
\end{array}\)

(a) Value to the user

\(\begin{array}{|c|ccccc|}
\hline
\text{Functions/UC} & 1 & 2 & 3 & 4 & 5 \\
\hline
\text{Location detection} & 0.4 & 0.5 & 0.8 & 0.2 & 0.5 \\
\text{Network detection} & 0.3 & 0.4 & 0.6 & 0.8 & 0.3 \\
\text{User preferences} & 0.3 & 0.8 & 0.3 & 0.8 & 0.3 \\
\hline
\end{array}\)

(b) Usage frequency

\(\begin{array}{|c|ccccc|}
\hline
\text{Functions/UC} & 1 & 2 & 3 & 4 & 5 \\
\hline
\text{Location detection} & 0.63 & 0.66 & 0.76 & 0.2 & 0.5 \\
\text{Network detection} & 0.5 & 0.66 & 0.56 & 0.73 & 0.46 \\
\text{User preferences} & 0.56 & 0.76 & 0.33 & 0.8 & 0.46 \\
\hline
\end{array}\)

(c) Simplicity of integration

\(\begin{array}{|c|ccccc|}
\hline
\text{Functions/UC} & 1 & 2 & 3 & 4 & 5 \\
\hline
\text{Location detection} & 0.7 & 0.7 & 0.4 & 0.3 & 0.8 \\
\text{Network detection} & 0.6 & 0.6 & 0.6 & 0.6 & 0.6 \\
\text{User preferences} & 0.8 & 0.7 & 0.4 & 0.8 & 0.3 \\
\hline
\end{array}\)

(d) Summary of mobile use case evaluation matrix

Table 3.5: Example of mobile use cases evaluation matrix elaboration

Once the list of use cases have been updated, it is possible to use UML's Use Case diagram to show the the functions provided by the UI and use the <<extend>> relationship (represented with a dashed arrow) to indicate what adaptation behaviour of the extension use case may be inserted in the extended use case. Constraints are then associated with the relationship to indicate the conditions in which adaptation behaviour is execution, as shown in Figure 3.19.

\(^{\text{10}}\)Note that for the evaluation of simplicity of implementation of network detection, it is assumed that there are mechanisms in the network that inform the device of the levels of QoS, which are not dependent on the application used.
3.5. Mobile User Interface Development Process

In stationary applications, once user requirements are gathered and use cases are developed, the next step would be to start producing mock-ups of the application, including the UI. It is very crucial in UI development cycle, as UI designers describe UIs using abstract UI terminology. Then during subsequent implementation phases, they create prototypes using platform-independent widgets, defining their behaviour, properties and basic UI layout. We need to follow the traditional stage of software application development cycle. At a high level, the traditional user interface development cycle follows these steps, which can be iterative:

1. Requirement gathering
2. Use case development
3. Mock-up production
4. Software development and implementation
5. Testing and usability evaluation
6. Use acceptance

As we are considering mobility dimensions, these processes need to be modified because we want to integrate context-of-use information in the application development process by linking conditions to effects on the UI architecture and behaviour, and support the model-based approach in the development process by combining model-specific processes with the standard processes. As a result, we have added new steps that take into consideration context aspects and integrates the model-based approach. Figure 3.20 illustrates the updated mobile UI development process, highlighting in grey the new and modified steps.

Figure 3.19: Example of Use Case diagram with adaptation support

3.5.2.2 Updated mobile UI development process
3.5. Mobile User Interface Development Process

The new cycle include the following steps:

1. Gather the requirements of the application and prepare a scenario.
2. Create the relevant use cases from requirements and iteratively select the most important ones using the mobile evaluation matrices by surveying users and/or based on estimation.
3. Create diagrams to convey the external functionality (i.e. the visible part) of the UI and the internal functionality of the system (i.e. the invisible part) using CTT and UML notations. These two processes should run in parallel, ensuring that the programmers work on the business logic and UI designers continue to specify the UI and adaptation rules using UML. Although the process used in the design of application business logic should be independent from the process of creating UI models, the two need to be synchronised so that the final UI can be 'plugged' into the application's functions and for the adaptation of the UI to be consistent with adaptation features implemented in the backend. In the above diagram, we have intentionally expanded on the stages of developing the UI, and hid away the details on the steps needed for the development of the application business logic. The UI-specific UML's are constructed using an iterative approach that separates the concerns of the UI from that of the context of use:
   
   (a) Develop a task model by processing the scenario (or use cases if sufficiently detailed) and create CTT diagrams that represent the tasks involved in the interaction with the user.
   
   (b) Develop UML diagram to represent the structure of the context model.
   
   (c) Develop UML diagram to represent the structure of the adaptation model and their effects on the task and concrete presentation models (i.e. adaptation rules).
4. Build an abstract UI mock-up based on the generated task model, using editing tools such as TERESA.

5. Build the internal aspects of the application that delivers the business logic functionality to the end user. This includes integration with the mobile infrastructure such as device specific functionality, location information, context information, etc.

6. Perform a series of tests on the internals of the application (not shown in the diagram).

7. Test the resulting concrete presentation UI from the applications of adaptation rules (i.e. for consistency, structure and behaviour).

8. Build a final UI using the selected rendering technology and platform-specific modalities.

9. Test of the generated final UI's as a result of the application of adaptation rules (i.e. usability, layout, etc.), and refine the adaptation rules accordingly, for instance, by introducing UI design guidelines via setting heuristics during UI generation, and applying stylesheets to modify the layout and style of the final UI.

10. Go through the users acceptance process in which the users interact with the application including the UI's, then accept the product or give additional requirements complement/replace existing requirements, and the process is repeated.

In the following two sections we will expand on how the task model is generated and how adaptation rules are established, which we believe are two crucial aspects of this process. The implementation aspects will be covered in Chapter 4 where we will show on how to serialise the different models to enable their transformation and adaptation. In Chapter 5, we present further discussions on the implementation and deployment aspects, with an example that shows how to follow this cycle in practice with the help of a demonstrator application.

### 3.5.2.3 From scenario description to task model

UML is biased toward design and development of the internal part of software systems, and has proven its effectiveness in doing this. However, some UML constructs can be useful in building task models, such as use cases. The latter can be used to identify the actors involved (i.e. users, systems) and describe the structure and flow of data between the different entities, which can then be mapped into the roles of the CTT models. A use case can also allow the identification of the main tasks that should be performed by users and the system or their interaction, which are mapped to tasks with corresponding category. However, interaction aspects, such as composition of the UI, sequence of actions, etc., can not be well captured in use cases and to overcome this limitation they can be enriched with scenarios and informal descriptions of specific use of the system considered.

Indeed, a careful analysis of the scenario allows designers to obtain a description of most of the activities that should be considered in a task model. The main difference between a task model and a scenario is that a scenario indicates only one specific sequence of occurrences of the possible activities while the task model should indicate all the possible activities and the related temporal relationships. This is more relevant in the case of context-dependent task models whereby the affects of a change in the context need to be identified as well.
3.5. Mobile User Interface Development Process

We start with an informal description of a scenario. The scenario should include most of the main activities involved by the application considered. Next, keywords that denote activities are highlighted and added to the list of tasks. At this point, it may be necessary to edit the names of the tasks to make them more general and try to group multiple activities under the same task. In terms of task allocation, it is down to the designer to make such a judgment: a task is set to user type, if internal cognitive actions are required; to application type if business logic actions are required; to interaction type if the activities consists of user interactions with the device; or to abstract type (if it is an activity that entails other subtasks). These allocations are reflected in the icons used in the task model editor.

The next step is to identify the structure of the task model. First, the UI designer needs to identify the hierarchical structure that describes the various levels of abstraction among the task nodes, and specify their temporal inter-relations, their attributes and associated objects. Once the tasks are identified, it is important to indicate the objects that have to be manipulated to support their performance. Two types of objects are considered: concrete UI objects and application domain objects, such as back-end functions or application methods that need to be executed when a user activates a particular concrete widget. The extraction of temporal relationships and task properties can be performed after listing the tasks, by analysing the scenario semantics and highlighting the keywords which hint to the nature of the temporal relationships between the different activities. Table 3.6 shows a possible mapping between keywords that could be encountered in a scenario and their significance in the context of a task model.

The resulting draft hierarchical task model can then be further edited and refined by existing ConcurTaskTrees editors such as TERESA or CTT. By following this approach, it is possible for a designer to build an early task model which contains the main activities, and which can be refined later if needed.

<table>
<thead>
<tr>
<th>Description</th>
<th>Temporal relationship</th>
<th>CTT notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;do task t1&quot; in order to &quot;do task t2&quot;</td>
<td>Hierarchy</td>
<td>( t_1 \text{ is child of } t_2 )</td>
</tr>
<tr>
<td>&quot;do task t1&quot; then &quot;do task t2&quot;</td>
<td>Enabling operator</td>
<td>( t_1 \triangleright t_2 )</td>
</tr>
<tr>
<td>&quot;do task t1&quot; then &quot;do task t2&quot; accordingly</td>
<td>Enabling operator with information exchange</td>
<td>( t_1 [&lt;&gt;] t_2 )</td>
</tr>
<tr>
<td>&quot;do task t1,&quot; is interrupted by &quot;do task t2&quot;</td>
<td>Disabling operator</td>
<td>( t_1 \downarrow t_2 )</td>
</tr>
<tr>
<td>&quot;do task t1,&quot; and &quot;do task t2&quot;</td>
<td>Concurrent operator</td>
<td>( t_1</td>
</tr>
<tr>
<td>&quot;do task t1,&quot; and &quot;do task t2&quot; synchronously</td>
<td>Concurrent operator with information exchange</td>
<td>( t_1</td>
</tr>
<tr>
<td>&quot;do task &quot; or &quot;do task t2&quot;&quot;</td>
<td>Choice operator</td>
<td>( t_1</td>
</tr>
<tr>
<td>&quot;do task t1,&quot; then &quot;do task t2,&quot; when finished</td>
<td>Order independence operator</td>
<td>( t_1 = t_2 )</td>
</tr>
<tr>
<td>&quot;do task t1,&quot; until &quot;do task t2,&quot; starts</td>
<td>Deactivation operator</td>
<td>( t_1 \triangleright t_2 )</td>
</tr>
<tr>
<td>&quot;do task t1,&quot; unless &quot;do task t2,&quot; starts, then resume when finished</td>
<td>Suspend resume operator</td>
<td>( t_1 [&lt;&gt;] t_2 )</td>
</tr>
<tr>
<td>&quot;do task t1,&quot; optionally</td>
<td>Optional attribute</td>
<td>( t_3 )</td>
</tr>
<tr>
<td>&quot;do task t1,&quot; repeatedly (or n times)</td>
<td>Iteration attribute</td>
<td>( t_1^n \text{ or } t_2(n) )</td>
</tr>
</tbody>
</table>

Table 3.6: Mapping between textual descriptions and CTT notation
3.5.2.4 Elaboration of adaptation rules

As explained before, an adaptation rule is composed of the condition part, which is formed by a context condition defined with a relational statement of context parameters and threshold values that combine arithmetic and logical operators; and the adaptation part, which is composed of a set of actions defined for the task and concrete presentation models, which may include the addition/removal of task nodes, change of task's temporal relationships and change in the attributes of the concrete interactor objects (CIO's). And example of such mapping is shown the table below.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Adaptation Type</th>
<th>Model Level</th>
<th>Operation</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ctxparam_1 = 1$ and $ctxparam_2 &gt; 10$</td>
<td>Structure</td>
<td>Task</td>
<td>Add</td>
<td>$t_1, t_2$</td>
</tr>
<tr>
<td></td>
<td>Attribute</td>
<td>Task</td>
<td>Change</td>
<td>$relation(t_1) = &quot;</td>
</tr>
</tbody>
</table>

Table 3.7: Example of an adaptation rule

Depending on the type of adaptation to be supported by the UI, the designer is involved in specifying the three parts of a transformation rule. For instance, in the case of an action which results in the addition of a new task tree to the task model, the UI designer is expected to define the new task subtree based on structural change exhibited by the UI. Similarly, for behaviour change to the UI, the UI designer is expected to define the task node parameters which achieve that transformation. Finally, if any visual-level changes are explicitly specified in the requirement elicitation stage, they should also be included in the adaptation rules.

In section 2.7.9.3, we explained the adaptation process of UI models using examples of context elements, and in this part, we expose how we can extract the adaptation rules from a scenario. The general approach is as follow:

1. Determine the mobility dimensions that trigger the change in the UI such as network status, location or user profile. This will provide us with the name of the context, its parameters (if composite), its nature (static or dynamic) and its source (i.e. environment, platform or user).

2. Determine the nature of change in the UI (i.e. structural, behavioural, visual style) based on its expected effects on the UI, which should be available during the requirement elicitation stage.

3. Determine the level at which the transformation action needs to be executed (i.e. task model or presentation model).

4. Specify nature of transformation actions required (structural, attribute).

5. Determine the task nodes and presentation objects concerned with the changes.

6. Specify the states of the context which trigger a change in the UI. These are specified by relating the context parameters to a given value using an arithmetic operator. This will form the condition part of the adaptation rule. If the combination of multiple context elements is needed to trigger adaptation, then the various constituent parameter-operator-state terms can be combined using a logical operator (and, or).

\[11\]The use of UI mock-up could help users think about the possible effects on the UI as a result of a change of context.
7. Define the transformation actions in terms of operations (i.e. add, remove, change, etc.) and operands (i.e. parameter, task nodes/subtrees) involved in the action.

While the starting point of these requirements is from potential users, it is down to the UI designer to augment them with quantifiable values and relate them to the implementation considerations and the features of the development tools, to determine steps 2 onward. Experts in the various domains that are related to the dimensions of mobility, such as location and network QoS, could provide recommendations on the typical operational conditions of mobile applications. These suggestions could be compiled over time into chart whereby UI designers can look up a given context element and determine the various value ranges supported, and take that into consideration during the elicitation of user requirements. The information extracted from the whole process, can then be modelled visually using UML diagrams as explained in section 3.4.5, and also used to create XML-based versions which can be used for implementation, as we will show in section 4.3.7. It is important to note that the UML profile we proposed does not support any intelligence that can deduce automatically new adaptation rules, so designers have to imagine all the possible adaptations according to the contexts-of-use under consideration.

3.6 Conclusion

In this chapter, we have proposed a UML profile to model the task model, the context model and the adaptation model, with focus on modelling the structural aspects of the interaction and complementing existing notations such as CTT. We have also shown how to combine standard software development cycle with model-based processes to support the design and implementation of mobile user interfaces that are adaptable to a change of context. Finally, we have presented an approach to generate a task model from an informal scenario description, and specify structural, behavioural and style adaptation rules. Notably, the proposed UML profile is less complex in comparison with existing UML-based notations, yet, we believe that it sufficient for the needs to specify the various models in implementation-independent manner and use them as part of a mobile user interface development cycle.
4.1 Introduction

To address the problem of implementing pervasive user interfaces and respecting each device constraints, much work has been done on the semi-automatic generation and adaptation of the user interface (UI) on different hardware and software platforms [23]. Each method provides a different approach to describing a UI in a platform-independent manner, using a high-level UI descriptive language (UIDL). This description is then rendered with the concrete platform-dependent UI by using a rendering engine.

There are different approaches to developing high-level UIDL, among which we can mention language based (e.g. XML) and grammar-based (e.g. BNF\(^1\)) notations. These approaches generally model some aspect of the user interface and are combined with other models to describe the complete UI. It is mainly used for later stage of UI development, where interaction widgets are described, instead of the requirement and design stage.

Noticeably, in recent years, XML has proved to be a solid foundation for the description of the different UI models because it offers a number of properties which make it particularly suitable for user interface model description, as we will see in this chapter. In the following sections, we will first present a review of how XML has been used as a high-level UIDL and provide examples of such uses. Then we will provide a detailed account of how we have used XML for the description of the various models considered in our approach, from task model to final user interface rendering, including context and adaptation models.

\(^1\)BNF is widely used meta-syntax used as notation for grammars of programming languages, protocols, etc.
4.2 Markup-based Description Languages

4.2.1 XML for UIDL

The predecessor to the development of XML-based UIDL's was declarative\(^2\) UI description languages. However, they did not manage to leave the academic arena because of their specialisation and lack of agreements on the syntax and semantics. With the advent of web technologies, the UI development work was no longer restricted to professionals but allowed everyone with little (or no) programming experience to develop web-based interactive applications using markup languages for content (e.g. XHTML) and layout/style (e.g. CSS). Since the access to web content has become more ubiquitous, this has called for the development of ways to communicate the device characteristics and user preferences and provide techniques for device-independence content creation and visualisation. XML has been proposed as the evolution of web markup language which offers a number of characteristics which make it suitable for the description of UI models:

- **Formal:** The grammar rules of XML are defined in Extended Backus-Naur Form Notation (EBNF), which provides a formal definition of the language rules. This formal structure makes it also easier to write XML parsers.

- **Platform-independent:** Since XML is a text-based format, the requirements for its use are fairly basic. Provided that a computing platform can allow the programming of a text parser, it is possible to write an XML parser. In fact, nowadays, there are XML parsers written in virtually all programming languages and available on all platforms.

- **Declarative:** XML can be used to describe what each UI element does without specifying how it does it. A useful tool to render XML statements into meaning-rich formats is XSL (eXtensible Style Language). It provides transformation and formatting capabilities that can be applied to the XML document or part of it.

- **Consistent:** The XML format has a native support for consistency through the use of schema specification written in DTD\(^3\) or XSD [91].

- **Modality independent:** XML has proved to be a robust format for the description of multimodal interaction, and can be transformed to various output formats. Examples include: XHTML, WML (Wireless Markup Language), VoiceXML [90], EMMA (Extensible Multi-Modal Annotation markup language) [94], etc.

- **Extensible and reusability:** Since XML is a meta-language (i.e. language used to define the grammar and terminology of another language), it can be extended very easily. Furthermore, the structure of XML makes it possible to reuse parts of the original XML file into other descriptions.

- **Processing requirements:** XML has been supported on Java Mobile Edition (JME) since 2000 using web services API's which enable the processing of XML data. In addition, the task of parsing and interpreting the XML can be done on the mobile device itself, or some proxy such as an application server that processes content for the device.

---

\(^2\)Declarative in the sense that the UI designer can select which UI elements to generate, instead of indicating how to create them

\(^3\)A Document Type Definition defines the set of valid elements (or tags) and the structure of the XML document to which it complies. It can be declared in-line in the XML document, or as an external reference.
4.2.2 Review of XML-based UIDL's

XML-based UI markup language have been proposed to cover much of the models in MBUID approaches. The level of abstraction varies: some include information about the end-user device, some make assumption on the type of modalities, some others incorporate user information in the form of preferences, or user's behaviour. They can be classified by the model (or models) they are used for. Those that positioned at the abstract end of the spectrum generally rely on model transformation to increase their specialisation and adapt to a set of requirements. These languages can be broadly categorised into three groups:

1. **Platform-independent vocabulary and toolkits**: Use a set of generic widgets to specify the UI, which are then mapped to platform-specific widgets (e.g. AUIML) or left to the client-side runtime environment to perform the mapping (e.g. XUL, XAML).

2. **Extend established markup languages**: Add meta-data to the UI description in the form of additional markup or attributes. This extra information is used to hint adaptation for certain device or delivery scheme (e.g. UIML, XForms).

3. **Model-based user interface development**: Different models are used to describe different aspects of the interaction between the user and the application/device. Models are transformed from the most abstract ones to the most concrete ones with the help of a runtime architecture (e.g. XIML, UsiXML).

In the following sections, we offer a short review of some of the key UIDL's found in the scientific literature, namely: AUIML, UsiXML, UIML, XForms, XIML and XUL, covering the three above-mentioned categories.

4.2.2.1 AUIML

The Abstract User Interface Markup Language (AUIML) [17] is an intent-based description language developed by IBM, which was eventually discontinued in 2004. This method stipulates that once the core of the application is written, the AUIML toolkit can be deployed to implement Java Swing (i.e. Java GUI library) or a web user interface. The description is manipulated with specialised tools/plug-ins, because it is not human readable. AUIML provides a separation between the user interface structure (data model) and style (presentation model), like most UIDL's.

4.2.2.2 UsiXML

The User Interface eXtensible Markup Language Language (UsiXML) [38] is a description language which describes the UI with various levels of abstraction, to support independence with respect to platform, modality and context. It's the most advanced and complete approach since it supports multiple models including domain, task, presentation, context, mapping models, and enables transformations from one model to another (using a transformation model, which is itself defined with XML). This language is well supported by a list of tools and applications, such as interpreters, UI generators and UI development environments.

---

4The designer specifies what the UI widget should do and not how it should be rendered.

5http://www.alphaworks.ibm.com/tech/auiml
4.2.2.3 UIML

User Interface Markup Language (UIML) [58] is used to define device-independent user interfaces. It is a notation that is used to describe the presentation model and to some extent the domain model. However, it does not natively support dialog and task models.

One of the advantages of UIML in comparison to other high level UIDL's is its expressiveness and simplicity; it allows to describe UI’s behaviour in a device-independent manner, it is able to describe content, structure, behaviour and style of UI separately, while using a small number of XML tags. In the latest (and likely the last\(^6\)) version of UIML V3, the UI is composed of four parts:

1. Headers part, denoted with `<head>` tags and provides extra information to the UIML parser or browser.
2. Interface part, which describes a user interface in terms of presentation widgets, component structure and behaviour specifications.
3. Template part, which enables reuse of UIML elements. When an element appears inside a template element it can used multiple times by elements with the same tag.
4. Peers part, which specifies how UIML interface components are mapped to the target UI platform. Normally, there are several peers subsections for each specific target platform, rendering technology and backend logic.

4.2.2.4 XForms

XForms [93] is an Internet-centric UI description language that is meant primarily for form-based user interaction and data submission. It separates the device-independent data of the form (XPoms Model), from its presentation (XForms User Interface), which provides standard visual controls that can be mapped to other GUI description languages e.g. HTML, WML and VoiceXML. An additional description is needed to define how XForms sends and receives data, including the ability to suspend and resume the completion of a form (XForms Submit Protocol). One of the advantages of XForms in comparison with HTML 4 forms is that UI's that use it require fewer round trips with the server and are more self contained. Furthermore, unlike other languages (e.g. UIML and UsiXML), XForms defines distinct and discrete controls and elements that define a language for building a generic UI. Consequently, it can be considered to be a meta language intended to create other languages that are used to build UI's. Although XForms is a promising solution to the web form description and data handling, there is still limited and out-of-date support for this language on portable devices.

4.2.2.5 XIML

eXtensible Interface Markup Language [64], has been specifically designed for developing multi-device user interfaces. XIML supports task, domain, user, dialog and presentation models. XIML is used to describe abstract concepts at the task, domain and user level, which are mapped to concrete representations on the device. This framework makes extensive use of inter and intra-model relationships. However, it seems that development of the language has ceased in recent years.

\(^6\)There has been no development on the specifications since 2002, and no editing or development tools are available for the mobile platform for testing and evaluation.
4.2. Markup-based Description Languages

4.2.2.6 XUL

XML User Interface Language (XUL) [25] is a cross-platform user interface markup language, which was initially developed to create the UI for the Mozilla web browser (http://www.mozilla.org), but has grown since to become a cross-platform development platform to support desktop and, to lesser extend, mobile applications. It heavily borrows from W3C standard technologies (e.g. CSS, XHTML).

4.2.2.7 Others

Other XML-based descriptive languages that are rather geared towards the desktop computers include Microsoft’s XAML7, which provides a large list of basic GUI widgets which can be grouped hierarchically to create a fully functional GUI's. The language syntax closely matches other XML-based description languages such as XHTML and XUL. SwingML (http://swingml.sourceforge.net/) is another XML-based set of specifications to define Java Swing-based GUI’s, for applications which run off a web server. It complements Java's support for browser-based application by providing a server-side generated Swing-based UI that can substitute traditional HTML control tags.

4.2.3 Discussion

During the review of the different high level UI description languages, we noticed that most of them have not been widely used beyond the scope of the research that they were created for in the first place. In addition, among those that had a more widespread usage, they either had no dedicated support for the mobile platform (like AUIML and XForms), or were converted into more well-supported formats but suffered from limited in functionality and richness, such as HTML or WML, to be able to run over mobile devices (like XIML, UsiXML). In the following table, we propose to evaluate the main high level UIDL’s according to their level of separation of data from presentation, logical grouping and mobile support.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>XForms</th>
<th>XUL</th>
<th>AUIML</th>
<th>XIML</th>
<th>UsiXML</th>
<th>UIML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separation of data/presentation</td>
<td>Yes</td>
<td>Partially</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Logical Groupings</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Demonstrated Mobile Support</td>
<td>Limited</td>
<td>Yes</td>
<td>No</td>
<td>View only</td>
<td>View only</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 4.1: Review of key high level UI description languages

As far as mobile devices are concerned, XForms clearly lacks the support of the mobile platform with very few commercial renderers available, in addition to its relatively high processing requirements. XIML is primarily a web-centric technology and can be used to generate UI’s which can be viewed with mobile devices. UIML provides tools to generate WML and VoiceXML. The latter example makes use of specialisation of generic components by using XML style sheets (XSL) or similar technologies. However, this is a very narrow slice of specialization techniques. We believe that specialization should not be limited to just transforming content but should include modifications in the structure and behaviour of the interface, adding device- or interface-specific features,

removing features not supported by (or not suitable for) the intended user interface or device, and taking all of the other context information into consideration. This is precisely what our approach aims to accomplish by using a number of XML-based description languages that support the model transformation based approach.

4.3 Proposed Approach

4.3.1 Introduction

The characteristics of XML which have been outlined above make it an adequate choice for the description of UI models. In fact, the TERESA approach, on which our work is based, makes extensive use of the language in all of the UI models supported (i.e. task/dialog and presentation), to facilitate the inter-model transformations. Furthermore, XML is well adapted to carry data entities (such as those found in the context and adaptation models), convey structural information (such as the decomposition of a presentation unit) and can be easily transformed and formatted to render usable final UI's. Indeed, XML is used in the design stage as well as the implementation aspects in our approach including the specifications of context information, adaptation rules and the final UI.

In addition, the requirements associated with the processing of XML files are particularly adequate for resource-constrained devices, such as mobile phones. Finally, the fact that XML is a text-based format means the existing editing tools can already be used to create and modify XML files. These tools range from the basic text editors to the more advanced visual editors. In the following sections, we will provide details about the various areas in which XML has been used, highlighting the advantages gained from using it and explaining how it relates to the other notations used in our approach such as CTT and UML.

4.3.2 UML and XML

As we have seen previously, some of the UI models used in our approach have been, partly or fully, designed using UML such as the task model, the context model and the adaptation model. It is true that UML is mainly used to convey more abstract concepts, whereas XML tend to be used to convey more concrete details. In fact, the use of XML representation provides a bridge between the conceptual level used for modeling UI aspect and the implementation level used by the system.

Nonetheless, this distinction ensures a clean separation between the modelling activity (carried out by UI modellers and designers) and the implementation activity (carried out by developers and programmers). As we have seen in section 3.5.2, the first steps in software development cycle involves virtually no technical activity, where users and UI designers can formulate application requirements. On the other hand, the latter stages of the cycle involves the development of UI mock-ups, implementation and testing, which are carried out in collaboration with similar activities related to application's business logic development. In the context of mobile development, UML and XML can be seen as complementary in modelling applications:

1. Using UML enables to create a visual aid to design mobile applications since it uses visual diagrams, which simplifies the understanding of XML documents which can be quite verbose and long. As a result, UML eases the process of design and analysis.
2. Mapping application-specific features of XML to UML has another benefit in encapsulating the semantics of the application of XML and making it easier to understand those semantics through the graphical tools that UML offers.

3. XML offers some advantageous characteristics such as XML Schema which ensures the validity of the models and uniformity of the models against the vocabulary definition. Despite the fact that XML is text-based format, there are ways to compress it, and processing overhead are minimal even on less capable devices. It also gives a more practical representation to be used directly by the UI generation and adaptation system.

There are generally two main approaches to mapping UML and XML:

1. At the metamodel (or meta-metadata) level: This can be done by mapping XML to existing features of UML (e.g. class, associations). One way to achieve this is to use XMI (XML Metadata Interchange) [51], which is a standard XML-based serialisation format that can be used by UML tools to facilitate interoperability among them. However, adopting the XMI format can result in very verbose XML profiles which are not suitable for fast processing on the mobile platform.

2. At the model (or metadata) level: It is possible to map an XML's schema or DTD to a model in UML using class diagrams (and vice versa). Since the UML profile defined in the previous chapter is Class diagram based, we believe that creating mapping at the model level is the most appropriate approach. This allows the mapping of class stereotypes, tagged values and associations into XML schema. Then the class instances can be mapped to XML tags and properties which can be stored, processed and manipulated.

In the following sections, we will show how such mappings have been used. Our first priority has been to maintain the semantics of the UML in the XML Schema so that UI designers and those implementing the applications can align their work. Furthermore, since we are targeting resource-constrained devices, the second priority has been to keep the XML schema simple enough to be easily processed, even if that means that not all information contained in the UML models is integrally conveyed in the XML files.

4.3.3 Task Model

The original TERESA model-transformation approach defines an XML-based notation, called TeresaXML\(^8\), which represents a task model defined in CTT. It is possible to map the UML model presented in section 3.4.3 to XML by applying a simple semantic mappings between the stereotypes in the UML and the ELEMENT tags in XML; and between class properties with XML element attributes list (ATTLIST) of those elements. Furthermore, class associations could be used to indicate the hierarchical structure of the XML elements. While this is feasible, it is more efficient to use the original CTT editing tools to create fully compatible and complete XML-based representations of the task model. This is because the notation used for the definition of the UML model is a simplified version of the original CTT, and in mapping it into XML, there are inevitably some information about the task models that will be missing (such as task pre-conditions).

\(^8\)http://glove.isti.cnr.it/tools/TERESA/teresa_xml_ctt.html
4.3. Proposed Approach

An excerpt from the task model is shown in Listing 4.1. This is in fact the XML version of the CTT model, depicted in Figure 3.14a, and UML model, depicted in Figure 3.14b. It provides a description of a set of tasks for the mobile platform: “Show museum info”, whose parent is a task “Access general info”, right sibling is “Back to main page” and left sibling is “Select access to general info”. The latter two tasks are defined below. The DTD of the task model is shown in Appendix A.2.

Listing 4.1: Excerpt from a Task Model defined with TeresaXML

```xml
<Task Identifier="Show museum info" Category="application" Iterative="false" Optional="false" PartOfCooperation="false" Frequency="Medium">  
  <Name> </Name>  
  <Type> Overview </Type>  
  <Description> </Description>  
  <Platform> Cellphone </Platform>  
  <Precondition> </Precondition>  
  <TemporalOperator name="Disabling"/>  
  <TimePerformance>  
    <Max> </Max>  
    <Min> </Min>  
    <Average> </Average>  
  </TimePerformance>  
  <Parent name="access general info"/>  
  <SiblingLeft name="Select access to general info"/>  
  <SiblingRight name="Back to main page"/>  
  <Object name="overview" class="Text" type="Perceivable" access_mode="Access" cardinality="null">  
    <Platform> Cellphone </Platform>  
    <InputAction Description="null" From="null"/>  
    <OutputAction Description="null" To="null"/>  
  </Object>  
</Task>

<Task Identifier="Select access to general info" Category="interaction" Iterative="false" Optional="false" PartOfCooperation="false" Frequency="Medium">  
  <Name> </Name>  
  <Type> Control </Type>  
  <Description> </Description>  
  <Platform> Cellphone </Platform>  
  <Precondition> </Precondition>  
  <TemporalOperator name="SequentialEnabling"/>  
  <TimePerformance>  
    <Max> </Max>  
    <Min> </Min>  
    <Average> </Average>  
  </TimePerformance>  
  <Parent name="access general info"/>  
  <SiblingRight name="Show museum info"/>  
  <Object name="info" class="Text" type="Perceivable" access_mode="Access" cardinality="null">  
    <Platform> Cellphone </Platform>  
    <InputAction Description="null" From="null"/>  
    <OutputAction Description="null" To="null"/>  
  </Object>  
</Task>

<Task Identifier="Back to main page" Category="interaction" Iterative="false" Optional="false" PartOfCooperation="false" Frequency="Medium">  
  <Name> </Name>  
</Task>
```
4.3. Proposed Approach

4.3.4 Abstract Presentation Model

The abstract presentation model defines a platform-independent UI structure and behaviour of the UI. It concerns the structure of the UI and functionality rather than the low-level implementation details. We based the XML-based specification of the model on TeresaXML AUI\(^9\) since it is an integral part of the TERESA model transformation approach. The added advantage of using this XML schema is that it is freely available, and comes with an integrated visual AUI editor.

The language defines a number of presentations for the UI static structure and a number of transitions to indicate how the UI evolves over time. Each presentation is constituted of a set of interactors composed of number of compositions that exhibit an interaction functionality e.g. text entry. Interactors can be combined using composition operators. Such operators can involve one or two expressions, each of them can be composed of one or several interactors or, in turn, compositions of interactors. While each presentation defines a set of interaction techniques perceivable by the user at a given time, the connections define the dynamic behaviour of the user interface, by indicating what interactions trigger a change of presentation and what the next presentation is. Both the static arrangement of interactions in the same presentation and the dynamic behaviour of the abstract user interface are derived by analysing the temporal operators included in the task model specification (which can also be defined with the XML version of CTT\(^10\)). The presentation operators that are supported:

- **Grouping (G):** Indicates that tasks have the same parent, whereby the same operators is applied to siblings. This is triggered by [ ] (Choice) in the task model.

- **Ordering (O):** Indicates a temporal ordering between tasks. This is triggered by [ ] (information exchange + enabling).

\(^9\)For reference, the full specifications of the abstract presentation model is available from [http://giov.e.isti.cnr.it/tools/TERESA/doc/au1.txt](http://giov.e.isti.cnr.it/tools/TERESA/doc/au1.txt) in the form of a DTD

\(^10\) [http://giov.e.isti.cnr.it/tools/TERESA/teresa_xml_ctt.html](http://giov.e.isti.cnr.it/tools/TERESA/teresa_xml_ctt.html)
• **Relation (R):** Applicable only if there is relation between leaf tasks. All subtasks could be disabled by a task if it is positioned at the right of a disabling operand [>].

• **Hierarchy (H):** Indicates priority between tasks.

For instance, a sequential operator between two tasks implies that the related presentations will be sequentially triggered: this will be rendered, at the abstract level, by associating a connection between two different abstract presentations, so that the first task will trigger the activation of the second presentation and render the sequential ordering. Using a concurrency operator between two tasks implies that the associated interactors will be presented at the same time so the, associated abstract objects will be included in the same presentation. Structurally, an abstract presentation is made up of a number of Connections, and Interactors or Interactor Compositions:

- **Connection:** It describes the behaviour of UI, identify the Abstract Interactor Object (AIO) ID whose activation allows the interface to move to another presentation. It provides a presentation_name, to indicate the presentation it is moving to, and the connection type, which can be *elementary* where the an interactor_id is specified, or complex, which represents a combination of connectors linked with logical operators AND, OR to build multi-interactor based activation.

- **Structure:** It is used to represent the static arrangement of the UI, which can be made up of single AIO or a composition of them (known as **Composite Interactor Object**). The composition itself can be made up of one presentation operator with a number of AIO's or AIO compositions.

- **Interactor:** It can be an interaction or only_output object. The interaction itself can be a (1) selection, (2) edit, (3) control or (4) interactive_description. Selection AIO can be single_choice or multiple_choice, and each of them can have low, medium or high cardinality. Edit object can be text_edit, object_edit, numerical_edit or position_edit. Control AIO can be navigator or activator. As for only_output AIO, there are four options: (1) text, (2) object, (3) description and (4) feedback.

Since we are not targeting multi-modal interaction in our approach, the existing version of TeresaXML AUI is adequate to provide a bridge between the task model and the concrete presentation model. An excerpt from the abstract presentation model generated automatically from the full task model above is shown in Listing 4.2.

**Listing 4.2: Excerpt from an Abstract Presentation Model defined with TeresaXML**

```xml
<presentation name="presentation_1">
  <connection presentation_name="presentation_3">
    <conn_type><elementary_conn interactor_id="Select.access_to_artworksl"/></conn_type>
  </connection>
  <connection presentation_name="presentation_6">
    <conn_type><elementary_conn interactor_id="Select.access_ticket_bookig1"/></conn_type>
  </connection>
  <interactor_composition><operator name="grouping" /></interactor_composition>
  <first_expression><interactor id="Show_introl"/>
```
4.3. Proposed Approach

4.3.5 Concrete Presentation Model

Each platform-independent AIO from the abstract presentation model is associated with one or more concrete interactor objects (CIO's) which contain a collection of multiple alternative concrete representations. Each of the abstract widgets has a number of properties which hint to the presentation middleware on how to display them on the different devices. This can be illustrated as a tree, as shown in Figure 4.1. There is a root node which represents the main presentation unit and the subsequent nodes representing UI "containers". The tree siblings represent the "atomic" widgets such as text entry fields and sliders. Depending on the platform's type and constraints (e.g. availability of a screen, graphics support, modalities supported), different mappings of AIO's are possible.

Accordingly, TERESA provides one language for abstract presentation description and multiple XML-based concrete presentation languages depending on the target device (which depends on the size of the display and its media support) and the type of modality supported (i.e. graphical UI, voice)\(^\text{11}\). The process of converting from abstract to concrete presentation is semi-automatic in

\(^{11}\)As of Teresa version 2.5a, there are six concrete implementations supported: multimedia desktop, mul-
4.3. Proposed Approach

TERESA, whereby the designer sets the default mappings for AIO's and presentation operators. For the mobile concrete UI, the designer can select from TERESA CUI editor between different options on how to represent a particular structural compositions depending on the operator used. A Grouping (G) can be mapped to either Unordered List on Column, or Fieldset; Ordering (O) can be mapped to an Ordered List on Column; Hierarchy (H) depends on the size of the device selected, to determine the number of activities supported, and Relation (R) can be mapped to a Form. The list of abstract to concrete interactors mappings for the mobile platform is shown in in the table below.

<table>
<thead>
<tr>
<th>Only_Output Interactors</th>
<th>Interaction Interactors</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIO</td>
<td>CIO</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Navigator</td>
</tr>
<tr>
<td>Text, Image or combination of both (Text with Image)</td>
<td>A Link, which is a text-based navigator, or Button</td>
</tr>
<tr>
<td>Text</td>
<td>Activator</td>
</tr>
<tr>
<td>Label</td>
<td>Reset Button, Button with Script or Activate Database</td>
</tr>
<tr>
<td>Object</td>
<td>Text Edit</td>
</tr>
<tr>
<td>Image container</td>
<td>Text Field (visible/hidden) or Area Field</td>
</tr>
<tr>
<td></td>
<td>Numerical Edit</td>
</tr>
<tr>
<td></td>
<td>Text Field (visible/hidden)</td>
</tr>
<tr>
<td></td>
<td>Single Selection</td>
</tr>
<tr>
<td></td>
<td>Radio Buttons (horizontal or vertical) or drop down list</td>
</tr>
<tr>
<td></td>
<td>Multiple Selection</td>
</tr>
<tr>
<td></td>
<td>Checkbox</td>
</tr>
<tr>
<td></td>
<td>Interactive Descriptions</td>
</tr>
<tr>
<td></td>
<td>Link, GraphicalLink, Button or Text</td>
</tr>
</tbody>
</table>

Table 4.2: Mappings between AIO’s and CIO’s in TERESA

Given our focus on the mobile platform and graphical user interface, we have adopted the mobile timedia pda, mobile, digital TV, and voice
4.3. Proposed Approach

version of TeresaXML CUI, of which the full schema is available from Appendix A.3 in the form of a DTD. An excerpt from the concrete presentation model generated from the full abstract presentation model above is shown in Listing 4.3.

Listing 4.3: Excerpt from a Mobile Concrete Presentation Model defined with TeresaXML

```xml
<presentation name="presentation_1">
  <presentation_properties>
    <title value="Marble Museum"/>
  </presentation_properties>
  <connection presentation_name="presentation_3">
    <conn_type>
      <elementary_conn_interactor_id="Select_access_to_artworks1"/>
    </conn_type>
  </connection>
  <connection presentation_name="presentation_6">
    <conn_type>
      <elementary_conn_interactor_id="Select_access_ticket_booking1"/>
    </conn_type>
  </connection>
  <connection presentation_name="presentation_2">
    <conn_type>
      <elementary_conn_interactor_id="Select_access_to_gen_info1"/>
    </conn_type>
  </connection>
  <interactor_composition>
    <operator id="Grouping_L_0">
      <grouping>
        <first_expression>
          <interactor id="Show_introl">
            <only_output>
              <text>
                <input_text value="Welcome to the Marble Museum of Carrara "/>
              </text>
            </only_output>
          </interactor>
        </first_expression>
        <first_expression>
          <interactor id="Select_access_to_gen_info1">
            <interaction>
              <control><navigator><text_link label="Select access to general information"/></navigator></interaction>
            </control>
          </interactor>
        </first_expression>
        <first_expression>
          <interactor id="Select_access_to_artworks1">
            <interaction>
              <control><navigator><text_link label="Select access to artworks"/></navigator></interaction>
            </control>
          </interactor>
        </first_expression>
        <first_expression>
          <interactor id="Select_access_ticket_booking1">
            <interaction>
              <control><navigator><text_link label="Select access ticket booking"/></navigator></interaction>
            </control>
          </interactor>
        </first_expression>
      </grouping>
    </operator>
  </interactor_composition>
</presentation>
```
4.3. Proposed Approach

Unlike current MBUID approaches, such as CUP 2.0, TERESA and UIML, which have favoured the use of web-based final UI markup languages (e.g. HTML, XForms) to generate a prototype, we aim to automatically produce the XML-based high-level UI descriptions as well as the corresponding source code and style sheet.

While TeresaXML provides the means to describe the structure and behaviour of the UI in a platform-independent manner, it achieves this to the detriment of its ability to express features found in other UI declarative languages, such as styling. In addition, TERESA provides support for mobile phones by creating HTML documents and rendering them on the mobile browser. While HTML is a very pervasive and well supported markup language for data presentation, it offers a very static experience to the user and requires many round trips with the server. Besides, since it relies on the web browser, it is limited in terms of what controls can be used and the level of functionality that can be supported, given that mobile browsers are cut-down versions of their desktop web browsers. More specifically, we have identified a number of missing features in TERESA as far as the mobile experience is concerned:

1. **No support for styling**: The generated user interfaces do not support styling themes, which limits the possibility to easily change the look-and-feel of an application's UI interface. We believe that the ability to support styling is important to create appealing and personalised user interfaces.

2. **Lack of support for soft keys and menus attached to the keys**: Due to the size limitation of mobile devices, there is a need to maximise the utility of the keys on the device, and increase the functions attached to them. For these reasons, mobile manufacturers and mobile operation systems developers support additional keys (e.g. softkeys, joystick, jog dial, etc.) which are used to provide quick access to special functions on the device and allow navigation through the GUI's widgets. These keys are generally placed on the left (LSK), right (RSK), and centre (CSK) of the display and various functions can be mapped onto them depending on the modality of the application. This requires the use of a GUI rendering engine that supports softkeys instead of relying on hardcoded key mapping used by the web browser.

3. **Lack of support for run-time adaptation of the GUI**: Since TERESA has been limited to generating static user interfaces, the user experience remains static throughout the usage.

4.3.6 Final UI Description
cycle. Modern, client-based, mobile GUI engines support run-time adaptation, in particular those that are based on XML. However, the current version of TERESA does not support them yet.

4. **Limited support of scripting:** Due to the limitations of mobile web browsers in terms of scripting language support, TERESA does make use of such functions when generating UI's for mobile. This constraint seriously limits the functions that can be activated from the UI (e.g. call for client-based function like calling), and reduce its dynamic behaviour (e.g. verify data input as the user types). We believe that such features are particularly useful on mobile phones since they offer better integration with device capabilities and support user's activities. This could be achieved by enabling the linking of actions to CUI's such as those of type activator (e.g. button, button_with_script), or navigator (e.g. link).

Alternatives to HTML include XML-based high-level descriptive languages such as XForms, UsiXML and XUL. However, XForms and UsiXML lack interpreters on mobile devices, whereas XUL can be natively supported on mobile devices using dedicated renderers. On the other hand, based on a review by Souchon and Vanderdonckt [74] of a number of XML high-level UI description languages, including XUL, where they evaluated the languages with respect to various criteria: the abstraction level supported (at the instance, model or the meta-model), number of XML tags used, expressivity (ability to express real world concepts, and also ease and usability), openness (denotes whether the expressed concepts and tags are fixed or user editable) and coverage introduced/used (depending on the level of abstraction). They argued that XUL addresses some requirements of supporting multiple platforms, but was not sufficiently expressive to be considered as a complete UIDL. It is true that the specification of XUL have stagnated since its inception in 2001, and we acknowledge the fact that it has limited expressiveness. But we believe that there is a practical reason for it: XUL, by design, is used as a description language for the final UI, and not for the description of the concrete presentation, so it is rather beneficial to have a relatively low level of complexity and expressiveness to ease the creation of user interfaces. Besides, XUL is an open standard, and has a number of freely available editing and rendering libraries. In addition, in comparison with HTML-based browsers, XUL-based rendering engines, such as Thinlet\(^\text{12}\), carry a number of advantages, such as richness of XUL's widget set, its styling support and its ability to reload the UI at runtime. XUL is one of the two XML languages that will be used to implement an application demonstrator in section 5.4, the other one being a closely related language that is used by another mobile GUI rendering engine, Kuix\(^\text{13}\).

**4.3.7 Context Adaptation Model**

We have defined the UML representation for the adaptation model in section 3.4.5, and in this part we present the serialised version which can be used with an adaptation middleware. It is the result of mapping of Class stereotypes and their tagged values to XML tags and their attributes. For reference, the full schema of the adaptation model is given in appendix A.4 in the form of an XSD file.

Each adaptation rule is specified with the `<rule>` tag, which in turn contains two main child elements, `<condition>` and `<action>`, representing the set of actions and their corresponding actions.

\(^{12}\)http://www.thinlet.com

\(^{13}\)http://www.kalmeo.org/projects/kuix
4.3. Proposed Approach

Furthermore, the rule priority can be specified within the rule by the priority parameter, and its impact level by the level tag. The condition describes the context situation that needs to be fulfilled for the action to be carried out.

A context condition is represented as a relational statement made up of two operands and an operator (defined with the type parameter). An operand can be a simple context element which is described with a name and value, or more complex context element with multiple parameters (specified with a series of ctxParameter and value). Simple conditions are formed by relating context parameters to context values, using arithmetic operators (< represented with lt, = represented by equal, != represented with notEqual and > represented with gt), whereas a complex condition is formed by relating simple context conditions (or nested complex context conditions) using a logical operation (and, or). Not that currently nesting of complex conditions up to two levels is supported.

In the case of nested conditions, the inner conditions are evaluated first, followed by enclosing conditions. If the top-most condition is evaluated to true, then the corresponding set of actions is executed (specified between two actions tags). The latter is made up of a series of actions specified with the levelAction tag and parameter level to indicate the UI model on which the action is performed. An action is made up of two parts: the change part, which details the operations and their corresponding parameters to be executed (using the param tag) at the specified model, and the elementList, which contains the list of task nodes (indicated with the element tag) affected by the change part. To enable the structural, behavioural and presentation style adaptations, as described in section 2.7.9.3, two types of transformations are supported:

1. changeAttribute: It is used to set an attribute of a task node (or list of nodes) to a specific value (or values). It has two associated parameters, first one specifies the name of the attribute to be changed e.g. active, platform, category, type, name, etc. The second parameter contains the value to which the parameter should be set. It is possible to have multiple attributes to change simultaneously, by alternating the attribute name and value for each of the parameters that are effected. This action can also be applied at the concrete presentation level, by providing task nodes’ names that correspond to the target presentation units or objects, so that their attributes can be modified in a similar fashion that the task nodes are. The syntax used to specify this type of transformation is shown below:

   <param>parameter_name1</param>
   <param>parameter_value1</param>
   <param>parameter_name2</param>
   <param>parameter_value2</param>

2. changeStructure: It is used to specify the actions which will affect the overall structure of the tree, either by adding, sorting, or removing a task or a set of tasks. The tree operations are: sort, add and delete, and are only applicable to the task model. The syntax used for each type of structural change will be presented in a separate section below, after explaining how the condition part of an adaptation rule, regardless of it type, is validated and evaluated.

4.3.7.1 Condition validation and evaluation

For each context in the condition part of the adaptation rule, it is compared with the current context profile. If the context name exists, then it retrieves the list of affected levels indicated in the action
part of the adaptation rule, and compare it with the list of models attached to the corresponding context element in the context profile. If the former is a subset of the latter, then the condition is considered to be valid and evaluated accordingly, otherwise, the next condition is checked. In the case where a condition has one or more context elements which are not found in the context profile, the associated condition is ignored. A flow chart showing this process is depicted in Figure 4.2.

![Flow chart for the validation adaptation rules](image)

**Figure 4.2: Flowchart for the validation adaptation rules**

When there is a change in the context, the device sends a differential profile where only the parameters that have changed since the last version are included. In this case, the condition checking process is executed again to take into consideration the latest context information. However, adding new context parameters to the context profile while the application is used is not taken into account during adaptation.

For practical considerations, we have limited condition nesting to two levels i.e. the top-most condition can contain a set of conditions, and the latter can contain simple conditions (made up of a single or multiple parameters). The main condition may have a logical or numerical operator and the inner ones use only numerical operators. We believe that having these simple constructs and condition guidelines are sufficient to express a large set of contextual conditions using a combination or logical and numerical operators. In addition, if needs be, more complex logical conditions can be
emulated by having separate conditions listed with the same actions parts, so that different actions can be activated for the same set of contextual conditions.

Note that there is no particular requirement on the order in which the parameters of the adaptation conditions are listed. Furthermore, conditions can be specified in various ways to simplify the work of the UI designers. The following statements are valid:

\[
\text{<condition><ctxparameter...><ctxparameter...>}
\]
\[
\text{<condition><ctxparameter...><condition><ctxparameter...>}
\]
\[
\text{<condition><ctxparameter><ctxparameter...><condition><ctxparameter...><ctxparameter>}
\]

### 4.3.7.2 Task model structural operations

Looking more closely into the `changeStructure` transformation type, the following operations are applicable onto the task model:

1. **Sort:** Sort the list of tasks as specified in the `<elementList>` according to an attribute specified in as a parameter. This operation only applies to siblings which are related with a choice operator (`|= |`).

2. **Add:** Add a task tree as a child or sibling of the task specified in the `<elementList>`. The first parameter indicates the number of tasks to add, and the subsequent parameters define task's attributes and their corresponding values. The first parameter specified is the type, which indicates the relationship between the newly created node (or subtree) and the listed task node. The type could be set to `child`, `left` (sibling) or `right` (sibling). The following example illustrates how one task is created with name and category parameters set. This task is inserted as the right sibling of a task with name `ref_task`.

   `<elementList`
   `<element>ref_task</element`
   `</elementList`

   `<param>1</param>`
   `<param>type</param>`
   `<param>right</param>`
   `<param>name</param>`
   `<param>task_name</param>`
   `<param>category</param>`
   `<param>user</param>`

The parameters that are supported to create a new task node are summarised in the table below. Only the name, category and relation with the reference task (i.e. child, right or left sibling) are mandatory.

When a task node (or subtree) is added as a child, it is inserted at the right-most end of the children (if any), and its left sibling adds a default temporal relation set for enabling (`>>`) if not specified in the action section with parameter `link`. A node can be added as a right sibling, where the previous relationship relates the node on its left with the newly added node. If the new node has a sibling on the right than the link needs to be specified, otherwise the
4.3. Proposed Approach

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String</td>
<td>Unique identifier of the task</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the task</td>
</tr>
<tr>
<td>category</td>
<td>String</td>
<td>Abstraction/User/Interaction/Application</td>
</tr>
<tr>
<td>iterative</td>
<td>Boolean</td>
<td>Set task to be iterative</td>
</tr>
<tr>
<td>optional</td>
<td>Boolean</td>
<td>Set task to be optional</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>Description of task</td>
</tr>
<tr>
<td>platform</td>
<td>String</td>
<td>Family of devices supported</td>
</tr>
<tr>
<td>link</td>
<td>String</td>
<td>Specify the temporal relationship between that task and the tasks specified in the <code>elementList</code></td>
</tr>
</tbody>
</table>

Table 4.3: Parameters required to add a new task

Default enabling relationship is used. If there is no sibling on the right then no link is added. When a task node (or subtree) is added as a left sibling, the previous relationship relating the left node to its immediate right sibling is maintained if set, otherwise no link is used. In this case, a new link needs to be defined between the new node and the one on the right. The three different types of operations are shown in Figure 4.3.

![Diagram](https://via.placeholder.com/150)

(a) add a child node

(b) add a right node wrt $T_2$

(c) add a left node wrt $T_3$

Figure 4.3: Illustration of the task-level 'add' operations

It is possible to add multiple children to a single task by having the set of individual parameter sets separated with a parameter that specifies their relationship to the listed task node, as shown below. In this example, three task nodes are added to `ref_task`: first one as right sibling, second one as left sibling, and last one as child.

```xml
<elementList>
<element>ref_task</element>
```
4.3. Proposed Approach

It is also possible to add tasks by referring to an external task specification written in Tera-ssaXML notation. In this case, the new task model is inserted as a sub-tree. The first parameter indicates how the task subtree is added (child, right sibling or left sibling) and the following parameter indicates the name of the source file.

<param>3</param>
<param>type</param>
<param>right</param>
<param>name</param>
<param>task_name1</param>
<param>category</param>
<param>user</param>
<param>type</param>
<param>left</param>
<param>name</param>
<param>task_name2</param>
<param>category</param>
<param>application</param>
<param>type</param>
<param>child</param>
<param>name</param>
<param>task_name3</param>
<param>category</param>
<param>interaction</param>

3. Delete: Delete an individual task or a number of tasks from a given task node specified in the <elementList>. The latter indicates the name of the task from which the search starts. The first parameter indicates the number of tasks concerned, the second one indicates the type of deletion (either individual i.e. single, or as subtree attached to the task i.e. subtree), the third parameter indicates the name of the task (or subtree root node) to be deleted. The deletion type and task name are inserted for each task or subtree that is subject to deletion. Example is shown below, where a single node and a subtree are deleted from ref_task. Since the deletion of a task/subtree involves searching for the target node, it is important to carefully select where to search from, especially if there is more than one task with the same name. Naturally, if no matching task is found or the node-to-be-deleted is the one specified in <elementList>, then no node is deleted.

<elementList>
<element>ref_task</element>
</elementList>

<param>2</param>
<param>single</param>
<param>task1</param>
<param>subtree</param>
4.3. Proposed Approach

4.3.7.3 Specification of the initial task model

One of the challenges of applying structural changes to the task model is to determine how to setup the initial task model. For this, we need to review what modifications can be applied to the task model in response to adaptation requirements. Three types of changes can be applied to the initial task model:

1. New tasks are added to the initial task.
2. Current tasks are removed from the initial task.
3. New tasks are added and other tasks are removed at the same time from the initial task model.

In the first case, there is no particular requirement on the initial task model. In the second and third cases, there is a need to define the initial task model in such a way that it incorporates all tasks related to an application to make it fully functional, so that some tasks can be altered or removed if contextual conditions become more constraining. Since mobility dimensions are more likely to impose limitations on the functions of an application and its user interface, such as degradation of network, it is reasonable to assume that the number of enabled interactor widgets on the user interface is more likely to decrease than increase as a result of a change of contextual conditions. Since these widgets are concrete representations of the initial nodes found in the task model, it implies that the initial task model should contain all the tasks nodes resulting from the analysis of static use cases, as well as those resulting from the analysis of context-sensitive use cases.

As much as the careful specification of the initial task model is important, the extent to which task deletion is used needs to be limited. This is because in some cases, the application of adaptation rules on a task model can make the resulting task model unusable, halting any further transformation of the model, and hence no final user interface can be generated. For illustration, there are two occasions when a tree is considered invalid in CTT: when a node has a sibling on the right but is not linked to it, and when a node has only one child (Figure 4.4).

By introducing extreme structural changes with the deletion of some task nodes, it is likely that the user interface cannot be generated. For this reason, it is important for the UI designer to understand the implications of the adaptation rules applicable to the task model, and avoid formulating rules that may result in unusable models.

4.3.7.4 Task level attribute operations

Most task node parameters are modifiable using the adaptation rules. The task attributes are changed by specifying the parameter name followed by its new value. The latter overrides the old one whether the parameter has been initialised or not in the task model. The list of supported parameters includes: id, category, iterative, optional, frequency, name, type, description, platform\(^\text{14}\) and link.

\(^{14}\text{The list of supported platforms is specified by setting any of the following parameters to true: mobile, desktop and mmpda (for multimedia PDA). To add a new type of platforms, the new platform parameter is used instead.}\)
4.3. Proposed Approach

(a) Single child node

(b) Not linked sibling

Figure 4.4: Impact of applying adaptation rules on a task model

(which represents the temporal relationship). Note that in the case of parameters with boolean data type, such as optional, the associated values need to be set to false or true to be applied. In the following code excerpt, the category and description of ref_tasks are modified. To indicate that a new platform is supported, the parameter new_platform is set to the new family of devices, in this case smartphone. Note that although temporal relationships (through the link parameter) can be overridden with the adaptation rules, it is not advised to do it since it may result in an non-valid task model, which cannot be transformed to final UI's.

<elementList>
<element>ref_task</element>
</elementList>

<param>category</param>
<param>application</param>
<param>description</param>
<param>this is task description</param>
<param>new_platform</param>
<param>smartphone</param>

4.3.7.5 Concrete presentation level attribute operations

At the concrete level, only attribute-level changes are supported, and these transformations are only applied after the transformation rules have been applied onto the task model. However, the name of the concrete UI presentations and interactor objects are made up of the task node name combined with a numerical suffix (e.g. presentation_4 or interactorobject_3), which indicates its order within a composition of interactors (i.e. grouping, ordering, relation and hierarchy). The ordering of interactors and presentations depend on how tasks are grouped together during the transformation
4.3. Proposed Approach

from task model to presentation model. It is not reasonable to expect UI designers to anticipate the effect of operators on the presentation units and interactor objects and accurately predict their ordering.

To minimise errors and facilitate the work of UI designers, an internal mapping between the task nodes and their abstract representations is created during the production of the presentation task sets (PTS). The latter is effectively the dialog model which reflects the way tasks are grouped together and the way transitions occur from a presentation unit to another when a particular interactor is used (for instance, the navigator AIO). This information can be exploited to retrieve information about the correspondence between task nodes and AIO’s. Then since AIO’s and CIO’s share the same structure (and names), it becomes possible to use a list of task nodes in the <elementList> to point to their concrete equivalents, and change their attributes accordingly.

Similarly to the task level changes, the attributes to be changed and their new values are indicated using the <param> tags. However, unlike task nodes which have a fixed set of parameters, CIO’s are more diverse in nature and their attributes change from one to another. For this reason, we present in Table 4.5 the list of the most common parameters which can be used in the adaptation rules, in addition to their supported values in TERESA. Note that it is also possible to change the default background colour and font settings associated with most concrete presentation interactor objects. This indirectly enables changes to their visual style once they are converted into final UI widgets.

For each interactor object, the list of acceptable values for each parameter is set in the concrete TeresaXML schema. It is however possible to specify a parameter value outside the default range using adaptation rules. For instance, the default font size of a Text Edit interactor could be set to 12pt, or that its default text length increased to 12, or that the font name of a Navigator interactor to Verdana. Furthermore, it is possible to attach new parameters to existing interactors. For instance, adding a width and height parameters to Image Navigator interactor to set image dimensions, or adding an action parameter to it to enable additional functions when it is activated. The advantage having a choice to what to set existing parameters and ability to add extra ones, is that it frees the designers from the limitations of TERESA in terms of richness of information associated with the different interactor objects, and also enable the adaptation of the concrete presentation model to suit the capabilities of the final UI renderer. An example of how these two extension mechanisms will be demonstrated in section 5.3.4.

However, the two disadvantages of extending the syntax and semantics of concrete interactors are: 1) the need to modify the concrete TeresaXML schema which results in the inability to use it with the standard editing tools, and 2) the need to implement a specialised software that post-processes the resulting concrete presentation model and map the new parameters to final UI widgets' attributes.

Finally, it is worth noticing that given the wide choice of objects and parameters, the creation of adaptation rules for the concrete presentation model is somewhat more complex than that for the

15The PTS are calculated internally by TERESA and are presented in an XML format. This process is a necessary stage in the conversion from a task model to an abstract presentation model. PTS are TERESA’s equivalents of Enabled Task Sets (ETS).

16Keeping to this list values ensures that the resulting concrete presentation model is compatible with the original schema. This could be important if compatibility with editing tools is deemed necessary.
### 4.3. Proposed Approach

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
<th>Description</th>
<th>Used in</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>font_name</td>
<td>Arial, Courier, Times, New_Roman</td>
<td>font type</td>
<td>Default font settings</td>
<td>Any name can be set but the parser will reject it</td>
</tr>
<tr>
<td>bg_color</td>
<td>colour name or hex representation</td>
<td>background color in default Background</td>
<td>Default background colour</td>
<td></td>
</tr>
<tr>
<td>font_size</td>
<td>9 pt, 10 pt, 11 pt</td>
<td>font size</td>
<td>Default font settings</td>
<td>Any value can be set but the parser will reject it</td>
</tr>
<tr>
<td>font_color</td>
<td>colour name or hex representation</td>
<td>font colour</td>
<td>Default font settings</td>
<td></td>
</tr>
<tr>
<td>font_align</td>
<td>Left, Center, Right</td>
<td>default text horizontal alignment</td>
<td>Default font settings</td>
<td>Only Left is support by default</td>
</tr>
<tr>
<td>label</td>
<td>string</td>
<td>label attached to UI object</td>
<td>Activator, Navigator (Link, Button), Selection, Edit</td>
<td></td>
</tr>
<tr>
<td>script</td>
<td>string</td>
<td>script attached to activator</td>
<td>Activator</td>
<td>Can include method call or method definition</td>
</tr>
<tr>
<td>enabled</td>
<td>true, false</td>
<td>UI Object inability</td>
<td>Activator, Navigator (Link)</td>
<td>Default is disabled</td>
</tr>
<tr>
<td>image_src</td>
<td>path to image file</td>
<td>URI to image file</td>
<td>Navigator (Image), Description (text)</td>
<td>Relative path</td>
</tr>
<tr>
<td>image_alt</td>
<td>string</td>
<td>alternative text</td>
<td>Navigator (Image), Description (text)</td>
<td></td>
</tr>
<tr>
<td>text_src</td>
<td>path to text file</td>
<td>URI to text file for output</td>
<td>Text (file), Description (file)</td>
<td>Relative path</td>
</tr>
<tr>
<td>text_value</td>
<td>string</td>
<td>text for output</td>
<td>Text (text), Description (text)</td>
<td></td>
</tr>
<tr>
<td>text_length</td>
<td>4, 5, 6, 7, 8, 9, 10</td>
<td>length of edit text field</td>
<td>Text Edit and Numerical Edit</td>
<td>Any value can be set but the parser will reject it</td>
</tr>
<tr>
<td>password</td>
<td>true, false</td>
<td>set text input as password field</td>
<td>Text Edit and Numerical Edit</td>
<td></td>
</tr>
<tr>
<td>sel_card</td>
<td>low, medium, high</td>
<td>cardinality of password field</td>
<td>Single and Multiple Selection</td>
<td></td>
</tr>
<tr>
<td>sel_align</td>
<td>horizontal, vertical</td>
<td>horizontal alignment of selection</td>
<td>Single Selection</td>
<td></td>
</tr>
<tr>
<td>sel_type</td>
<td>radio_button, drop_down_menu, checkbox</td>
<td>sub-type of Selection</td>
<td>Single and Multiple Selection</td>
<td>Relevant to Single Selector only</td>
</tr>
</tbody>
</table>

Table 4.5: Examples of parameters for the adaptation rules
4.4 Conclusion

In this chapter, we have presented the various XML-based notations that have been used or defined to offer a serialised version of the models that have been previously described in UML and CTT. XML has proven to be adequate for this undertaking as it offers many advantages, such as formalism, extendability, ease of processing and platform independence. As part of our contributions, we have developed an XML schema for the context and adaptation models, and have explained in details how UI designers could use them to capture context information and formulate adaptation rules respectively. We have in particular shown how adaptation rules could be used as a powerful mechanism to modify the structure as well as the attributes of task and concrete presentation models, effectively enabling a very granular adaptation of the final UI. Noticeably, the effect of those rules were not limited to changing the characteristics of existing interactor objects but also helped in enriching their specifications to match the capabilities offered by the final UI renderer.
Chapter 5

User Interface Generation and Adaptation

5.1 Introduction

While the previous chapters provided details on the design and modelling aspects of the user interface (UI), this chapter discusses the architectural aspects of the UI management system and shows how the various UI models can be used in practice to create a context adaptive UI's for resource-constrained devices. This chapter is divided into six main parts. We start by describing a reference UI management system, including list of recommendations which can guide developers during the implementation and deployment stage, and explain the implementation choices made for rendering the final UI. This is followed by a description of the modifications introduced to the default processes of TERESA runtime to support the novelties described in the previous chapters, with particular focus on the semi-automatic UI generation and the context adaptation mechanisms. We will then provide detailed presentation of one mobile application that was implemented and another prototype application to show the different steps needed to turn a scenario into a set of requirements, then to models' specifications, which can then be used to generate the final UI and enable its adaptation. We also highlight the issues encountered during this process and offer some recommendations to UI designers. We conclude this chapter by an evaluation of our approach and a discussion on the modelling notation, and the UI generation and evaluation process.

5.2 The UI Management System

5.2.1 Recommended UIMS Architecture

The UI Management System (UIMS) is responsible for providing the different services needed to generate user interfaces from abstract models and adapting them to the context-of-use. The UIMS in mobile conditions is more complicated than in desktop-type conditions since the conditions are subject to more frequent changes, in addition to limited resources. Hence the need to integrate context information in this process while deploying a distributed architecture.
5.2. The UI Management System

After a review of the different software patterns to achieve the separation of concerns needed in an environment where UI functions and layout depends on various context-of-use concerns, namely MVC, PAC and Transformation-based techniques, we recommend the use of a combination of PAC and transformation techniques, to encapsulate information and behaviour about the interaction with the user that are independent of the final UI and for the inter-model transformations to specialise the UI according to the current context.

The architecture is based on a distributed topology, whereby the UI-related processes that manage context information, generate an abstract UI and transform it to the final UI reside on one system, the abstraction of the core functionality of the service/application is another process on a different system, and the presentation of the final UI for user interaction in yet another system. In addition, since these processes are distributed, there are also components that facilitate messaging among all of the components. This subdivision separates the concerns of the functional implementation of the service given to the users, the interaction with the user through the user interface, and the variations in the UI depending on the context of use.

The benefits of this architecture is the separation of concerns between the internal implementation of the business and the implementation of the UI. It also enables the reuse of components, better scalability by distribution of processes, and easier code maintenance. Furthermore, the separation of concerns between the mobility dimensions and interface-specific modalities allows the reuse of the transformation components in other systems and simplify the support for other modalities.

There are different ways in which this architecture can be implemented and the technologies which can be deployed for that. For this reason, the implementation decisions are left to the developer. However, we recommend the use of open systems such as HTTP and web-services since they are widely used and supported in the mobile world. It is however important to note that possible consequences of adopting a distributed approach include degradation in performance due to additional layer of abstractions and context processing, and inherited complexity due to the distributed nature of the processes.

Assuming a client-server architecture, Figure 5.1 provides more details on the processes, data flows and processing units involved in enabling a context-aware interactive UI generation and adaptation [96].

![Reference UIMS architecture for context-aware adaptive user interface generation](image)

Figure 5.1: Reference UIMS architecture for context-aware adaptive user interface generation
5.2. The UI Management System

The reference architecture is divided into a client-side, represented by the user and the interaction device, and a server side, which comprises the back-end logic and UI modelling, design and processing. Some components of the architecture need to be specified at design time such as adaptation rules and the different default abstract models; and others, such as context model, are composed at runtime. Looking more closely into the framework, it is composed of the following components:

- **Adaptation rules**: They form the link between the contexts-of-use considered by the designer and its effect on the UI models. They are specified at design time.

- **Context adaptation server**: This is a storage place for context profiles received from the end-user device and immediate environment. Context server gathers user context, platform context, and environment context. Actions on the different UI models are activated when the gathered (and processed) context matches one of the contexts in the context-adaptation specification.

- **UI model server**: This is where the UI-specific models are stored and updated. It is also responsible for applying the adaptation rules onto the task and concrete presentation models, and activate the inter-modal mapping until a final UI is generated.

- **Application server**: This is only considered in the case of a distributed application whereby the application server provides the content to the user interface. The server is responsible for creating the bindings between the interactor objects on the user interface and the service back-end.

5.2.2 Implementation Considerations

5.2.2.1 Java Mobile Edition (JME)

Java is undoubtedly one of the most open and complete solution to application programming, on desktop as well as mobile devices. In addition, SUN - the creator of Java - has also created the Java Community Process (JCP) (http://jcp.org), which allows Java to continually evolve. Within this process, various vendors propose and produce recommendations on the evolution of the platform and the API's in the form of Java Specification Requests (JSR) (http://jcp.org/en/jsr/overview). Java ME (Mobile Edition), which is the mobile version of Java, also benefits from this process. However, this process is known to be slow to take in new innovation in software development, which may penalise evolutions of JME and other Java-related technologies. SUN has recently decided to open source the JME runtime in the hope to get more third party developers involved in the specification processes. However the impact has been smaller than anticipated by SUN. Nonetheless, Java remains the dominant force in mobile development and offers the most vendor-neutral solution for mobile application development.

SUN's Java ME represents a very matured software development platform for mobile applications that has been adopted by millions of developers to create a variety of applications (known as MIDlets). In addition, SUN Microsystems estimates that there are more than 2.6 billion handsets that run Java ME worldwide\(^1\), which means that 8 out of 10 handsets shipped in 2008 were running Java technology, spanning low-end feature-phones to high-end smartphones. More specifically, we believe

\(^1\)http://www.sun.com/aboutsun/media/presskits/2009-0212/index.jsp
5.2. The UI Management System

that Java ME is adequate to create a client-server architecture for the development of adaptive UI's since it is network-ready and provides a number of libraries to process XML documents.

Java ME tackles the problem of proliferation of devices by implementing two separate profiles. The most common of these are the Mobile Information Device Profile (MIDP) aimed at mobile devices, such as cell phones, and the Personal Profile aimed at consumer products and embedded devices like set-top boxes and PDA's. Profiles are subsets of configurations, of which there are currently two groups: the Connected Limited Device Configuration (CLDC) which provides the most basic set of libraries and virtual-machine features that must be present in each implementation of a JME environment, and the Connected Device Configuration (CDC) which is richer than CLDC. However, both CLDC and CDC do not specify how the GUI should be rendered on the device, and relies on the mobile operating system to do so. JME CLDC is the configuration that best suits resource-constrained devices and it is the minimum supported by the majority of current mobile handsets. The extent to which a JME application can support the different mobility dimensions, which have been outlined in section 2.2.1, is explained below:

1. **Location Awareness:** Location awareness has been added in JSR 179 (Location API) as an optional package to build on top of CLDC 1.1 and higher. This JSR is intended to work with various positioning techniques e.g. GPS, cell-based triangulation, but the API will be agnostic to the method of finding the location.

2. **Network Quality-of-Service:** JME connection framework is able to create any type of connection (TCP, UDP, HTTP), and network providers and device vendors can provide their own API's on the device. However, it is still not possible to access information about the signal strength from within a mobile application written in Java.

3. **Limited power supply management:** This is not currently possible since MIDP cannot access native device capabilities, since by design Java favours application portability over access to low-level features.

4. **Support for multiple modalities:** The focus of JME has always been on developing graphical UI's, and it currently lacks standard libraries to support other modalities such as touch screen and voice.

5. **Cross-platform development:** JME is widely supported on a large number of handsets and integrated in a number of mobile operating systems, as opposed to other development platforms such as Nokia's Symbian, Microsoft's Windows Mobile and Apple's iPhone. As such, it is possible to write an application once and deploy on a number of platforms.

From the above list, we remark that JME offers supports for a number of contextual dimensions, and more API's and libraries are constantly in development, through the JCP process and by third parties, to support wider range of context sources and improve interaction and usability. For instance, the latest version of Java ME incorporates new set of API's such as Location API (JSR 293) for location-based services, which add support for geo-coding, support maps given from 3rd parties, and also navigation support. There are also other JSR's being standardised such as: UI customisation (JSP 258), Java Language & XML User Interface Markup Integration (JSR 290) and Advanced Graphics and User Interface (JSP 209), however these standards are still at the draft level.
5.2.3 Java ME and XML Support

As far as XML processing is concerned, JME provides three types of parsers:

1. **DOM Parsers**: These parsers go through the entire XML document and create a DOM (Document Object Model) representation of the document. Since the entire document is parsed and represented in some format in memory, it is memory and processing intensive, hence they are better suited for small size files.

2. **Push Parsers**: These parsers emit events as they parse through the document. While they go through the entire document, they do not keep a representation of the document in memory. JME provide a specific library for this task as part of JSR 172, which is used to access web services. This provides for a lightweight API that is suitable for use on resource-constrained devices.

3. **Pull Parser**: These parsers do not go through the entire document, but only return data when they are asked to read the next node in a document. For instance, the kXML XML library (http://kxml.sourceforge.net/) has a simple interface allowing for a mobile application to do the necessary work without loading the entire document in memory.

Selecting the parser is somewhat a balancing act that often requires some knowledge of the average and maximum size of documents. For speed, DOM parsers are the best but they require more memory and processing power, on the other hand, push and pull parser are lightweight solutions that are suitable for resource-constrained devices. For the purpose of implementation, we recommend the use of DOM XML parsers, as parsing and adaptation speed are crucial in this case where UI fast generation and responsiveness are important.

5.2.4 Java ME Graphical UI Rendering Engines

Due to the accelerating pace for the development of mobile devices and applications, different graphical user interface were developed to make efficient use of device capabilities. Some graphical development libraries are built specifically for particular category of devices (e.g. Blackberries), class of operating systems (e.g. Symbian, Linux, Mac OS X, Android), whereas others attempt to run on cross platform software platforms. In line with our objective to enable the widest possible support for our approach and develop a demonstrator that can run on virtually all mobile devices, we have decided to consider only the graphical UI development environments that are supported by Java ME.

Concerning the user interface, JME does not explicitly define GUI functions, instead, the GUI classes for the JME are included in profiles such as the MIDP 1.0 and 2.0 profile. The MIDP GUI consists of both high-level and low-level user interface components, each with their own set of events. The low-level API's allow the developer to define abstract GUI components but give little control on the look and feel, to maximise portability. The major two classes are Canvas and Graphics, and they are suited to create games for instance. It leaves rendering to the underlying implementation which adapts to the hardware and native user interface widgets. The low-level API is designed to finely control the placement of the graphic elements, as well as access to low-level input events, but it is device-dependent. The high-level API's (subclasses of the Screen and Item classes) allow high-level GUI components (e.g. text fields) to be rendered by the mobile device, either with native look-and-feel or as preset by the operating system.
The next version MIDP 3.0 will be geared towards a richer mobile experience, and requires that mobile devices support at least 64k colours with screen size above 176 x 220 pixels. In the MIDP 3.0 specification, there have been several additions to the high-level API's including the ability to render animated images and a mechanism to create tabbed panes. In addition, MIDP 3.0 enables the running of multiple midlets at the same time, which can communicate with each other. That reinforces the idea that Java ME is a very mature software development platform which is currently widely supported and is set to continue to integrate more features as interaction technologies evolve and device capabilities improve.

Many graphical Java ME API's from third parties are available allowing developers to create more advanced user interfaces that contain vector graphics as well as 3D objects. A review of the currently available mobile GUI frameworks on JME is presented in Appendix D. Out of these GUI frameworks, Table 5.1 provides four open source frameworks that are built on the top of JME: Thinlet\(^2\), JME Polish\(^3\), Kuix\(^4\); and four commercial development frameworks: SolidForms\(^5\), TagsMe\(^6\), Picoforms\(^7\), DataMovil\(^8\). They are compared in terms of the base language used to describe the UI's, support for events, CSS (cascading stylesheet), license terms and also whether the UI can be changed at dynamically and whether customised widgets can be added programmatically (Table 5.1).

<table>
<thead>
<tr>
<th>Name</th>
<th>UID</th>
<th>Events</th>
<th>CSS</th>
<th>Runtime</th>
<th>Custom</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thinlet</td>
<td>XUL</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>open source (OS)</td>
</tr>
<tr>
<td>JME Polish</td>
<td></td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>open source (OS)</td>
</tr>
<tr>
<td>Kuix</td>
<td>XML</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>open source (OS)</td>
</tr>
<tr>
<td>TagsMe</td>
<td>XML</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>commercial</td>
</tr>
<tr>
<td>SolidForms</td>
<td>XForms</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>commercial and OS</td>
</tr>
<tr>
<td>Picoforms</td>
<td>XForms</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>commercial</td>
</tr>
<tr>
<td>DataMovil</td>
<td>XForms</td>
<td>yes</td>
<td>N/A</td>
<td>no</td>
<td>no</td>
<td>commercial</td>
</tr>
</tbody>
</table>

Table 5.1: Open source JME-compatible GUI libraries

JME Polish is a suite of tools and technologies aimed to ease and speed up the development of mobile applications with richer user interfaces than what is achievable with standard Java ME. GUI's are defined by using standard JME classes from java.microedition.lcdui in addition to custom components, then styling them with an external CSS file or with the #style pre-processing directive in the source code. However, it does not support a descriptive language. As it has been noted in section 4.2.2.4, XForms support on mobile devices is limited despite the advantages they offer to form-based interfaces. Currently, there are two commercial implementations of XForms processor for MIDP devices, and one free implementation which dates back to 2006. As a result of the limited support for mobile devices and lack of a free and open UI renderer, we have decided to not use XForms for the demonstrator.

The remaining XML-based GUI development frameworks include TagsMe, Thinlet and Kuix. The former is highly inspired by the web paradigm and related technologies (e.g. HTML, JavaScript).
The TagsMe framework enables different deployment of mobile applications, which allows a variety of infrastructure configurations with different degrees of automation and levels of distributions between the client and the server.

A typical TagsMe application is composed of *modules* which represent simple screens on the device, which contains all interactive elements. The application is built by connecting the modules and allowing the user to browse between them. It targets MIDP 2.0 including touch-screen devices. Default screen dimensions are specified at design time but the UI objects can be resized and repositioned if screen resolution is different from design. It is strongly geared towards graphics support, animations and multimedia. Furthermore, it supports a large number of events depending on the UI objects used. The UI descriptions (written in XML) can be stored locally, on the mobile device, or accessed remotely on a server. For performance considerations, it supports caching and preloading of images for the UI. Finally, TagsMe implements a simple scripting language which can be called from within the XML file, analogous to javascript support within HTML pages. The application has a commercial licence, limiting the ways in which we can use it in our work. Besides, only the GUI builder application is available as a time-limited demonstrator, which makes it less suitable for development and evaluation purposes.

On the other hand, Thinlet and Kuix are open source software. They are lightweight: Thinlet library can be included into one file weighing 254 KBytes uncompressed and 39KB compressed, and Kuix' library is 176 Kbyte compressed. Furthermore, both toolkits support different layout managers for text and form-based widgets, styling (using CSS in Kuix' case) and event handling. CSS in particular is important since it allows a great level of customisation of the user interfaces, from colour to layout, which could help create a consistent experience across multiple devices and allow end user personalisation. Kuix is a relatively new addition compared to Thinlet, and has limited support for some data structures such as tables and rich text, lacks an XML schema, which makes it hard to validate the UI description and ensure that it can be rendered properly. However, it has a more advanced support for events generated by the UI widgets and has also the ability to update the UI at runtime and enable a more dynamic UI.

5.2.4.1 Thinlet

Thinlet is a XUL-styled UI development framework in which the UI description is separated from the business logic. Thinlet provides most graphical elements (buttons, text fields, lists, menus, etc.)\(^9\), and is compliant with CLDP 1.0/MIDP 1.0\(^10\). Thinlet uses a grid-based layout where component are placed in a grid of rows and columns, allowing some components to span multiple rows or columns. It uses the components' preferred sizes to determine how big the cells need to be. The process of creating a Thinlet application consists of three steps:

1. Create an XML description of the user interface.
2. Create a sub-class of Thinlet. Thinlet to handle UI events and provide the business logic of the application.

---

9The list of supported GUI widgets can be found in [http://thinlet.sourceforge.net/component.html](http://thinlet.sourceforge.net/component.html)

10The current version posted on the official website ([http://www.thinlet.com](http://www.thinlet.com)) is only available for J2SE, with a plan to support other platforms in the future. The last known MIDP version of Thinlet dates back to July 15th, 2005, where it was posted on the Sourceforge site ([http://thinlet.sourceforge.net/home.html](http://thinlet.sourceforge.net/home.html)).
3. Use the thinlet.AppletLauncher or thinlet.FrameLauncher method to render the UI by parsing a local XML description file.

Each of the GUI widgets should have a unique name, and set of properties. They can be specified using XML descriptions or created and manipulated programmatically from the source code, although the former approach is more suitable to maintain the separation of concerns between the business logic and the UI. As for visual styling, Thinlet does not support external CSS files, but one can specify the styling parameters programmatically (from within the source code) or inline with the description of user interface. The typical structure of a Thinlet application is depicted in Figure 5.2.

In terms of navigation, the user can utilise the number keypads and/or arrows to move from one widget to another, and softkeys can also be used to access shortcuts defined programmatically.

![Deployment diagram of a typical Thinlet application](image)

Figure 5.2: Deployment diagram of a typical Thinlet application

Events are tied to UI components via their action attribute which indicates the name of the method implementing that event handler. The value of the action attribute must be a public method of the Java class that handles the UI component. The event description can also contain parameters (in brackets, separated by comma or white space characters) to process them. This is particularly useful for instance to process form data and update the GUI accordingly. Other events are also supported, as shown in Table 5.2, where the list of events available to the most common widgets is provided.

Knowing which events can be triggered by the different graphical widgets and user actions, and those which can be captured by the application backend logic, helps in mapping the concrete interactor object to those widgets in such a way that the final UI takes full advantage of the dynamic features provided by the rendering engine.
5.2. The UI Management System

<table>
<thead>
<tr>
<th>widget / event</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>button</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>checkbox</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>togglebutton</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>combobox</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>textfield</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>textarea</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>spinbox</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slider</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>list</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>table</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tree</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>menuitem</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1) init, (2) focuslost, (3) focusgained, (4) action, (5) insert, (6) remove, (7) caret, (8) perform, (9) expand collapse

Table 5.2: List of events triggered by the most common widgets in Thinlet

5.2.4.2 Kuix

Kuix is an application development framework that provides most graphical elements needed to create the graphical user interfaces (GUIs). Kuix is compliant with CLDC 1.0/MIDP 2.0. Widgets are specified using a combination of XML for their attributes and CSS for their styling\(^\text{11}\). As far as the layout of the GUI is concerned, Kuix’ advantage over Thinlet is that the user interface is designed by describing constraints between elements, which are then positioned by the rendering engine at runtime depending on the device screen size.

Kuix implements a basic event processor that handles all events originating from graphical widgets and user actions, popups, etc. Java ME sends a message to the current widget or frame. The latter does not handle the event itself but push it into the event stack. This stack is polled regularly by a dedicated thread that process events. Once all events have been processed, the display is refreshed. Kuix also offers the possibility to dynamically change the value attached to a widget using the data provider, which is a special object that has the ability to provide the Kuix engine with dynamic values, by binding a reference (from the XML) to a variable value. Table 5.3 shows the list of events available to the most common widgets in Kuix.

\(^{11}\)The list of supported GUI widgets can be found in [http://www.kalmeo.org/files/kuix/widgetdoc/index.html](http://www.kalmeo.org/files/kuix/widgetdoc/index.html)
5.2. The UI Management System

<table>
<thead>
<tr>
<th>widget/event</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>button</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>checkbox</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>choice</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gauge</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hyperlink</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>listitem</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>menu</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>menuitem</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>popupbox</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>radiobutton</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>radiogroup</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>tabitem</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>textarea</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>textfield</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1) onaction, (2) onfocus, (3) onlostfocus, (4) onselect, (5) onunselect, (6) onchange

Table 5.3: List of events triggered by the most common widgets in Kuix

The structure of a typical Kuix application is depicted in Figure 5.3. When compared to Thinlet functionally, Kuix offers context menus and multi-level shortcuts (e.g. to exit, navigate to another frame, etc.) at the bottom of the screen, which are accessible using softkeys or key combinations defined programmatically.

Figure 5.3: Deployment diagram of a typical Kuix application
5.3 Changes to TERESA Processes

5.3.1 Introduction

The default transformation processes supported by TERESA have been presented in section 2.7.4. Despite the flexibility exhibited by the tool, as it is, it cannot provide mechanisms that are needed to support our approach. For this reason, there is a need to extend TERESA and add new processes. However, this task is made more complex by the fact that the different software modules that apply these transformations are not available as standalone software packages because they are all integrated into the tool. In addition, the lack of documentation further complicates the task of creating new processes that interact with the core TERESA modules. In the following sections, we will explain the changes made to each of the key transformation phases, and when relevant, how we implemented them.

5.3.2 Objectives

To support the new models introduced in our approach, we needed to augment the inter-model transformation processes already provided by TERESA to:

1. Leverage existing transformation processes. This is particularly applicable to the steps from task model to concrete presentation model.

2. Support for context information. The original TERESA tool only supported limited information about the specifications of the rendering platform such as support for colour and size of display, and this information was only acted upon during the transformation from task model to presentation model.

3. Support context-adaptation. The original TERESA tool enabled the filtering of task tree based on the target platform i.e. desktop, mobile, PDA, etc. There is a need for a finer and richer adaptation of the task model and concrete presentation model based on current context information.

4. Generation of advanced GUI. The original TERESA tool had support for HTML and VoiceXML based user interface as far the mobile platform is concerned. In our approach, we propose more advanced markup-language based rendering for the graphical user interface.

5. Support dynamic adaptation through automation. The original TERESA was a desktop application that was used primarily for UI prototyping, requiring the designer to open, click on menu items, saving files then re-opening them to finally generate the UI. In our approach, we have provided additional modules to enable semi-automatic adaptation and generation of the UI.

Since TERESA tool is not composed of self-contained modules, we have implemented software interfaces to its different modules. For instance, interfaces were created to activate the modules that are used to read the XML representations of UI models, and those responsible for model-to-model mappings. This enabled automating activities that would normally be manually carried out by the designer when using TERESA, such as loading an XML file into TERESA then applying the required transformation. In addition, when it became necessary to support new features, such mapping from
5.3. Changes to TERESA Processes

TeresaXML to XUL, new modules were implemented to post-process UI models. The breakdown of modules based on whether they are part of TERESA's core, new software interfaces or completely new modules is shown in Figure 5.4.

![Composition of TERESA transformation modules](image)

Figure 5.4: Composition of TERESA transformation modules

It is important to note that the implemented software interfaces and modules run from the command line and used a series of scripts. This is what allows the semi-automation of the processes. By creating separate modules, it becomes possible to tweak the new modules to suit the needs of the designers and plug-in a new processing module along the transformation path from task model to final UI. However, automating some of the transformation processes, takes away the instant visual feedback that the designer would get when using TERESA. In addition, activating these modules using a series of command line scripts undermines the ability of the designer to visually fine tune the mapping between the tasks and presentation interactor objects for instance. Shifting the control from the visual editor to the software modules means that designers would have to tweak the source code to change the way model are post-processed to suit their requirements, for instance, when there is a need to support a different GUI library. Nonetheless, we believe that this is an acceptable cost given that it takes advantage of the powerful internal processes of TERESA while extending them to support new models (i.e. context and adaptation), and GUI engines. In the following section, we will look at each stage of the transformation in more details, indicating the decision made to facilitate automation.

5.3.3 From Task Model To Abstract Presentation Model

For the generation of the dialog model, we have opted for the "Joining when Enabling" heuristic (out of four supported by TERESA, as described in section 2.7.6) to merge two groups of tasks if they share most tasks at the same level with the exception with one element which are connected with an enabling operator (Figure 5.5). This has the effect to minimise the number of ETS (enabled tasks sets) and hence reduces the number of separate presentation units in the user interface (and by extension, the number of 'screens' in the final UI). This is particularly useful for mobile applications where the display space is limited.

We have also limited this phase of transformation to support single-user tasks (in the task model) and not multi-user (or "cooperative") task where there are multiple users exchanging information and using various devices (and UI's) to interact with their respective system and with each other. While
cooperative tasks are applicable in desktop user interfaces, they are not relevant to mobile interaction. The translation of the task model and ETS’s to abstract interactor objects is left unchanged.

5.3.4 From Abstract Presentation Model To Final UI

In TERESA, the mapping from abstract to concrete presentation is subject to specifying the platform type (i.e. desktop, mobile, etc). Since we are targeting the mobile platform, we have opted for the mobile concrete model provided by TERESA, which assumes a mobile device with large colour screen display with a minimum of 18 characters width and 8 visible rows with graphic support. Each of AIO is mapped to its corresponding CIO according the default mapping, as shown in Table 5.4.

<table>
<thead>
<tr>
<th>AUI</th>
<th>CUI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operators</strong></td>
<td></td>
</tr>
<tr>
<td>Grouping</td>
<td>Unordered List on Column, Fieldset</td>
</tr>
<tr>
<td>Ordering</td>
<td>Ordered List on Column</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Change font size</td>
</tr>
<tr>
<td>Relation</td>
<td>Form</td>
</tr>
<tr>
<td><strong>Interactors</strong></td>
<td></td>
</tr>
<tr>
<td>Navigator</td>
<td>Link, Button</td>
</tr>
<tr>
<td>Description</td>
<td>Text, Image, Text with Image</td>
</tr>
<tr>
<td>Text Edit</td>
<td>Visible Text, Hidden Password</td>
</tr>
<tr>
<td>Numerical Edit</td>
<td>Visible Text, Hidden Password</td>
</tr>
</tbody>
</table>

Table 5.4: TERESA AUI to CUI mapping defaults

This mapping is hard coded into TERESA and cannot be changed, so the only possibility to change this mapping (and the subsequent one to final UI) is to alter the conversion process from concrete presentation model to final UI. To add the support of Thinlet and Kuix in TERESA, we created a mapping between the concrete presentation interactors and the final UI widgets, as shown in Table 5.5.

Some of the GUI widgets which cannot be generated from the existing concrete interactor objects include: progress bar, menus, lists, tree (in Thinlet), and gauge, menu, list, popupbox, dndcontainer
5.3. Changes to TERESA Processes

<table>
<thead>
<tr>
<th>Abstract GUI widgets</th>
<th>TERESA Abstract: Concrete UI</th>
<th>Thinlet</th>
<th>Kuix</th>
</tr>
</thead>
<tbody>
<tr>
<td>(root) container</td>
<td>concrete mobile ui</td>
<td>Desktop</td>
<td>Desktop</td>
</tr>
<tr>
<td>(top) container</td>
<td>presentation</td>
<td>Panel, Dialog</td>
<td>Screen</td>
</tr>
<tr>
<td>container</td>
<td>presentation</td>
<td>Panel, TabbedPane, SplitPane</td>
<td>tabfolder, scrollpane, scrollcontainer</td>
</tr>
<tr>
<td>text output</td>
<td>input_text, text_file</td>
<td>Label, Textarea</td>
<td>Text, Textarea</td>
</tr>
<tr>
<td>image</td>
<td>object: image</td>
<td>Label</td>
<td>Picture</td>
</tr>
<tr>
<td>output</td>
<td>description: text, image</td>
<td>Textarea</td>
<td>Text, Textarea</td>
</tr>
<tr>
<td>selection</td>
<td>single_selection: radio_button</td>
<td>CheckBox with group option to turn it to radio button, ToggleButton</td>
<td>choice, radiogroup, radiobutton</td>
</tr>
<tr>
<td>selection</td>
<td>single_selection: drop_down_list</td>
<td>ComboBox</td>
<td>N/A</td>
</tr>
<tr>
<td>selection</td>
<td>multiple_selection: checkbox</td>
<td>CheckBox</td>
<td>checkbox</td>
</tr>
<tr>
<td>text input</td>
<td>text_edit, numerical_edit: textfield</td>
<td>Textfield, Passwordfield, SpinBox (num), slider (num)</td>
<td>textfield, gauge (num)</td>
</tr>
<tr>
<td>navigator</td>
<td>navigator: text_link</td>
<td>Button with type=&quot;link&quot;</td>
<td>hyperlink</td>
</tr>
<tr>
<td>navigator</td>
<td>navigator: button</td>
<td>Button</td>
<td>button</td>
</tr>
<tr>
<td>navigator</td>
<td>navigator: image_link</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>activator</td>
<td>activator: reset_button</td>
<td>Button</td>
<td>Button</td>
</tr>
<tr>
<td>activator</td>
<td>activator: button_and_script</td>
<td>Button</td>
<td>Button</td>
</tr>
</tbody>
</table>

Table 5.5: Mapping between Abstract/Concrete UI, Thinlet and Kuix widget tags

and scrollbar (from Kuix). Conversely, some CIO's do not have corresponding GUI widgets in Thinlet or Kuix, like table and interactive_description. In addition, the differences in widget attributes and styling support between the Thinlet and Kuix means that a separate concrete-to-final transformation module need to be implemented for each GUI rendering engine.

Considering that TERESA targets principally form-based UI's on mobile phones, where the user is expected to input and view data, the key area for improvement would be to support more layout and styling options for input and output widgets. To this end, we have introduced two main enhancements in the specification of TeresaXML CUI model to improve the user experience, as outlined below (this alterations are also reflected in the mobile concrete XML schema presented in appendix A.3):

- **Support for two types of concrete textual outputs**: It is very common in form-based data presentation to have a textual label preceding the data field and separated with a colon (e.g. Address: 8 St. James Street). While the label and data parts can be represented using TERESA’s default only_output interactors linked together with the grouping operator, then the label and associated data widgets are placed on two different rows on the final UI, breaking the (visual) connection between the label and data parts. For this reason, we have added a new optional parameter to only_output interactors to indicate whether it is label or data. If this parameter is specified, then the two interactors get rendered as two output widgets which are horizontally aligned. That includes: image, input_text and text_file.
5.4. Amazon Book Browser Demonstrator

- **Event handling and client-side processing:** TERESA is strongly influenced by the interaction mechanisms used in the web, since it uses HTML to render the UI. As a result, when activator interactors, such as buttons are used, they are converted into buttons wrapped around by a `<form>` tags to indicate that the form data will be posted to the server when the button is clicked. However, the current version of concrete presentation model does not support client-side processing for events triggered by the widgets or the users. However, as we have seen with Thinlet and Kuix, GUI renderers can capture a wider set of events. To make TERESA more aligned with these new features, we have added a parameter `action` to the following concrete UI's: `radio_button`, `drop_down_list`, `textfield`, `text_link`, `image_link`, `button`, `reset_button`, `button_and_script`. The `action` parameter specifies the method to call once that particular interactor is used. This parameter is mapped to the `action` parameter in the case of Thinlet, and `onAction` in the case of Kuix. While we have only shown how to add the `action` event handling since it is the most commonly used, it is possible to enable the capture of other events triggered by the various GUI widgets in a similar way.

Since any change to the concrete presentation model needs to be reflected onto the final UI, CIO's and their properties need to be adequately mapped to the final GUI widgets and their properties. And since the concrete model is automatically generated within TERESA, it is not possible to modify its internal processes to introduce those changes. In this situation, the way to overcome this limitation is to use the adaptation rules to introduce those changes at the concrete presentation level, effectively post-processing the model.

5.4 Amazon Book Browser Demonstrator

In this section, we present the first sample application illustrating how the proposed approach detailed in section 3.5.2 can be used to formulate user requirements and follow this up with design and implementation phases to generate context-aware UI's. The first example is related to a mobile application that connects to Amazon online store (http://www.amazon.co.uk) via wireless network and provides the ability to browse through the books, and view their details [98]. The rationale behind this choice is that it illustrates an application that uses web services to access remote services while the user interface on mobile device needs to exhibit a high level of richness, responsiveness and adaptation. The demonstrated application provides search functionality on books but can easily be adapted to support other types of items available in the online store.

5.4.1 Scenario and User Requirements

A list of functions that need to be exhibited by the UI (and the underlying application) is outlined below:

1. To provide one integrated application and UI that allows the search and display of book details including reviews.
2. To adapt the UI to the QoS of the network.
3. To provide minimal service during peak time to reduce cost linked to wireless access.
4. To update the user interface at run-time without interrupting the usage of the application.

5. To be a very portable application that can run as many devices as possible with different capabilities, from the a simple device with no graphics support to the mid-range mobile phone with graphical capabilities and larger display size.

6. To specify the characteristics of the user interface and how it should be adapted independently from the back end logic.

An accompanying scenario can also be given to help generate the use cases and the task model. Note that to facilitate the extraction of the task model, we indicate the keywords for tasks with bold text, indicators of temporal relations with italic text, and hints to (high level) task grouping and presentation composition with underlined text.

"The user starts the application, he\textsuperscript{12} is then presented with a search form where the user enters a keyword then activates the search based on the keyword. The application requests to connect wirelessly, and if authorised by the user, it proceeds to connect to the web server then retrieves book details. The different views are available to the user at the same time and the user can switch between them. Three views are available: book info, book details, and book reviews. In the book info view, the following details are displayed concurrently: book title, authors, release date, publisher name, average rating, offer price and listed price. In the book details view, the following details are displayed concurrently: book cover, book description, price as new and price as used. In the book reviews view, the following details are displayed concurrently: average rating, number of reviews available, title of the review, current rating and the text of the review as well as controls for review navigation. When the user activates the controls, he can move back and forth between the individual reviews, and the review details change accordingly."

There is also the need to add a scenario which explains how the UI should react to changes in the context-of-use. We use bold text to indicate the conditions, italic text to indicate their consequences, and underline text to indicate the presentation elements that are concerned with the change.

"The information displayed on the mobile device should accommodate the availability of wireless connectivity. When connectivity degrades the amount of book information displayed is reduced. The application should be sensitive to the cost of the connection incurred by user, and reduce the data load. The UI should use graphics when supported, otherwise it switches to minimal controls if graphics are not supported. The style of the display needs also to accommodate user's preferences. The font size and colour of the user interface should match the settings found in the user profiles."

5.4.2 Use Cases

From the scenario description above, it is possible to exact the main use cases, which indicate the main user activities that involve user' interaction with the system. The use cases that have been

\textsuperscript{12}The term 'he' is used in place of the more appropriate he/she throughout this analysis to avoid awkwardness of language.
formulated have been explicitly selected to highlight the possible modifications to the UI at run-time. Besides, the separation of concerns requires that the design and implementation of the UI should be independent from the design and implementation of the application business logic, as long as there is an agreement on the software interfaces that bind the UI widgets to application function.

- **Use Case 1**: User searches for books: Search is carried out by keyword, and all matched book titles are returned.
- **Use Case 2**: User views basic book info: Display title, author(s), release date, publisher name, average rating, offer price and listed price.
- **Use Case 3**: User views more book details: Display cover, book description, price as new and price as used.
- **Use Case 4**: User views current review information: Display average rating, number of reviews, current rating, review title and review text.
- **Use Case 5**: User navigate through the reviews: Using navigation controls to cycle through the reviews.

Given the number of use cases in this application is relatively small, we have considered all of them in the design and implementation phases. For more complex applications with more use cases and contextual dimensions, a selection process needs to be carried out. The use case evaluation matrix (presented in section 3.5.2.1) is one approach to select the use cases and context dimensions which have the highest value to the user and are most frequently used. This can be achieved by carrying out a survey with potential users who are asked about how often they will be using the functions offered by the application, and how useful they perceive them, and their responses could be on a scale between 0 and 10. Questions can be formulated as follows: "On a scale from 0 to 10, how important is it to have functionality / in a situation s", where / represents the function or mobility dimension under consideration, and s is the use case. Possible functions/mobility dimensions include "wireless access", "degraded access", "tariff", "device type" and "user profile". Then the responses are averaged out (total/number of participants) and the use cases with the highest rating would then be the most likely to be considered for development.

For illustration, it is possible to exploit Use Case diagram’s extension mechanism to show the adaptation use cases that can extend the “static” use case when certain context conditions are present (Figure 5.6).

### 5.4.3 Generation of the Task Model

Based on the use cases, and the keywords highlighted in the scenario description, we can generate a corresponding task model. Each sentence is translated into a task name associated with a task type and temporal relation. Table 5.6 shows the results of this translation. Note that abstract tasks have been used whenever there is indication that an activity is composed of other sub-tasks, such as searching for the book, and accessing review information. Also, the subscript attached to each task indicates the level of depth and order with respect to the parent node. For instance, $t_{abc}$ is the $c'$th task attached to the $b'$th node one level-up, which is itself attached to the $a'$th top node. Note that we have added tasks to maintain consistency of the UI, such as the ability to return to previous view. The decomposition of the task model is better illustrated with the CTT representation shown in Figure 5.7.
### 5.4. Amazon Book Browser Demonstrator

**Table 5.6: Listing of tasks for Amazon book browser application**

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Type</th>
<th>Reference</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>search book</td>
<td>abstract</td>
<td>$t_1$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>enter keyword</td>
<td>interaction</td>
<td>$t_{10}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>activate search</td>
<td>interaction</td>
<td>$t_{11}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>return to main page</td>
<td>interaction</td>
<td>$t_{12}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>connect app</td>
<td>abstract</td>
<td>$t_2$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>request authorisation</td>
<td>application</td>
<td>$t_{20}$</td>
<td>$</td>
</tr>
<tr>
<td>authorise connection</td>
<td>interaction</td>
<td>$t_{21}$</td>
<td>$</td>
</tr>
<tr>
<td>connect to WS</td>
<td>application</td>
<td>$t_{22}$</td>
<td>$</td>
</tr>
<tr>
<td>retrieve list of books</td>
<td>application</td>
<td>$t_{23}$</td>
<td>$</td>
</tr>
<tr>
<td>show list of books</td>
<td>application</td>
<td>$t_{24}$</td>
<td></td>
</tr>
<tr>
<td>access book</td>
<td>abstract</td>
<td>$t_3$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>select book</td>
<td>interaction</td>
<td>$t_{30}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>request book details</td>
<td>interaction</td>
<td>$t_{31}$</td>
<td>$</td>
</tr>
<tr>
<td>view book</td>
<td>abstract</td>
<td>$t_{32}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>return to search page</td>
<td>interaction</td>
<td>$t_{33}$</td>
<td></td>
</tr>
<tr>
<td>retrieve book info</td>
<td>application</td>
<td>$t_{320}$</td>
<td>$</td>
</tr>
<tr>
<td>view book info</td>
<td>abstract</td>
<td>$t_{321}$</td>
<td></td>
</tr>
<tr>
<td>show title</td>
<td>application</td>
<td>$t_{3210}$</td>
<td>$</td>
</tr>
<tr>
<td>show authors</td>
<td>application</td>
<td>$t_{3211}$</td>
<td>$</td>
</tr>
<tr>
<td>show release date</td>
<td>application</td>
<td>$t_{3212}$</td>
<td>$</td>
</tr>
<tr>
<td>show publisher</td>
<td>application</td>
<td>$t_{3213}$</td>
<td>$</td>
</tr>
<tr>
<td>show average rating</td>
<td>application</td>
<td>$t_{3214}$</td>
<td>$</td>
</tr>
<tr>
<td>show listed price</td>
<td>application</td>
<td>$t_{3216}$</td>
<td>$</td>
</tr>
<tr>
<td>show offer price</td>
<td>application</td>
<td>$t_{3216}$</td>
<td>$</td>
</tr>
<tr>
<td>view book details</td>
<td>abstract</td>
<td>$t_{322}$</td>
<td></td>
</tr>
<tr>
<td>show cover</td>
<td>application</td>
<td>$t_{3220}$</td>
<td>$</td>
</tr>
<tr>
<td>show description</td>
<td>application</td>
<td>$t_{3221}$</td>
<td>$</td>
</tr>
<tr>
<td>show new price</td>
<td>application</td>
<td>$t_{3222}$</td>
<td>$</td>
</tr>
<tr>
<td>show used price</td>
<td>application</td>
<td>$t_{3223}$</td>
<td>$</td>
</tr>
<tr>
<td>view book reviews</td>
<td>abstract</td>
<td>$t_{333}$</td>
<td></td>
</tr>
<tr>
<td>show average rating</td>
<td>application</td>
<td>$t_{3330}$</td>
<td>$</td>
</tr>
<tr>
<td>show review index</td>
<td>application</td>
<td>$t_{3331}$</td>
<td>$</td>
</tr>
<tr>
<td>show review title</td>
<td>application</td>
<td>$t_{3332}$</td>
<td>$</td>
</tr>
<tr>
<td>show current rating</td>
<td>application</td>
<td>$t_{3333}$</td>
<td>$</td>
</tr>
<tr>
<td>show review text</td>
<td>application</td>
<td>$t_{3334}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>go to previous review</td>
<td>interaction</td>
<td>$t_{3335}$</td>
<td>$</td>
</tr>
<tr>
<td>go to next review</td>
<td>interaction</td>
<td>$t_{3336}$</td>
<td>$</td>
</tr>
</tbody>
</table>
5.4. Amazon Book Browser Demonstrator

5.4.4 From Task Model to Dialog Model

The dialog model is derived from the task model by grouping the tasks that occur at the same time and are temporally linked (creating Enabled Task Sets (ETS)). The resulting model shows the flow of information between the different ETS's. The CTT model with TERESA's internal task-to-dialog transformer, resulting in the following decomposition\(^\text{13}\):

- ETS1: \( t_1 + t_{10} + t_{11} + t_{12} \)
- ETS2: \( t_2 + t_{21} + t_{24} \)
- ETS3: \( t_3 + t_{30} + t_{31} + t_{32} + t_{33} \)
- ETS4: \( t_{321} + t_{3210} + t_{3211} + t_{3212} + t_{3213} + t_{3214} + t_{3215} + t_{3216} \)
- ETS5: \( t_{322} + t_{3220} + t_{3221} + t_{3222} + t_{3223} \)
- ETS6: \( t_{323} + t_{3230} + t_{3231} + t_{3232} + t_{3233} + t_{3234} + t_{3235} + t_{3236} \)

The decomposition is illustrated visually in Figure 5.8.

5.4.5 From ETS to Abstract Presentation Model

After the grouping of the tasks has been completed, each of the ETS is converted into the structure of the abstract presentation model, whereas its dynamic behaviour is extracted from the task model. For each ETS, there is a corresponding numbered presentation. Table 5.7 shows the distribution of AUI interactors (\textit{onlyOutput} and \textit{interaction}) and connections among the presentation units. The connections indicate the transitions between presentations when interactors of type \textit{activator} and \textit{navigators} are used.

A more granular and detailed decomposition is provided in Table 5.8, where we provide the ID and datatype of interactors as well as the target presentation units. Note that the presentation unit

\(^\text{13}\)This decomposition only illustrates the low-level ETS. In fact, ETS4, ETS5 and ETS6 are also part of a super set since they occur at the same time.
5.4. Amazon Book Browser Demonstrator
Table 5.7: Abstract presentation structure for the Amazon book browser application

with ID = 0 represents the welcome screen of the application, and has not been modelled with CTT. Figure 5.9 illustrates the compositions of the different abstract presentation units and the flow of control is indicated with arrows.

5.4.6 From Abstract to Concrete Presentation Model

The automatic mapping to the concrete presentation model takes into consideration the capabilities of the mobile device in terms of screen size and support for graphics. At this stage, it is possible to
5.4. Amazon Book Browser Demonstrator

<table>
<thead>
<tr>
<th>Connections</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Conn. ID</td>
<td>Interactor</td>
</tr>
<tr>
<td>1.1</td>
<td>activate_search</td>
</tr>
<tr>
<td>1.2</td>
<td>return_main_page</td>
</tr>
<tr>
<td>3.1</td>
<td>request_book_details</td>
</tr>
<tr>
<td>3.2</td>
<td>return_search_page</td>
</tr>
<tr>
<td>4.7</td>
<td>view_book_info</td>
</tr>
<tr>
<td>5.4</td>
<td>view_book_details</td>
</tr>
<tr>
<td>6.5</td>
<td>go_previous_review</td>
</tr>
<tr>
<td>6.6</td>
<td>go_next_review</td>
</tr>
<tr>
<td>6.7</td>
<td>view_book_reviews</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interactors: only_output</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter. ID</td>
<td>Name</td>
</tr>
<tr>
<td>4.0</td>
<td>title</td>
</tr>
<tr>
<td>4.1</td>
<td>authors</td>
</tr>
<tr>
<td>4.2</td>
<td>date</td>
</tr>
<tr>
<td>4.3</td>
<td>publisher</td>
</tr>
<tr>
<td>4.4</td>
<td>avg_rating</td>
</tr>
<tr>
<td>4.5</td>
<td>listedprice</td>
</tr>
<tr>
<td>4.6</td>
<td>offerprice</td>
</tr>
<tr>
<td>5.0</td>
<td>cover</td>
</tr>
<tr>
<td>5.1</td>
<td>description</td>
</tr>
<tr>
<td>5.2</td>
<td>newprice</td>
</tr>
<tr>
<td>5.3</td>
<td>usedprice</td>
</tr>
<tr>
<td>6.0</td>
<td>avg_rating</td>
</tr>
<tr>
<td>6.1</td>
<td>review_idx</td>
</tr>
<tr>
<td>6.2</td>
<td>review_rating</td>
</tr>
<tr>
<td>6.3</td>
<td>review_title</td>
</tr>
<tr>
<td>6.4</td>
<td>review_text</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interactors: others</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter. ID</td>
<td>Name</td>
</tr>
<tr>
<td>1.0</td>
<td>search_keyword</td>
</tr>
<tr>
<td>1.1</td>
<td>activate_search</td>
</tr>
<tr>
<td>1.2</td>
<td>return_main_page</td>
</tr>
<tr>
<td>2.0</td>
<td>auth_conn</td>
</tr>
<tr>
<td>2.1</td>
<td>list_of_books</td>
</tr>
<tr>
<td>3.0</td>
<td>select_book</td>
</tr>
<tr>
<td>3.1</td>
<td>request_book_details</td>
</tr>
<tr>
<td>3.2</td>
<td>return_search_page</td>
</tr>
<tr>
<td>4.7</td>
<td>view_book_info</td>
</tr>
<tr>
<td>5.4</td>
<td>view_book_details</td>
</tr>
<tr>
<td>6.5</td>
<td>go_previous_review</td>
</tr>
<tr>
<td>6.6</td>
<td>go_next_review</td>
</tr>
<tr>
<td>6.7</td>
<td>view_book_reviews</td>
</tr>
</tbody>
</table>

Table 5.8: Detailed decomposition of the abstract presentation model for the first demonstrator application
specify the one-to-one mappings between abstract interactor object and their concrete counterparts, although it is not possible to manipulate the parameters of the concrete CIO’s. This is however achievable through the use of adaptation rules, as explained above, which are applied onto the resulting concrete UI model in the subsequent phase of transformation. The following mappings are used between the abstract operators and abstract interactors, and their concrete equivalents.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Concrete</th>
<th>Interactor</th>
<th>Concrete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grouping</td>
<td>Uncored List</td>
<td>Navigator</td>
<td>Button</td>
</tr>
<tr>
<td>Ordering</td>
<td>Ordered List</td>
<td>Interactive Description</td>
<td>Text with Image</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Bigger size</td>
<td>Text Edit</td>
<td>Visible Text</td>
</tr>
<tr>
<td>Relation</td>
<td>Form</td>
<td>Numerical Edit</td>
<td>Visible Text</td>
</tr>
</tbody>
</table>

Table 5.9: Mapping between abstract operators and interactors, and their concrete equivalents

5.4.7 Generation of the Adaptation Model

From the scenario description and from the use cases, we can establish the context factors that influence the functionality and layout of the UI. The four mobility dimensions to which the UI needs to adapt can be extracted from the scenario; these are: network QoS, connection tariff, device’s graphics support and user preferences. In the following paragraphs, we will derive the context parameters that help us characterise these dimensions and associate quantifiable values to them.

In the case of network QoS, as far as the user perception is concerned, the two most important factors that determine the quality of service of a network connection are bandwidth and delay. Assuming that the device is connected to a UMTS network, our previous research on web access over mobile network [95] has shown that setting the bandwidth above 128 Kbps and 200 ms for the delay will ensure a perceptually good quality of service. This implies that below those thresholds the quality of service may affect the perceived quality of service.

Regarding the effect of connection tariff, the cost per unit (e.g. per Mbits downloaded) of a mobile connection generally depends on the day of the week and time of the day where the connection takes place. Consequently, we can use this information to determine when tariff cost changes and hence when the UI needs to adapt. For the Amazon book browser, we will assume that a cheap tariff is applied between 6pm and 6am during week days and all day during weekends.

As per the application requirements, both network QoS and connection tariff are responsible for the same transformation of the UI. When the connection degrades or when the tariff charge is high, some information can be omitted, hence reducing the data load. To achieve this, we disable the display of the detailed information about the book and reduce the level of details of the reviews. In both cases there is an alteration of the structure and behaviour of the UI, which consists in removing GUI widgets that are specifically derived from the CIO’s contained in the ‘book details’ presentation unit, and changing the attributes of the review presentation unit, by retrieving review summary rather than the full description. Looking at the task tree hierarchy, we can easily identify the corresponding task nodes that match both presentation units i.e. ‘view book details’ and ‘view book reviews’
respective. The transformations to be applied onto the task model consist in removing the 'view book details' and changing the function call associated with 'show review text' task. The derived adaptation rules are shown in the table below.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Condition</th>
<th>Change Type</th>
<th>Level</th>
<th>Operation</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network QoS</td>
<td>bw&lt;128kbs OR delay&gt;200ms</td>
<td>Structure</td>
<td>Task</td>
<td>Remove</td>
<td>Remove subtree f322</td>
</tr>
<tr>
<td>Tariff</td>
<td>day is weekend OR 6am&lt;time&lt;6pm</td>
<td>Attribute</td>
<td>Task</td>
<td>Change</td>
<td>Assign f3224 to procreviewdisplay()</td>
</tr>
</tbody>
</table>

Given that devices with different capabilities should be able to display the user interface of the application, it is imperative to provide different levels of graphical support depending on their capabilities. Accordingly, when a device does not support graphics, the cover of book is not displayed, as well as all widgets that would use graphics (for instance, book rating and book review navigation controls). This transformation involves a change to the task model as well as the concrete presentation model. For the task model, it consists in removing the "show cover" task, and for the presentation model, the targeted CIO's should substitute graphical icons with textural representations. The derived adaptation rules as shown in the table below. Note that the parameters of the second rule are the tasks node that correspond to $CIO_4.4$, $CIO_6.0$, and $CIO_6.2$.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Condition</th>
<th>Change Type</th>
<th>Level</th>
<th>Operation</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>No graphics support</td>
<td>Structure</td>
<td>Task</td>
<td>Remove</td>
<td>f3220</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Attribute</td>
<td>Concrete presentation</td>
<td>Change CUI attributes</td>
<td>Icon graphics to text</td>
</tr>
</tbody>
</table>

The last dimension to which the application needs to adapt is personalisation of the theme of the UI, and this is particularly important in graphical UI's. Given that different users are expected to use the application, the UI needs to accommodate the user preferences with regard to the visual style. As we have pointed out before, one of the advantages of using an XML-based high-level UI description language is the ability to separate the content from the styling of the UI. In the case of the Amazon book browser application, the styling can be supported either from within the UI description language or using an external Cascading Stylesheet (CSS) file. In either case, it is required to apply changes at the concrete presentation level to incorporate those visual styling specifications. Assuming that there are two users, A and B: For user A, use a standard font size with blue background to the UI; for user B, use default font size with white foreground colour and black for background to create higher contrast. Therefore, these changes affect the background colour of the main application 'views' and all presentation interactors that output textural information. To point to those interactors, it is sufficient to list the task nodes that correspond to those CIO's (i.e. title, authors, publisher, avg_rating, listed_price, view_book_info, etc.), as per the syntax of presentation-level attribute operations (section 4.3.7.5). These transformations are shown in the table below.
5.4. Amazon Book Browser Demonstrator

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Condition</th>
<th>Change Type</th>
<th>Level</th>
<th>Operation</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>User preferences</td>
<td>User A</td>
<td>Attribute</td>
<td>Presentation</td>
<td>Change CUI attributes</td>
<td>fontsize: 12, bgcolor: blue</td>
</tr>
<tr>
<td></td>
<td>User B</td>
<td>Attribute</td>
<td>Presentation</td>
<td>Change CUI attributes</td>
<td>fontcolor: white, bgcolor: black</td>
</tr>
</tbody>
</table>

In summary, there are the four dimensions of context and descriptive parameters to which the UI needs to be adapted and their derived parameters:

1. Network QoS (represented by bandwidth and delay)
2. Tariff (represented by date and time)
3. Device (graphics support)
4. User preferences (represented by user profile)

The dimensions and entities used in the above tables are then used to create a UML representation of the context and adaptation models. The context-related elements specified here need to be reflected in the context model. The context elements that are considered in the adaptation model represent the minimum set of elements found in the context model to enable any kind of adaptation. Hence, the context adaptation dimensions should be a starting point for the elaboration of the context model. It is in fact possible for the context model to include more information about the user, environment and the platform in anticipation of future adaptation rules that would depend on the availability of this context information. However, it is advisable to keep the amount of context information exchanged between the client, server and any other third-party context provider to a minimum to avoid creating bottlenecks in a distributed architecture like this one.

Figure 5.10 depicts a partial UML diagram of the context model and the adaptation model. They are both constructed using the stereotypes and attributes previously defined in the UML profile (sections 3.4.4 and 3.4.5). The information contained in the UML model can also be serialised into an XML document to be used for context management and adaptation according to the schemas defined in section 2.7.8.5 and 4.3.7. For illustration, an excerpt from the corresponding XML file that contains the adaptation rules is shown in Listing 5.1.

Listing 5.1: Partial XML file for the adaptation rules of the Amazon book browser

```xml
<root>
  <rule priority="1">
    <condition type="or">
      <condition type="gt">
        <context type="user" name="tariff">
          <ctxParameter name="time" value="18"></ctxParameter>
        </context>
      </condition>
    </condition>
    <condition type="lt">
      <context type="user" name="tariff">
        <ctxParameter name="time" value="18"></ctxParameter>
      </context>
    </condition>
  </rule>
</root>
```
5.4. Amazon Book Browser Demonstrator

(a) Partial context model

(b) Partial adaptation model

Figure 5.10: UML diagrams of a partial context model and adaptation model of the Amazon book browser
5.4. Amazon Book Browser Demonstrator

5.4.8 From Concrete Presentation Model to Final GUI

Up to this point, the specification of structural and behavioural aspects of the UI as well as context adaptation rules have been provided without making this process dependent on the final UI rendering technology. This is in fact the final phase of the process which consists in mapping the concrete model, which is modality specific, to the final UI. This mapping takes into consideration the characteristics of the underlying UI rendering engine, user interface design heuristics/guidelines and UI styling preferences.

At a high level, the mobile application is made up of four main “views”: the first view represents the welcome screen. The second view displays the search page where the user can enter the keyword search query. The third view displays the list of results, and the fourth view shows the different details concerning the book. The latter represents concurrent and different aspects of the book’s information, and the user is able to switch between them instantaneously. This navigation flow of the user interface can be represented with UML activity diagram as show in Figure 5.11.

5.4.8.1 Accommodating the rendering engines

The Amazon book browser application has been rendered using Thinlet and Kuix. As depicted in 5.2, Thinlet uses a number of XML files to describe the different UI views, in addition to graphics and styling documents. These are called by the main Java class which initialises the UI frames, place the
5.4. Amazon Book Browser Demonstrator

Figure 5.11: UI navigation flow of the Amazon book browser application

widget on them, renders the UI and takes care of event management. The following table shows the mapping between the top-level concrete presentation units, described in Figure 5.9, and the XML files that represent them. Each of the lower-level CIO's (like interactors, navigators, text edit, etc.) are converted into GUI widgets according to the mapping defined (Table 5.5).

<table>
<thead>
<tr>
<th>CIO Presentation Unit</th>
<th>XML/XUL File</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>explorer.xml</td>
</tr>
<tr>
<td>2</td>
<td>connection.xml</td>
</tr>
<tr>
<td>3</td>
<td>result.xml</td>
</tr>
<tr>
<td>4</td>
<td>info.xml</td>
</tr>
<tr>
<td>5</td>
<td>details.xml</td>
</tr>
<tr>
<td>6</td>
<td>reviews.xml</td>
</tr>
</tbody>
</table>

Table 5.10: Mapping between concrete presentation units and XML files for Thinlet

The transformation tool that was implemented to enable this takes in the concrete presentation representation (defined in TeresaXML) and converts it into a set of XML files that contain the different GUI widgets, and resize graphics to fit on a display of a targeted mobile device. It also creates a template Java class (which can be modified or replaced by the programmer) and a launch script\(^\text{14}\) to use for testing and prototyping. This process is illustrated in Figure 5.12.

\(^\text{14}\)The script uses Ant which is a software tool for automating software build processes similar to make on for c/c++.. It is used here to compile, package and run the mobile midlet for testing and evaluation on a desktop machine.
Figure 5.12: Transformation from TeresaXML to Thinlet

Similarly, we have also developed an equivalent tool that converts TeresaXML to a set of XML files, java classes, resized graphics, and launch script, which can be used to render a GUI using Kuix rendering engine. This process is illustrated in Figure 5.13.

Figure 5.13: Transformation from TeresaXML to Kuix

5.4.8.2 UI layout heuristics

Given the limited screen size on mobile devices, it is important to present a maximum of information while at the same time not clutter the screen or force the user to scroll down the display. To this end, we have established a number of heuristics to improve the usability of the interfaces. These heuristics are introduced programmatically to the CUI-to-final UI transformation module or manually to the final UI, and they are specific to the current GUI engines used, but not to the current application. These heuristics were formulated based on testing and evaluations of UI prototypes.

For instance, when converting a presentation unit which contains less than 6 rows of interactor objects, it is converted into a non-scrollable panel in the case of Thinlet and Kuix. And if
there are more than six different rows of CUI objects, then the presentation unit is converted into *scrollable panel* in Thinlet's case and into *scrollpane* in Kuix' case.

Also, to support simultaneous views of output data, we propose to use multiple tabs to display information, instead of having the content of multiple presentation units display onto one frame. The use of multi-tabs, could help in maximising the amount of data presented to end users especially that mobile devices are characterised by a small display size. We present an illustration of the difference between rendering with one frame (Figure 5.14a) and rendering with multiple tabs (Figure 5.14b).

Two conditions are set for this heuristics to be applicable to maintain UI consistency:

1. None of the constituent presentation units should include a *navigator* interactor object (e.g. represented by a link or button in the final UI), ensuring that the user does not navigate away from the current frame.
2. The presentation encompasses a number of concrete compositions linked with the *ordering* operator (implying that they need to be displayed together and at the same time), and each interactor composition itself is made up of a number of CIO's linked with the *grouping* operator (implying that they need to be grouped in an ordered list). The top-most presentation unit is mapped to the tab pane (or container), where each contained interactor composition is mapped to a separate tab panel.

![Figure 5.14: Illustration of the different approaches to display output information](image)

The multi-tab rule is applicable to our demonstrator, since most of the UI objects used are text or image output in nature, and in particular to the book details view. By applying this approach to presentation units with ID 4.0, 5.0 and 6.0, we can display them as separate tab panels on the same top-container. Both Thinlet and Kuix support tabs via the *TabbedPane* (or a *Tabfolder*) widgets. Consequently, the content of the previously mentioned *info.xml, details.xml, reviews.xml* are merged together to form one XML file named *details.xml* which is made up of multiple tabs.

Furthermore, given that one concrete UI object can be mapped to multiple final UI widgets, we have used the following mapping rules as far as the Amazon book browser application is concerned:

- *input_text* or *text_file* CIO's are converted into a *textarea* (in both Thinlet and Kuix) unless the text is less than 25 characters in which case it is converted into a *label*. 


- **numerical_input** CIO's are converted a **slider** in Thinlet (or **gauge** in Kuix) if there is a fixed step increase. If the step size if more granular (i.e. less than one unit) then **spinbox** in Thinlet (or **textfield** in Kuix) is used instead.

### 5.4.8.3 UI styling preferences

Due to the difference in layout managers, which are responsible for placing the widgets on the UI, between Thinlet and Kuix, and the limited information about the layout in the generated CUI model, some styling hints were added to take advantage of the visual capabilities of each rendering engine. In Thinlet, the placement and span of widgets on the screen is defined in terms of columns and rows, hence all cells on a single row should be filled before moving into the next row. On the other hand, there is more flexibility to control the layout and look-and-feel of Kuix-based GUI's because they have a more fluid layout manager and support external CSS files.

In the original TERESA, one of the final UI output formats supported is XHTML and CSS file. In short, CIO's are mapped to common HTML tags\(^1\): For instance, it uses `<p>`, `<input>` to designate input/output widget objects; `<li>`, `<ul>`, `<fieldset>` to group those objects; and align some CIO types in one row, as in the case of a text entry field with its label. However, the tool does not check for the validity of the generated HTML documents, or apply layout rules, resulting in the pages not being displayed properly on some browsers or not matching designers' and users' expectations.

For this reason, manual changes to the styling associated with each of the widgets have been applied to this demonstrator. One way to achieve more automation is to have the adaptation rules insert additional styling parameters to the various CIO's and reflect those changes onto the final GUI when mapping to the final UI. However, this technique is flexible but not permanent in the sense that the same styling adaptation rules need to be used across different applications. A more permanent approach would consist in using an external stylesheet (e.g. CSS) that can apply the styling changes across all the views and depending on the widget name, type and class.

### 5.4.8.4 Application of adaptation rules

In section 5.4.7, we have explained the process by which adaptation rules are formulated. In this part, we discuss how the adaptation rules are applied and show the result of the application of this process on the Amazon book browser. The transformation of the task models, and the applications of UI heuristics and styling hints result in a functional graphical user interface. Figure 5.15a depicts the rendering of the UI using Thinlet in favourable conditions (i.e. no adaptation rules applied). Figure 5.15b depicts the rendering of the same UI using Kuix.

\(^1\)Note that there is not check for the validity of HTML documents, resulting in the pages not being displayed correctly on some browsers.
Depending on the prevailing context-of-use, one or more adaptation rules can be applied. For instance, when the connectivity degrades (or that the user is connected at peak time, hence using a more expensive tariff), while the user has a device that does not support graphics, adaptation rule (1), (2) and (3) are used, and the process of generating a 'lighter' version of the UI is initiated. The UI server updates the UI by disabling the graphical icons, and retrieving shorter descriptive book text. The new GUI is then communicated to the client then loaded onto the screen while the application is still running.

As per the adaptation rules, the degradation of connectivity quality results in the application of structural and attribute changes to the task model and the presentation models. In this case, the “view book details” task (linked to the “details” tab on the GUI) has been removed from the task model, and some presentation elements have been replaced with text-based widgets like labels. Figure 5.16a depicts Thinlet rendering of the GUI after adaptation, and Figure 5.16b depict a similar situation when rendering in Kuix. It shows that the “details” frame has been discarded, all image-based widgets have been replaced with simpler widgets, and the reviewer’s comment has been shortened.
Regarding adaptation to user preferences, to support a potential user A, the resulting UI should display larger fonts with blue background, and to support another user B, the font colour should be set to white and background colour to black. This level of adaptation is not supported in Thinlet since it does not support advanced styling like changing font size and colour. On the other hand, Kuix fully supports cascading style sheet (CSS). This is achievable in two ways: either by inserting styling information along with the XML-based description of the final UI, or use an external style sheet file.

Since the final UI is automatically generated from the concrete presentation model, we have opted to use inline styling e.g. `<text style="color:white;bg-color:black">label</text>`, whereby adaptation rules augment the targeted CIO’s with a `style` parameter, which is then mapped onto the `style` attribute of corresponding GUI widgets. Figure 5.17 shows the results of the application of adaptation rules to match the preferences of user A and user B.

It is undoubtedly difficult for the designers to anticipate the dynamic behaviour of the UI just by looking at the task nodes and the temporal relationships specified in the task model from the initial requirements. Current CTT tools, such as TERESA and CTTE, provide the possibility to simulate “what-if” scenarios and generate static UI’s, and then test them manually. However, those tools cannot be used to test how a UI will look like when introducing context-awareness which adds another level of complexity and unpredictability.

On the other hand, the modular framework we have developed combines existing TERESA processes with new ones, which enables a step-by-step transformation of the models, from the initial task model to final UI with the option to support context-based adaptation. This allows the different modules to be used as standalone tools to simulate how each phase transforms the UI models, and how a particular change in context could affect the UI (using for instance the Ant scripts that are automatically generated as part of the model transformation process).

This approach could be used for testing. For instance, for each identified condition, a corresponding context model is created manually using XML. Then the adaptation rules would be applied onto the initial task model. The designer can then run the full transformation process and test the final UI. Despite its limitations, we believe that this is a practical approach given the current constraints imposed by TERESA.
5.4. Amazon Book Browser Demonstrator

Figure 5.17: Adaptation of the UI of Amazon book browser to different user preferences

5.4.9 Implementation and Deployment

The proposed demonstrator application consists of a Java ME-based application on the client side that communicates with Amazon’s web service to search and retrieve book information. This approach ensures the separation of concerns between the back end processing and the UI. The reason for using web services is that it facilitates the rapid development of distributed application by abstracting the communication aspects. Specifically, Amazon’s platform exposes a number of services which enable the query of its internal database of items for sale. The book catalogue is accessible through a set of WS API’s (known as Amazon Associates Web Service), which is part of the Amazon Web Services (AWS)\(^\text{16}\), and allows the retrieval of product information, reviews, etc. The client communicates with Amazon’s WS platform using XML over HTTP. The database is queried by calling a remote method using formatted URI with a list of parameters as a suffix. An example of the syntax of a query is shown below:

\[
\text{http://webservices.amazon.com/onca/xml?Service=AWSECommerceService}\&\text{AWSAccessKeyId=}\[\text{Access Key ID}\]\&\text{Operation=}\text{ItemSearch}\&\text{Keywords=}\[\text{A Keywords String}\]\&\text{SearchIndex=}\[\text{A Search Index String}\]\&\text{Sort=}\[\text{A Sort String}\]
\]

\(^{16}\text{http://aws.amazon.com/}\)
The returned search results are formatted as an XML stream, and the amount of information contained can be controlled depending how the request URI is formatted. Furthermore, information can be limited to particular aspect of the book e.g. reviews, offers, images, etc. To connect to the web service provider, we have used JME library JSR 172 JAX-RPC. It provides the infrastructure to access Web Services based on the model of synchronous Remote Procedure Calls (RPC) using XML messages. However, this library has some limitations such as no support for asynchronous messaging, no support for SOAP messages with attachments and can only support the literal representation of messages. For these reasons, plain XML messages are exchanged over HTTP with WS service provider.

The user can search the online book store by typing a keyword on the search page. The online store is queried by sending a request to the web service and the results are returned as an XML stream, formatted than displayed on the device. When the user selects a particular book, basic details are shown on the “info” tab, extra information is displayed on the “details” tab such as a preview of the cover and a short description, and detailed customers' reviews are shown in the rightmost tab.

To enable this level of functionality, different software modules have been implemented in our work to demonstrate the feasibility of using a model-based approach to create adaptive UI's independently from the processes and mechanisms used to fetch remote data, handle communication aspects, and collect and distributed context information. A broader discussion on how these modules can be integrated into a complete architecture that can be used to deploy and enable the application over the network is presented in section E.2. This potential deployment strategy is based on prototypes that were partly implemented to explore how our model-based framework could be integrated into a live system.

5.4.10 Comparison with TERESA Default Rendering

The original TERESA transformation tool is configured to produce a set of HTML web pages when requested to generate mobile UI's. For the user interfaces to be rendered and used, a web browser needs to be preloaded on the mobile device and the web engine needs to support the version of HTML produced by TERESA. This imposes a number of limitations on the richness of the UI, its functionality and its portability. To illustrate each of these points, we start by showing what a set of UI's from the Amazon book browser would look like if they were generated solely with TERESA tools. In Figure 5.18, we depict the three main screens of the application as they would appear on a Nokia N70 web browser.

As far as the structure of the UI and its look-and-feel are concerned, when compared with Thinlet and Kuix renderings, the web based option has clearly a number of shortcomings. First, there is no support for tabs, which means that the different views are inter-linked with buttons that help navigating between them. Also, since all interactor objects are output types, they are displayed as a list of textual and graphical items. That made the mobile display unsuitable to display all the data, forcing the user to scroll. Furthermore, the fact that the whole application resides on the server, means that if there is any problem with the network, the service delivery as well as its UI will become unavailable. Finally, the lack of exchange of information on the current context-of-use makes this application non-adaptable to user, device and environmental contexts.

Functionally speaking, when using native UI's, like with Thinlet and Kuix, the events triggered by the use of the application are captured and processed locally. That means that for application's functions
5.4. Amazon Book Browser Demonstrator

(a) Info screen

(b) Details screen

(c) Reviews screen

Figure 5.18: Final UI's generated with standard TERESA for the Amazon book browser
can be called instantly because there is no need to have a round-trip exchange with the server. By enabling native UI’s, it is possible for the programmers to take full advantage of the device hardware and software platform such as the use of sounds, touch-screen, and integrate context information such as location. It has to be noted that TERESA supports the inclusion of JavaScript code inside the web-based UI’s, which allows to carry-out local processing while the page is rendered. However, while these scripts can function on desktop web browsers, JavaScript is not as widely supported on the mobile platform, and is quasi non-existent in more resource-constrained devices.

5.5 Museum Guide Demonstrator

The second example illustrates how our proposed approach can be applied to a different type of applications where the user interface is more complex. The demonstrator is a museum guide application which provides visitor with information about artefacts as well as suggestions for tours to navigate around the museum. In this scenario, the user interface needs to adapt to the location of the user and to the level of richness desired by the end user. Unlike in the case of the Amazon book browser, we will only explain the process of generating the task model and the adaptation rules, and will not provide an implementation of the application. In fact, this demonstrator will serve as basis for comparison with CUP approach which is closely related to our work, and from which this scenario is inspired.

5.5.1 Scenario and User Requirements

The example describes a museum application that can operate in two modes. The first is a guided tour which direct the visitor to follow a certain route around the museum, and the second one is exploratory, and let the visitor wander around the museum while prompting him with information about some artefacts that might be of interest. The UI - and by extension, the underlying application - should be able to offer the following functions:

1. Ability to provide location information about the visitor while in the museum
2. Ability to load this application on the visitor’s own portable device
3. Offer a number of museum tour options (in exploratory mode)
4. Provide information about artefacts in the vicinity
5. Provide suggestions on artefacts to view next (in exploratory mode)
6. Provide customisable level of details depending on the ticket type and capabilities of device
7. Retrieve user related information about preferences and previous itineraries
8. Offer information in textual, graphics and audio formats, depending on the type of artefact, and location

The application scenario can be formulated, with highlighted keywords, as follows:
"The user takes the navigation device provided by the museum. The user is asked to select one of the two modes of operation: **guided tour** or **exploratory modes**. If he chooses **guided tour**, then he is offered a choice of itineraries to follow. If he has used the **guide tour** before, his details will be **pre-loaded**, and he would be offered to **resume** his **previous tour** or initiate one of the remaining tours. In both modes of operation, the user's location is **tracked** around the museum, and this **information** is **displayed** on the guiding device or spoken out. In exploratory mode, he gets **recommendation** on **next artefacts** to view, and the user selects the artefact **he** is interest in to **get directions**. When he gets closer to one of the artefact, **related information** is **displayed** on the guiding device or spoken out. He can also optionally **rate** the artefact, which is recorded in the system."

The adaptation part of scenario is as follows, using the same notation that was used for the Amazon book browsing scenario:

"The level of details and amount of information displayed depend on the ticket type and capabilities of device. **Premium users** get **images**, **extended descriptions**. **Non-premium users** get **summarised textual descriptions**. The current view on the display is **updated** based on the location of the user. The **look-and-feel** of the application **changes** based on the **part of the museum visited**."

The corresponding use cases extracted from the scenario description are listed below. Since the number of use cases is small, we will cover them all in the rest of analysis:

- **Use Case 1**: User selects application mode
- **Use Case 2**: Use selects tour itinerary
- **Use Case 3**: User selects next artefact to view based on system recommendation
- **Use Case 4**: User views artefact details
- **Use Case 5**: User provides rating of the artefact (optional)

### 5.5.2 Generation of the Task Model

From the scenario described above, we can generate the following task model specification as shown in Table 5.11. Note that tasks $t_{21}, t_{24}, t_{34}$ could also be considered abstract tasks since they may encompass other sub-tasks that reflect the different types of information and control on the UI. However, we chose to keep them as application tasks so that it is possible to compare our approach with CUP 2.0. For brevity, we will focus on the tasks that correspond to the guided tour part of the scenario for the rest of the analysis. An illustration in CTT of that part of the task model is provided in Figure 5.19.
5.5. Museum Guide Demonstrator

Table 5.11: Listing of tasks for museum guide scenario

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Type</th>
<th>Reference</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visit museum</td>
<td>abstract</td>
<td>$t_1$</td>
<td></td>
</tr>
<tr>
<td>Enter name</td>
<td>interaction</td>
<td>$t_{11}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>Show tour options</td>
<td>application</td>
<td>$t_{12}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>Choose tour type</td>
<td>interaction</td>
<td>$t_{13}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>Guide to artefacts*</td>
<td>abstract</td>
<td>$t_2$</td>
<td></td>
</tr>
<tr>
<td>Show list of artefacts</td>
<td>application</td>
<td>$t_{21}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>Go to artefact</td>
<td>user</td>
<td>$t_{22}$</td>
<td></td>
</tr>
<tr>
<td>Update position</td>
<td>application</td>
<td>$t_{23}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>Show artefact info</td>
<td>application</td>
<td>$t_{24}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>[Rate artefact]</td>
<td>interaction</td>
<td>$t_{25}$</td>
<td></td>
</tr>
<tr>
<td>Artefact’s alerts*</td>
<td>abstract</td>
<td>$t_3$</td>
<td></td>
</tr>
<tr>
<td>Walk around abstract</td>
<td>abstract</td>
<td>$t_{31}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>Walk in museum</td>
<td>user</td>
<td>$t_{311}$</td>
<td></td>
</tr>
<tr>
<td>Update position</td>
<td>application</td>
<td>$t_{312}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>Artefact alert</td>
<td>application</td>
<td>$t_{32}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>Ask info</td>
<td>interaction</td>
<td>$t_{33}$</td>
<td>$&gt;&gt;$</td>
</tr>
<tr>
<td>Show artefact info</td>
<td>application</td>
<td>$t_{34}$</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.12: List of context-sensitive tasks

<table>
<thead>
<tr>
<th>Context source</th>
<th>$t_{11}$</th>
<th>$t_{24}$</th>
<th>$t_{21}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>User profile</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Platform capabilities</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User location</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.19: Museum guide task model using standard CTT notation

5.5.3 Generation of the Adaptation Model

From the adaptation part of the scenario that relates to the guided tour of the museum, we could extract the tasks affected by the adaptation and relate them to the context elements that are likely to trigger them (Table 5.12). What information about artefact to show to the user depends on user’s location, information about his last visit to the museum, and also whether the device can display it. Whether the user is asked to enter his name depends on whether he is a new user. Finally, the list of artefacts displayed at any one time depends on the user location in the museum.

Table 5.12: List of context-sensitive tasks

In fact, the description of this scenario inspired by the work of Van Den Bergh [80], who used a
similar scenario when presenting the CUP 2.0 approach. This allows us to compare between the two approaches when it comes to how context adaptation is enabled. In CUP, the above-mentioned tasks are designated as context-sensitive, either user, application, interaction or environment, depending on their type, overlaying the letter “C” on the top of the standard icons for user, application and interaction tasks in CTT notation, as shown in Figure 5.20. Using this approach, it is possible to indicate what tasks are context-sensitive, but it does not provide information on how to adapt them to context.

By contrast, in our approach we include additional details on how context information relates to the changes on the task and concrete presentation models. This is shown in Table 5.13, where we provide more details on the contextual conditions and what operations need to be executed onto the task and presentation models, and their parameters, to achieve the desired effect on the UI.

Furthermore, in CUP 2.0 the context model specifies the different situations in which an application can be used. For each situation or context of use, the context model contains a package with the stereotype <<contextOfUse>>. This package can only contain instances of classes of stereotype
Figure 5.21: Example of the context model for the mobile museum guide

<context>. Values within one context of use are combined using a logical and, and when different contexts of use are combined, a logical or is used so that multiple contexts of use can be associated with a task. Ranges of values for parameters can be indicated by specifying a minimum and a maximum, or by listing their possible values. An example of the context model modelled in CUP 2.0 is shown in Figure 5.21. It depicts four contexts-of-use in relation to the location of the visitor in the museum, three of which are more accurate whereas the last location context instance refers to any location within the Byzantine era room.

Our approach brings two enhancements when compared to CUP 2.0’s approach: 1) It describes the context in more details; 2) It explicitly relates context-sensitive tasks to context information and specifies how adaptation is carried out by defining the transformation actions to be applied onto the task and/or presentation models. These two aspects can then be modelled using UML to show their structure and composition, and then be serialised into XML to be used as part of the UI adaptation system. This last point is also a clear differentiator with CUP 2.0, since there is no way to turn the design models into a format that is ready for the implementation and generation of UI’s.

A partial UML representation of the adaptation rule is depicted in Figure 5.22. The information contained in the UML model can also be described using the XML schema specified earlier for the adaptation rules in Listing 5.2.

Listing 5.2: Partial XML file for the adaptation rules document of the museum guide application

```xml
<rule priority="1">
  <condition type="and">
    <condition type="equal">
      <context type="user" name="registration">
        <ctxParameter name="roomvisited">
          <value>no</value>
        </ctxParameter>
      </context>
    </condition>
    <condition type="equal">
      <context type="environment" name="location">
        <ctxParameter name="room">
          <value>RomanEra</value>
        </ctxParameter>
      </context>
    </condition>
  </condition>
  <actions>
    <levelAction level="task">
      <elementList>
        <element value="Show artefact info" />
      </elementList>
    </levelAction>
  </actions>
</rule>
```
Figure 5.22: UML diagrams of a partial context model and adaptation model of the museum guide.
5.6. Discussion

5.6.1 Mobile Usability Evaluation Techniques and MBUID

Evaluating a desktop system is a simpler task since the situation in which the system will be deployed is the same situation in which the usability evaluation will take place. In addition, the context-of-use remains the same after deployment. In a mobile setting, the context is constantly changing which makes it difficult to anticipate and evaluate how the device will be used. The context is affected by how the user interacts with the application, which requires the session to be recorded directly by video or indirectly by a tracking and capture software. There are also issues relating to the ability to simulate usage conditions similar to the real world conditions such as network conditions. Furthermore, the social setting is extremely hard to control and manipulate.

Mobile computing is a relatively new discipline when compared with desktop computing. A consequence of this is that there is no widely agreed method for conducting evaluation studies. This has also been pointed out by Beck et. al [5], who conducted a survey of major mobile HCI publications between 1996 and 2002. They discovered that of 114 papers, only 50 had some evaluation component. Of those 50, most used evaluation techniques developed for desktop systems.

Heuristic Evaluation (HE) is a one such technique which was created by Nielsen [44] as a way of structuring expert evaluation of an interface. The basic idea is that interface designers set up a number of design heuristics based on their experience, and against which the interface is evaluated by a usability expert\textsuperscript{17}. It can be carried out relatively quickly and cost-effectively because it requires a smaller number of people. However, this list of heuristics has been formulated with static UI in mind, and there is, as yet, no accepted list of heuristics for evaluating mobile devices. Developing these heuristics for a domain as new as mobile computing is a large task, and it is likely that it will take some time before it becomes possible to form such heuristics.

Another form of expert evaluation is Cognitive Walkthrough. In this approach, the expert evaluator walks-through a particular task, seeing if the user's goals can be met by the information and functionality provided by the interface. It has been adapted to the mobile context by Po et al. [60],

\textsuperscript{17}This list can be viewed at http://www.useit.com/papers/heuristic/heuristic_list.html
proposed a ‘contextual walkthrough’ whereby the cognitive walkthrough is conducted in the same conditions as those experienced by the end-user. However, based on preliminary results the authors found no significant difference between conducting the cognitive walkthrough in the lab or in the end-user context, and its advantages may not be sufficient to outweigh the costs in terms of time, training and gaining access to appropriate contexts of use.

We believe that cognitive walkthrough is not usable in our case. There is in fact no need to verify whether the functions implemented in the UI contribute to helping the user achieving his goals. This is because the task model has been built in the first place from the scenario that was constructed in collaboration with end users. Furthermore, the formalisation of the context-of-use and adaptation rules ensure that the final UI would behave in the way it has been designed for.

Finally, our contributions are centred around the development of abstract models and techniques to introduce context information to enable adaptation. Since it is modality-independent, standard usability techniques to evaluate the resulting UI are not relevant as they will depend on the nature of the interaction exhibited by the final user interface and the contexts-of-use that surround its usage. For these reasons, the evaluation of UI will be carried out on a more abstract level, i.e. in terms of models and processes.

5.6.2 MBUID Approach Evaluation

In this section, we will provide a comparison of our approach with respect to current approaches found in the literature to highlight the differences and improvements introduced. We review the approaches that are most closely related to our work; they concern approaches which focus on the design and development of context-aware user interfaces on resource-constrained devices. We provide information about what models are used in each method, their use of UML and XML, and their support for adaptation. In this discussion, we have included four approaches, namely, UsiXML [38], CUP 2.0 [84], WISDOM [46] and UML* [20]. There are eight criteria used for evaluation, with each them having a number of options (note that N/A standard for not applicable or not available as appropriate):

1. **Software phase**: Requirements (R), Design (D), Implementation (I)
2. **UI Models covered**: Domain (Do), Context (C), Task (T), Dialog (Di), Abstract Presentation (AUI), Concrete Presentation (CUI), Other (O)
3. **UML support**: All (A), Partial (P)
4. **Structural UML 2.0 diagrams used**: Class (C), Composite Structure (CT), Component (CM), Deployment (D), Object (O), Package (P)
5. **Behavioural UML 2.0 diagrams used**: Sequence (SQ), Use Case (UC), Activity (A), State (ST)
6. **XML support**: All (A), Partial (P)
7. **Context-driven transformation level**: Task (T), Dialog (Di), Abstract UI (AUI), Concrete (CUI)
8. **Adaptation stage**: Design (D), Compile (C), Runtime (R)
As it can be observed from Table 5.14, the closest approaches to ours are UsiXML and CUP 2.0. UsiXML offers a complete XML-based notation for the definition of a number of models including domain, task, abstract user interface, concrete user interface and context models. It is focused on business applications and tries to be as complete as possible in the definition of all the models that are considered to be relevant. Mappings and transformations between the different models are defined in separate models. Context-specific user interfaces models can be derived in a top-to-bottom approach guided by transformation models which create a mapping between the model's atomic elements depending on a given context of use, though the final user interface model is not considered, making it less suitable for runtime UI generation and adaptation, unlike our approach which supports dynamic context adaptation up to the presentation level, after the application has been implemented. Besides, despite the great number of tools available for UsiXML, not all tools support UsiXML completely and most of them are special-purpose tools built especially for UsiXML. However, some tools support the export of UsiXML models into TeresaXML to generate concrete presentation and ultimately the final UI. In doing so, it indirectly supports the mobile platform, albeit it is constrained by the inherent limitations of TERESA in terms of support for mobile final UIs (i.e. limited to HTML and VoiceXML).

CUP 2.0 also provides a fairly complete UML-based notation for the definition of a number of models including application model, system interaction model, abstract user interface model, deployment model and context model. However, the approach lacks support for serialised version of the models that can be used for implementation, and also it does not provide explicit support for context adaptation. On the other hand, our approach combines UML and XML-based model notations. In addition, our approach is the only one that provides automated and granular task and presentation model modification in reaction to change of context through powerful adaptation rules.

WISDOM provides a systematic approach to designing and implementing interactive systems. However, the authors assume a particular user interface technology or style, in addition to the fact that the UI is strongly bound to the logic of the underlying application, which makes it less adaptable and reusable. As for UMLt, it focuses on GUI design and implementation. It used an abstract presentation model which illustrates the composition of the UI and extended activity diagram to show its behaviour. Unlike other MBUID approaches, it sets explicit relationships with the application model, instead of transforming the models into a concrete representation. Hence, it offers a very rigid approach which cannot be used for the generation of adaptive UIs.

Table 5.14: Comparison of our approach with other methods found in the literature

<table>
<thead>
<tr>
<th>Criteria</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>R, D, I</td>
<td>C, T, Di, AUI, CUI, O</td>
<td>A</td>
<td>C</td>
<td>UC</td>
<td>A</td>
<td>T, CUI</td>
<td>D, R</td>
</tr>
<tr>
<td>UsiXML</td>
<td>D, I</td>
<td>Do, C, T, Di, AUI, CUI, O</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>A</td>
<td>N/A</td>
<td>D, C</td>
</tr>
<tr>
<td>CUP 2.0</td>
<td>R, D</td>
<td>C, T, Di, AUI, CUI, O</td>
<td>P</td>
<td>C, D, O, P</td>
<td>A</td>
<td>P</td>
<td>N/A</td>
<td>D, C</td>
</tr>
<tr>
<td>WISDOM</td>
<td>R, D, I</td>
<td>T, CUI, O</td>
<td>A</td>
<td>C</td>
<td>UC, A, ST</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>UMLt</td>
<td>D, I</td>
<td>T, AUI, CUI, O</td>
<td>P</td>
<td>C</td>
<td>UC, A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Adaptation, Deployment, Application, System Interaction, Internal Analysis Model
5.6. Discussion

We have also compared our approach with other proposals on a more general set of criteria, and have given a rating on how much each approach satisfies a given criteria, from 0 to 3, 0 indicating no support and 3 which refers to complete support. There are ten dimensions for comparison, some of which originate from the requirements we set for the context model (section 2.7.8.2):

1. **Simplicity**: To what extent are the notations and processes complex for UI designers?
2. **Standard**: To what extent are the notations and processes based on standards?
3. **Context model support**: To what extent is context information modelled?
4. **Context model integration**: To what extent is a context model integrated in the model-driven approach i.e. is part of the same process flow?
5. **Adaptation model support**: To what extent is an adaptation model supported?
6. **Adaptation model integration**: To what extent is an adaptation model integrated in the model-driven approach to generate adaptive UI's?
7. **Generality**: To what extent this approach can be applied to generate UI's with different modalities and for other platforms beyond mobile?
8. **Final UI support**: To what extent does the approach lead to usable final UI's?
9. **Styling support**: To what extent is visual styling supported?
10. **Tool support**: To what extent software tools are available?

<table>
<thead>
<tr>
<th>Criteria</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our Approach</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>UsiXML</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>CUP 2.0</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>WISDOM</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>UMLi</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.15: Comparison between the MBUID-based approaches

As it can be observed from the table above, our approach supports most of the features set out above. However, it is currently only applicable to the mobile platform and has limited support for UI styling since it is built on the top of a framework (i.e. TERESA) that provides little flexibility in this respect. In addition, it lacks an integrated tool for model creation, editing and transformation, and instead it relies on third party editing tools such as MagicDraw. On the other hand, UsiXML fully supports styling in the design phase and provides a number of tools to manipulate the models and interpret them. However, it does not integrate the adaptation model in the overall model derivation process, and has limited support for context information. CUP 2.0's scores are close to those of UsiXML with the exception of features 5 and 9 which are missing, namely support for an adaptation model and UI styling.

WISDOM introduces a minimal set of extensions to the UML, enabling small teams of developers to produce interactive systems. However, it can only generate final UI's based on AUIML markup language, and it also lacks complete support for context, adaptation and styling. UMLi suffers

19This concerns mainly UML and XML editing tools.
from the same shortcomings, in addition to its relatively complex UML-based notation despite the provision by its authors of a specialised tool for that purpose. In addition, it makes assumptions on the type of modality used in the final UI. In summary, we believe that our approach provides the right balance between the need to be simple and standard-based, while providing a way to integrate context and adaptation models into the model transformation process in a way that has not been achieved before.

5.6.3 Context Model Evaluation

After proposing the use of UML for the definition of context entities and relationships and XML for the serialised version, we present its evaluation by comparing it to the most related approaches which have been presented in Chapter 3. Based on the set of requirements set in section 2.7.8.2, we can compare the properties of each model with respect to the extent to which they offer these features. Table 5.16 provides a classification of the existing context models that use UML (with or without XML). Each approach is rated between 1 and 3, where a mark of 1 denotes minimal support, mark of 2 denotes adequate support and mark of 3 denotes full support for a feature. Note that whenever it was possible, we would directly evaluate each approach based on the published specifications, otherwise we relied on published papers from their respective authors. The features are reproduced here with a short explanation on the extent to which our context model supports them.

1. **Semantics/Comprehensiveness:** Our approach to context modelling is very flexible and comprehensive in the sense that it supports the representation of context information provided it can be decomposed into a set of parameters and associated values.

2. **Generic/Applicability:** As far as the context specification is concerned, the syntax does not make any assumptions on the type of applications it will be used for. In this respect, the context model is to a large extent domain-independent. However, the fact that we include information about how the context elements relate to UI models can significantly narrow its applicability scope.

3. **Extensibility:** To complete the support for new stereotypes and concepts in our approach, it is possible to extend the UML profile to support additional information related to the context elements. For instance if we want to add the support for context source information, then we will simply define another stereotype «ContextSource>> to hold information about the context source, and add an association relationship that links it to the «Context>> stereotype. It is also noted these changes can be introduced into the XML schema that define the context specifications.

4. **Structured/Formality:** Since the model specification is based on UML, it inherits the formality in the grammar and semantics. Furthermore, when serialised to XML, it ensures that the description adheres to a certain structure and syntax.

5. **Conciseness/Lightweight Representation:** This is achieved by serialising the model into an XML profile which contains only the relevant information about the context-of-use. As for the UML representation, only Class diagrams are used.

6. **Design and Implementation:** By enabling the inter-exchange of context model that use UML between CASE tools, and inter-operability of context data represented by XML, it becomes possible for the context model to be used for design and implementation purposes.
5.6. Discussion

7. **Standard-based**: UML is the *de facto* standard for software design whereas XML is a widely used standard for data exchange. By basing our model specifications on these two standards, we make it more intuitive for designers and developers to quickly understand the information contained in the models.

8. **Tool Support**: The context model uses the graphical as well as the serialised version of the model, which are both supported by a number of CASE tools and text editors respectively. We have specifically provided support for the UML profile in MagicDraw.

<table>
<thead>
<tr>
<th>Property</th>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</th>
<th>(6)</th>
<th>(7)</th>
<th>(8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our Approach</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>UsiXML</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>CUP 2.0</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>ContextUML</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Ayed and Berbers'</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.16: Evaluation of context and adaptation models

Regarding ContextUML, we can say that despite its extensibility and its ability to model different situations, it is best adapted to the specification of context-aware web services and mobile distributed systems. Furthermore, this approach does not define a serialisation format and the only option available is to convert UML to XMI which is far from being concise and human-readable. As for applicability of UsiXML's context model, it is based on CC/PP standard and hence it is limited to a predefined set of context elements, whereas our approach is extensible but has specific semantics that make it strongly dependent on the MBUID approach we have proposed, especially when indicating the target models that can be affected by a given set of context elements.

CUP 2.0 offers more ability to define context information which can be applicable to a variety of scenarios since the model is independent from the model-driven approach, although available examples are limited to modelling the context in UML, without providing running example of the use of context information in UI adaptation. In terms of tool support, since all UML-based approaches use the profile method to extend UML, most of their authors have provided support for CASE tools. However, regarding the serialisation of these models, only our approach provides a direct mapping between the UML class diagram and XML, whereas UsiXML is purely XML oriented. Finally, Ayed and Berbers's proposed a UML profile for context modelling and application adaptation that supports three types of adaptation: architectural, structural and behavioural. However, it suffers from a lack of serialisation format that is adequate to manually define the context and adaptation rules and use in a UI management system. Besides, there was no further work carried out to support CASE tools for actual application development and enable automatic model transformations.

5.6.4 UI Notation Evaluation

For qualitative evaluation of the notation, we use the cognitive dimensions framework [26], which provides a vocabulary and a framework for usability evaluation of programming languages, generated models and associated editors. The framework helps in determining whether design tools support human activities or not. To clarify some of the terminology used in this analysis, we have provided a table with definitions of the key dimensions under consideration.
Since our notation is based on UML and XML, which are widely adopted in software development, make it very portable and has a good *role-expressiveness*. The use of Class diagrams and associations indicate implicitly the decomposition of context elements and adaptation rule. Also, XML is well suited to represent hierarchical data models such those used in describing context information, adaptation rules, as well as UI presentation models. Furthermore, the fact that we have used publicly available specifications and tools makes the notation relatively easy to understand and use.

We have kept a separation between the models that specify the structure and behaviour of the UI (i.e. task, dialog and presentation) and those that present context information and adaptation rules. There is also a separation between the notations used for a design purpose (CTT, UML) and those used for implementation (XML and TeresaXML). This means that if the designer needs to change a particular model, there is specific notation to use. That means that our notation has a low *viscosity*.

Although we have used Class diagrams to specify context and adaptation models, the stereotypes used are clearly labelled with names that are distinguishable from the other entities. Also, for the serialised (textual) version of the UI models, we have used separate schemas to distinguish between the different models. This makes finding information in the model relatively easy, and hence the properties of *justaposability* and *visibility* are well exhibited.

Finally, the different models represent different aspects of the UI and can be represented with different set of UML stereotypes and XML schemas, and the information they carry can be understood independently from the other models. This highlights the fact that our notation has a low number of *hidden dependencies*, which potentially can reduce errors and confusion.

### UI Generation Tools Evaluation

As far as the final user interface is concerned, the fact that our approach targets primarily markup-based user interface markup languages. That means that the generated user interface will benefit from a number of advantages such as the separation of the business logic from the interaction aspects, ability to re-use parts of the user interface and ease of code implementation and maintenance. A detailed comparison between Thinlet and Kuix is provided in Table 5.18, where a number of features are compared, including platforms supported, support for style, use of layout managers, ability to create custom widgets, a listing of shared and exclusive widgets, and those widgets that are not supported by our approach since they cannot be generated from TeresaXML.
5.6. Discussion

<table>
<thead>
<tr>
<th>Features</th>
<th>Thinlet</th>
<th>Kuix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform support</td>
<td>J2SE, Java Personal Profile, MIDP, Android</td>
<td>MIDP</td>
</tr>
<tr>
<td>Style</td>
<td>Source code or inline limited styling</td>
<td>Source code, inline styling or external CSS</td>
</tr>
<tr>
<td>Layout manager</td>
<td>Single layout Manager</td>
<td>Support different layout managers</td>
</tr>
<tr>
<td>Custom widgets</td>
<td>No ability to create custom widget without modifying the Thinlet Library</td>
<td>Ability to create custom widgets programmatically</td>
</tr>
<tr>
<td>Shared widgets</td>
<td>Label, Button, CheckBox, ToggleButton, ComboBox, TextField, PasswordField, TextArea, TabbedPane, Panel, Desktop, Dialog, SpinBox, Slider, SplitPane</td>
<td>button, checkbox, choice, container, desktop, hyperlink, picture, radiobutton, radio-group, screen, scrollpane/scrollcontainer, tab-folder, text, textfield</td>
</tr>
<tr>
<td>Exclusive widgets</td>
<td>ToggleButton, ComboBox, Dialog, Table, Tree, Separator</td>
<td>choice, dndcontainer, popup-box, picture, radiobutton, radiogroup, screen</td>
</tr>
<tr>
<td>Not supported by our approach</td>
<td>ProgressBar, ProgressList, MenuBar, Tree, Separator</td>
<td>gauge, list, menu, dndcontainer, popupbox, scrollbar</td>
</tr>
</tbody>
</table>

Table 5.18: Comparison between Thinlet and Kuix

To further evaluate the two markup-based GUI rendering engines used for the first demonstrator, we use a framework that was proposed by Myers [41] for the evaluation of UI tools. Some of the metrics that have been adapted to our work are outlined below:

- **Depth**: How much of the user interface does the language and associated tools cover?
- **Breadth**: How many different user interface styles are supported?
- **Portability**: Will the resulting user interface run on multiple mobile platforms?
- **Ease of use**
- **Efficiency for designers**: How fast can designers create user interfaces with the associated tool?
- **Quality of resulting user interfaces**: Does the tool generate high-quality user interfaces? Does the tool help the designer evaluate and improve the quality?

In Table 5.19, we rate Thinlet and Kuix based on how much they match the above stated criteria, 0 indicates no support and 5 means full support. These evaluations are based on our subjective experience in using the software and prior evaluations found in the literature\(^{20}\). While Thinlet and Kuix offer the same capabilities in terms of support of GUI widgets, Kuix offers support for styling and generate better quality UI's than Thinlet. On the other hand, Thinlet supports the running of UI of desktop as well as various mobile platforms such as PDA's, provides a tool for visual creation and editing of UI's and offers simpler syntax at the cost of reduced functionality, customisation and richness. The fact that both rendering engines were supported with little modifications to the transformation shows the level of flexibility of the model-based UI development approach since these changes are needed only at the last stage of mapping (i.e. concrete presentation to final).

\(^{20}\)For instance, [74] and http://leepoint.net/notes-java/GUI/misc/80gui-generator.html
5.7 Conclusion

One of the objectives of our approach was to enable the generation of UI's at runtime based on an initial task model, and react to a change in the context of use according to the set of pre-defined adaptation rules. In addition, the objective has been to enable the creation of dynamic UI's that can interact with back-end services such as web services and enable advanced functions beyond the stateless mode found in the web model. The user interface has also to support modern GUI rendering engines which offer enhanced visual widgets and enable customisation of the user interface. In this chapter, we have exposed in details how our approach and the developed tools can be used to enable a functional mobile UI that uses native GUI rendering engines, and which are adaptable to a change of contexts-of-use. We have also presented an evaluation of our approach with respect to a number of related approaches on different dimensions including modelling notation, tools, and the level of support for design and implementation.

Table 5.19: Evaluation of Thinlet and Kuix

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Thinlet</th>
<th>Kuix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Breadth</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Portability</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Ease of use</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Efficiency</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Quality of UI</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

5.7 Conclusion

One of the objectives of our approach was to enable the generation of UI's at runtime based on an initial task model, and react to a change in the context of use according to the set of pre-defined adaptation rules. In addition, the objective has been to enable the creation of dynamic UI's that can interact with back-end services such as web services and enable advanced functions beyond the stateless mode found in the web model. The user interface has also to support modern GUI rendering engines which offer enhanced visual widgets and enable customisation of the user interface. In this chapter, we have exposed in details how our approach and the developed tools can be used to enable a functional mobile UI that uses native GUI rendering engines, and which are adaptable to a change of contexts-of-use. We have also presented an evaluation of our approach with respect to a number of related approaches on different dimensions including modelling notation, tools, and the level of support for design and implementation.

Table 5.19: Evaluation of Thinlet and Kuix

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Thinlet</th>
<th>Kuix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Breadth</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Portability</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Ease of use</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Efficiency</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Quality of UI</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Chapter 6

Conclusions And Future Work

6.1 Conclusions

As we witness the shift from desktop computing towards mobile and ultra-portable devices, the design and development of common user interfaces to accommodate the diversity and constraints of portable devices is emerging as a critical challenge. To overcome the limitations of current user interface design and implementation approaches, we proposed a design methodology and a software framework that facilitate the generation of user interfaces that are adaptive to the environment and platform constraints as well as to the user profile. With the approach presented in this report, we believe that we have paved the way towards an enhanced way of building context-aware user interfaces using Model-Based User Interface Development. MBUID shows the potential to radically change the way we develop adaptive UI's, by shifting the focus from code to models.

The user interface represents the medium through which a user can utilise the device and access the services developed for it. As these devices and services become increasingly divergent in their capabilities, the task of developing the UI interface becomes more complex. This trend is set to accelerate as portable devices are no longer limited to calling and messaging. To ease development to some extent, current software development platforms either assume a minimum set of common features between handsets, like Java ME, or make software development only possible on a pre-defined set of tightly controlled devices, like Blackberry. In either case, the interactive part of the application is not as portable and adaptable as the logic part because assumptions about the target device and usage have been hard-coded early on the development lifecycle.

Since our approach relies on abstract description models, it can be considered to be a rather unconventional method to designing and implementing user interfaces, when compared to more prevailing methods which consider the UI as one of the steps in the linear development lifecycle. For this reason, this approach is most suitable for applications that can support a level of separation once the application is deployed. Our approach takes advantage of the availability of high-level mark-up based languages and software frameworks which enable remote method invocation to access resources from multiple locations and dynamically generate the UI.

Our approach is also based on the clear separation between the aspects that relate to the interaction (from the user task execution perspective), and the aspects that relate to the underlying application
6.2. Contributions

(from a business logic perspective). This way, it becomes possible to design both components in parallel, while keeping the implementation of the user interface as late as possible in the process. Our approach is strongly task-oriented, in the sense that we identify the activities involved in the interaction, then determine the software objects they need to manipulate. This is in contrast to purely UML-based approaches which focus on modelling the objects that compose a system, instead of considering specific aspects related to user interaction. For this reason, we recommended to use both techniques, whereby CTT (ConcurTaskTress) notation can be used for the high-level design of user-oriented interactive applications, and UML for the low-level details for the design of the software objects that enable this interaction. As a result, it is necessary for UI designers and developers to familiarise themselves with the UML and XML notations and processes introduced in our work.

Specifically, the scope of the research presented in this report is limited to resource-constrained devices. By their nature, these devices can support a limited number of modalities and have limited capabilities which make it hard to maintain an acceptable level of usability. As a result, they are the most likely devices to need adaptation when contextual conditions change. As technology progresses, what we consider today to be state-of-the-art may become the norm in the future. To keep up with the increasing capabilities of the device's hardware and software, the approach described in this report can be augmented in different ways to accommodate these capabilities and enable a richer interaction. For the visual aspects of the UI, the abstract models can be converted to a source code that can be compiled offline and then be loaded into the device. This way, it becomes possible to create a UI that is more optimised and functional than what can be generated using markup-based languages. As for UI functionality, the support of new devices entails not just making structural and behavioural changes to the UI, but also taking into consideration the functions afforded by the underlying hardware and software platform. This consists in enabling the insertion of UI elements, such as maps, that are linked to a built-in GPS sensor for instance, or having the UI's that react to the tilting and orientation of the handset. These functions can only be enabled by making use of specific libraries, which are accessible at a lower level than high-level UI description languages.

Regarding the limitations of our approach, we acknowledge the fact that there are some issues with efficiency and speed since human intervention is required in most cases. However, our argument in favour of the use of models is that high level description languages make it is faster and easier to generate UI's. In fact, we do not claim that by automating some parts of the process, we will eliminate the need to fine-tune the resulting source code and make manual adjustments to the process. Furthermore, although it was shown that it is possible to specify the models for context-sensitive user interface using UML and a UML profile, it was also clear that the use of a UML profile has some drawbacks because the specifications of UI model yield large graphical diagrams which can be difficult to interpret intuitively. We believe that we have attempted to reduce the complexity of the generated models to some extent by limiting ourselves to one main diagram type throughout our approach. We have also recommended the use of CTT and UML notations and concepts to allow different stakeholders in the development process to take part in the design phase, not just by understanding models but also contributing to them.

6.2 Contributions

The area of user interface development for resource-constrained devices, such as mobile phones, has benefited from numerous research activities in the last decade. Furthermore, the need to accommo-
date the heterogeneity of devices, software platforms, users and usage scenarios calls for new methods to be developed to facilitate the design of interactive systems in a pervasive environment. To this end, model-based UI development techniques have been recently introduced to eliminate the need to rely on the characteristics of the interaction capabilities and mobile usage aspects in the early stages of user interface development. The variety and scope of the approaches in this nascent area provides evidence for the scientific and industrial interest in this non-traditional approach to UI design and generation, and creates an opportunity to further the research.

Specifically, in Chapter 1, the problem statement as well as the goals of the research work performed for this thesis were formulated. The aim was to develop a methodology for UI design and enable implementation based on abstraction and model transformation. The support for context modelling, the use of standard-based notations to specify interaction and adaptation aspects and demonstrate how such approach could enable the generation and adaptation of native mobile UI's, were also considered. Our work has resulted in the following contributions to the area of model-based design of user interface for resource-constrained devices:

In Chapter 2, we have described principles and patterns of software development in the context of mobile user interface design, implementation and deployment. We have explained why the separation of concerns is important and how the adoption of a model-based design approach brings advantages over existing software development patterns. We have also given an overview of an original model-based framework that uses a number of abstract models, namely task, abstract presentation, concrete presentation, context and adaptation models. We believe that this set is sufficient and suitable to create functional mobile UI's.

In that chapter, we have proposed an extensible, standard-based model that can be used to describe the current context-of-use. We considered the context-of-use to be a triplet of information related to the user, the platform and the environment. This definition provided us with the ability to include a wide range of context sources, yet by decomposing each element into its most basic components and grouping them into three sub-profiles (i.e. user, platform and environment), it was possible to structure context information in such a way that only part of the context model could be updated, using differential context profiles. We have also presented an XML-based notation to define the structure of the context model and describe the current context-of-use.

We have also explained how context information could affect the structure, behaviour and visual style of the UI through the change in the structure of the task and concrete presentation models and/or the attributes of their constituents. To enable this, we have initially introduced the concept of choice nodes to augment the original task model notation. That already represented an improvement over existing work found in the literature (e.g. [81] and [15]) because it combined pre-processing of the task model at the node level, and the post processing of the derived concrete presentation model. However, due to the limitations encountered with choice nodes, this approach was refined by separating the adaptation processes from the models' descriptions. The latter enabled the specifications of how a UI should change if the current context-of-use matched a set of pre-defined conditions. The advantage of having external adaptation rules, is that there is no need to change the syntax and semantics of the default models that are affected by adaptation. In addition, it was possible to modify multiple tasks nodes and/or presentation units in response to a change of context-of-use. The other advantage is the ability to make existing UI's context-adaptable provided that they have been developed using the TERESA framework, which is a well-established framework on which our is based.

In Chapter 3, we have proposed an original UML profile that can be used to describe the task,
context and adaptation models in a notation which is familiar to software developers. For modelling we have opted to use a structural UML element namely Class diagrams, which are reusable and top level UML entities. In addition, we have used the Use Case behavioural diagram to represent the main functions of the UI and illustrate the adaptation conditions and their effects on the UI. For completeness, we have also provided a detailed description of an extension to the standard user interface development lifecycle on how to convert the initial application scenario into use cases, then to a set of UI models and adaptation rules, which can be used to implement the user interface and its features.

In Chapter 4, we complement the XML-based notation of the context model, with an additional one for the context adaptation model, while exploiting existing notations provided by TERESA for the task and presentation models. The newly defined serialised versions of the models are direct mappings of their UML equivalents introduced in the precedent chapter. The syntax of adaptation rules allows fine control over the structure of the task model such as the addition and deletion of nodes and subtrees, as well as changing their attributes. As for presentation-level transformations, it is possible to modify the existing parameters of presentation objects and also add new ones to accommodate the capabilities of the UI rendering engine such as styling. The support for arithmetic and logical operations also enabled the creation of relatively complex adaptation rules with nested conditions, which is a unique feature in comparison with current UI design approaches.

In Chapter 5, we presented a reference UI management system that is adequate to supporting model-driven UI generation and adaptation, and provided a number of recommendations for its implementation and deployment. We have also shown how our approach can effectively be used to automatically generate user interfaces rendered with different rendering engines. That was achieved by creating new modules that interfaced with TERESA internal transformation processes, and developing new ones to support concrete-to-final UI mappings. This was accompanied by a details explanation of the steps needed to convert an informal description of a scenario into a set of requirements, then into the different UI models that specify the UI’s structure, behaviour and adaptation. Moving beyond web-based interfaces allowed us to support more advanced types of widgets and to provide more control over their placement on the screen and their visual style. That was achieved by applying a number of heuristics to the concrete-to-final mappings to improve the way information is displayed on mobile devices, and make it ready to support the new widgets enable by modern GUI renderers such as multi-tabs panels. Finally, we have evaluated our approach by comparing it to current UI model-based development frameworks, and discussed UI implementation and evaluation aspects.

6.3 Future Work

6.3.1 UML Modelling

Although it was shown that it is possible to specify the context and adaptation models using a UML profile, the use of UML has undoubtedly some drawbacks. To best convey structural and containment information, these two models need to graphically exhibit a hierarchical structure to indicate composition. However, UML models may look different when using other editing tools. To enforce the hierarchy structuring would require changing the UML meta-model, rather than just using extensions of the meta-model. In our case, we have opted to maintain compatibility with free and
6.3. Future Work

commercially available UML tools and propose a light-weight extension model while using a common meta-model.

Furthermore, our work focused on using UML to efficiently model the context and adaptation model, but work on complementing the set of models defined in our thesis with others, such as task and presentation models, could prove to be useful however it is difficult to keep them simple enough to be used by non-professional users. Just like in our attempt to model the task model, similar approaches by the research community have resulted in cumbersome models, strengthening the case of using CTT instead. At the same time, no known work has been proposed on the modelling of the abstraction and concrete presentation models in TERESA.

One possible venue of research on the modelling of the task model with UML would be the use of unique visual representation for each of the components that make up the model. This should make it easier to comprehend them, while maintaining consistency across different models (for instance, between task model and the action part of the adaptation model). As for abstract and concrete presentation models, there is the issue of duplication of stereotypes and their properties across them. Resolving this may require significant changes to the UML meta-model. For instance, a new meta-model could be defined in such a way that it is close enough to UML in semantics and representation, but allows the specification of UI models with meta-classes shared by the two models and provide alternative representations depending on concrete representation intended.

6.3.2 UI Generation

It is understandable that MBUID can be encountered with skepticism from the UI developers since the framework can never automate the creation of UI without needing to adjust the source code. In our work, we had to make some assumptions on the mapping between each abstract interactor object, its corresponding concrete interactor object and the final UI widget, to enable the semi-automatic transformation of the abstract models to concrete presentation model then to final UI. These mappings were driven by usability considerations and also by the type of rendering engine used for the final UI. First, there is a one-to-many mapping between AUI objects and their corresponding concrete and final UI, and we had to make subjective decisions (in the form of heuristics) on which mapping is best suited for the demonstrator, then this information was hard-coded into the transformation module or inserted into the adaptation rules. Secondly, the specificity of each GUI Tenderer is taken into consideration with regard the parameters and style features supported. This implies that if another designer is to be asked to support a new type of GUI engine, she would have to modify the mapping accordingly. Although the changes introduced are minimal, the need to modify the source code of the transformation modules and XML schemas of UI models may discourage designers who are not comfortable with coding and changing XML schema specifications. It can be argued that this new approach may be initially a little difficult for UI designers to adopt, but by developing visual tools it may give the opportunity for them to experiment more freely with the concepts introduced here. We strongly believe that the advantages that can be reaped from a model-based approach outweigh these obstacles.

With regard to automation, our goal was to automate as much as possible while keeping a level of manual intervention by the designers and programmers. There are two reasons for that: the first being that the current state of MBUID (especially if it relies on the CTT notation) depends greatly on the evolution of the tools developed for it, and at this point, there are still limitations to how
6.3. Future Work

much can be automated. The second reason is that we believe that since the user interface is a very important aspect of service delivery and user experience, it is important to ensure that the UI is usable, visually appealing, and allows customisation. Naturally, these subjective requirements cannot be completely satisfied through automation. The need for manual intervention is particularly visible, as we move away from abstract model towards more concrete ones, ending with the final UI.

However, a side effect of automation, is the inability of the designer to anticipate what the final UI would look like, especially when different devices with different software platforms are used. Unpredictability has been identified for long time for being one of the most common limitation of model-based UI design [42]. One approach to reduce this is to have a visual tool that can support the different UI models, similar to TERESA, and which can highlight the mappings between the components of the different models. The designer would be able to visually edit the UI at any level of representation, and she can also have real-time previews of the final UI as components are modified and mappings actioned. For completeness, the tool could also simulate contextual conditions and enable the designer to test the effect of the adaptation rules on the different models up to the final UI.

6.3.3 MBUID in Pervasive Computing Environments

As we have shown in the deployment proposal for the Amazon book browser demonstrator (appendix E.2), UI designed according to an MBUID methodology is perfectly suitable to be deployed in distributed environments where the context management part is separated from the business logic, which itself is separated from the UI generation processes. The separation of concerns afforded by MBUID and service-oriented architecture, and the use of a common syntax based on XML, enable the distribution of UI models and associated transformation processes among different computing units, and facilitate the inter-operability between the different processes. The generic user interface may be generated by one process, possibly residing on one system, transform that interface with another process, possibly residing on another system on the network, and present the final user interface for user interaction in yet another system. Finally, improvement of processing power available on mobile devices means that more transformation processes involved in the UI generation can be shifted to the end user device. That means that there will be less reliance on the server side, and therefore, the ability to generate and adapt the UI's instantaneously. Part of future work would be to implement a prototype that can demonstrate the feasibility of distributed UI generation and adaptation across different systems.

6.3.4 MBUID and Multimodality

Although multimodality is not the main target of our approach, we anticipate that the rapid growth of device capabilities will allow future low-end handsets to support some level of multimodality. The use of markup-based languages and the separation of interactions aspects from business logic enables the support of other modalities on the user interfaces, beyond the GUI (graphical user interface). Currently processing power represent a barrier for the diffusion of multimodality in the mobile world, but as portable devices become more capable, they will be able to run advanced modality processing applications such as speech recognition.
At the same time, the development of high-level markup-based languages such as W3C's EMMA notation [94] and development frameworks, such as W3C's Multimodal Interaction Framework [88], are likely to accelerate the adoption of these technologies. Besides, the increasing integration of context sensors into mobile handsets will facilitate the acquisition of information which will make multimodal interaction more reliable.

As far as our approach is concerned, there are three levels of support of multimodality that can be introduced: at the presentation level, at the interaction level, and at object level. The first level concerns the presentation of output data. In this case, the support for multimodality is enabled prior to the generation of a concrete presentation model. That means that it is possible to represent the interactor components and associated data using high-level multimodal specification languages like W3C's EMMA [94]. Supporting other implementation languages can be introduced with limited effort, as this would require simply modifying the transformation from the concrete interface description to the target implementation language. This approach has in fact already been demonstrated with UsiXML1 and, to lesser extent, with TERESA, where generators are used to produce XHTML + VoiceXML [90] (X+V) multimodal UI's.

Enabling a true bi-directional interaction between the user and the interface requires changes to the model specification and transformation processes. In addition, specialised software, residing on the device or remotely accessible, is required to process user input and generate interface output, such as gesture, touch-screen or voice-based. Given the complexity of software relationships and data flows between the UI and back-end, it is possible to encapsulate this information into a separate model, which can be described as the interaction model. The latter would include the bindings between, on one side, the user and UI actions, and, on the other side, the software operations responsible for translating these into input from and output to the system. In the case where the UI resides on the same platform as the multimodality processing unit, direct method invocation can be used. And in the case of a remote access, a messaging mechanism can be defined to send operations' parameters and activate the necessary multimodal transformation.

More specifically, to enable gesture controls the concrete presentation language needs to be augmented to support gesture information. As the main use of gesture is to navigate between interface elements, it is important to indicate the current focus, which can be shown by modifying the graphical elements associated with the interactor objects, such as highlighting the text or changing its colour. Customised gestures can also be defined in the concrete presentation model and mapped to the traditional GUI elements (i.e. control, activate, select and navigate). Gesture interaction is considered to be a primarily input modality, hence it is difficult to use gestures to prompt and give feedback types of interaction, which can be supported through the careful combination of other modalities to ensure a functional UI. Graphically, interactors can be implemented using links, buttons and menus. Input interactors for gestures are defined as extensions to existing graphical ones.

To enable the use and manipulation of other UI objects, such as graphics and animations, the approach presented in our work needs to be updated to handle the new interaction technique that go beyond keypad, and also include a mechanism to generate rich objects such as graphics. For this, a new interactor object needs to be defined in the abstract model which supports the different actions that can be applied onto the object. For instance, in the case of vector-based graphics, it is possible to specify rotation and scaling as two transformation actions. As for an animation, potential controls

---

1The UsiXML team has developed TransformiXML tool, which supports the transformation of abstract UI model into different concrete UI Models e.g. graphical, vocal and multimodal UI's.
can include starting, pausing and rewinding. It is also possible to push the level of granularity further by considering a graph or an animation as a composition of more basic components, such as graphical elements (e.g. arcs). These atomic components can be combined through the same operators we used for composing different GUI interactions, such as grouping, hierarchy and ordering. Depending on how the operators are used, they can indicate the sharing of some relationships between different interactors or the composition of the graphical elements that are part of the same structured object. The latter is rendered as independent objects, while the structured object composition provides additional interactive capability in such a way that user actions, e.g. zooming and rotation, are applied to all the grouped elements simultaneously.

6.3.5 User-generated Adaptation Rules

While our work primarily targets practitioners in UI design and development, it is possible to extend its scope to end users, at least on the aspects related to context adaptation. By enabling novice users to formulate their own adaptation rules, it is possible to empower the end users and give them more control over how their applications look and behave. One way to achieve this is to create a wizard-like application on a PC or on the mobile device to guide users through the process of associating conditions, defined in terms of high-level context conditions e.g. location, with their effect on the UI, e.g. activate voice control, using simple metaphors with support for graphical aids. This is more important as more wireless sensors are embedded inside mobile devices and in the environment, making available large amounts of context data that can be exploited by mobile services. As end users get more control over how context information is shared and used to automate some of the tasks, it is expected that this will lead to increased adoption of context-aware services. Furthermore, since simplicity and lightweightness were a priority in our research, we have not considered issues related to privacy and security. Nonetheless, these issues represent crucial areas for future research.

6.3.6 Usability Evaluation

To a large extent, evaluating mobile systems remains an open question. As we have noted in the previous chapter, there is little published work on the evaluation of end-user perspective in ubiquitous computing environments. Hence, this aspect remains one of the major challenges for mobile computing systems. For this reason, we believe that current evaluation techniques used to evaluate user interfaces in static mode, should be expanded and improved to support the process of contextual evaluation. Hence, this is an important area that should be the focal point of researchers in mobile computing in general.
References


References


References


References


Appendix A

TERESA Model Specifications

A.1 Current User Interface Context Schema

Listing A.1: Schema for Current User Interface Context model

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="CurrentUserInterfaceContext" type="uccType"/>
  <xs:complexType name="uccType">
    <xs:sequence>
      <xs:element name="profile" type="contextType" maxOccurs="unbounded" minOccurs="1" />
    </xs:sequence>
    <xs:attribute name="ctxEntityID" type="xs:string" use="required"/>
    <xs:attribute name="timeStamp" type="xs:time" use="required"/>
  </xs:complexType>
  <xs:simpleType name="lvlType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="task"/>
      <xs:enumeration value="concrete"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="natureType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="dynamic"/>
      <xs:enumeration value="static"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="profType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="user"/>
      <xs:enumeration value="environment"/>
      <xs:enumeration value="platform"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="paramType">
    <xs:sequence>
      <xs:element name="value" type="xs:string"/>
      <xs:element name="affectedUILevels" type="levelType" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
A.2 Task Model Schema

Listing A.2: Schema for Task model

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT TaskModel (Task) >
<!ATTLIST TaskModel NameTaskModelID CDATA #REQUIRED >
<!ELEMENT Task (Name, Type, Description, Platform*, Precondition?, TemporalOperator?, TimePerformance, Parent?, SiblingLeft?, SiblingRight?, Object*, SubTask*) >
<!ATTLIST Task Identifier CDATA #REQUIRED >
<!ATTLIST Task Category (abstraction | user | interaction | application) #REQUIRED >
<!ATTLIST Task Iterative (true | false) #REQUIRED >
<!ATTLIST Task Optional (true | false) #REQUIRED >
<!ATTLIST Task PartOfCooperation (true | false) #REQUIRED >
<!ATTLIST Task Frequency CDATA #REQUIRED >
<!ELEMENT Name (#PCDATA) >
<!ELEMENT Type (#PCDATA) >
<!ELEMENT Description (#PCDATA) >
<!ELEMENT Platform (#PCDATA) >
<!ELEMENT Precondition (#PCDATA) >
<!ELEMENT TemporalOperator EMPTY >
<!ATTLIST TemporalOperator name (SequentialEnabling | Disabling | Choice |
A.3 Mobile Concrete Presentation Model Schema

Listing A.3: Schema for Mobile Concrete Presentation model

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % length_value "4 | 5 | 6 | 7 | 8 | 9 | 10 | 12">  
<!ENTITY % font_value "Arial | Courier | Times_New_Roman | Times | Verdana">  
<!ENTITY % font_size "9_pt | 10_pt | 11_pt | 12_pt">  
<!ENTITY % bool_op "and | or">  
<!ENTITY % cardinality_value "low_card | medium_card | high_card">  
<!ENTITY % element_selection_alignment "horizontal | vertical">  
<!ENTITY % option "yes | no">  
<!ENTITY % position "column | row">  
<!ELEMENT concrete_mobile_interface (device_type, default_settings, presentation+)>  
<!ELEMENT device_type (big | medium | small)>  
<!ELEMENT big EMPTY>  
<!ATTLIST big graphic_support (%option;) #REQUIRED>  
<!ELEMENT medium EMPTY>  
<!ATTLIST medium graphic_support (%option;) #REQUIRED>  
<!ELEMENT small EMPTY>  
<!ATTLIST small graphic_support CDATA #FIXED "no">  
<!ELEMENT default_settings (background_color, font_settings, operators_settings, interactors_settings)>  
<!ELEMENT background_color EMPTY>  
<!ATTLIST background_color_value CDATA #REQUIRED>  
<!ELEMENT font_settings (font, color, size)>  
<!ELEMENT font EMPTY>  
<!ATTLIST font font (%font_value;) #REQUIRED>  
<!ELEMENT color EMPTY>  
```

Interleaving | Synchronization | SuspendResume | SequentialEnablingInfo) #IMPLIED

```xml
<ELEMENT TimePerformance (Max, Min, Average)>  
<ELEMENT Max (#PCDATA)>  
<ELEMENT Min (#PCDATA)>  
<ELEMENT Average (#PCDATA)>  
<ELEMENT Parent EMPTY>  
<ATTLIST Parent name CDATA #REQUIRED>  
<ELEMENT SiblingLeft EMPTY>  
<ATTLIST SiblingLeft name CDATA #REQUIRED>  
<ELEMENT SiblingRight EMPTY>  
<ATTLIST SiblingRight name CDATA #REQUIRED>  
<ELEMENT Object (Platform*, InputAction, OutputAction)>  
<ATTLIST Object name CDATA #REQUIRED>  
<ATTLIST Object class (Text | Numerical | Graphic | Image | Position | null) #REQUIRED>  
<ATTLIST Object type (Perceivable | Application | null) #REQUIRED>  
<ATTLIST Object access.mode (Access | Modification | null) #REQUIRED>  
<ATTLIST Object cardinality (Low | Medium | High | null) #REQUIRED>  
<ELEMENT InputAction EMPTY>  
<ATTLIST InputAction Description CDATA #REQUIRED>  
<ATTLIST InputAction From CDATA #REQUIRED>  
<ELEMENT OutputAction EMPTY>  
<ATTLIST OutputAction Description CDATA #REQUIRED>  
<ATTLIST OutputAction To CDATA #REQUIRED>  
<ELEMENT SubTask (Task*)>  
```
A.3. Mobile Concrete Presentation Model Schema

<!ATTLIST textfield
    label CDATA #REQUIRED
    length (%length_value; ) #REQUIRED
    password (%option;) #REQUIRED
    action CDATA #IMPLIED>
<!ELEMENT object_edit EMPTY>
<!ELEMENT numerical_edit (textfield)>
<!ELEMENT position_edit EMPTY>
<!ELEMENT control (navigator | activator)>
<!ELEMENT navigator (text_link | button | image_link)>
<!ATTLIST navigator target CDATA #IMPLIED>
<!ELEMENT text_link EMPTY>
<!ATTLIST text_link
    label CDATA #REQUIRED
    action CDATA #IMPLIED>
<!ELEMENT button EMPTY>
<!ATTLIST button
    label CDATA #REQUIRED
    action CDATA #IMPLIED>
<!ELEMENT activator (reset_button | button_and_script | activate_database)>
<!ELEMENT reset_button EMPTY>
<!ATTLIST reset_button
    label CDATA #REQUIRED
    action CDATA #IMPLIED>
<!ELEMENT button_and_script EMPTY>
<!ATTLIST button_and_script
    label CDATA #REQUIRED
    script CDATA #REQUIRED
    action CDATA #IMPLIED>
<!ELEMENT activate_database (query_element+)>
<!ATTLIST activate_database
    label CDATA #REQUIRED
    database.name CDATA #REQUIRED
    table_name CDATA #REQUIRED
    attributes_names CDATA #REQUIRED
    target_presentation CDATA #REQUIRED>
<!ELEMENT query_element EMPTY>
<!ATTLIST query_element
    form_element_id IDREF #REQUIRED
    attribute.name CDATA #REQUIRED>
<!ELEMENT interactive_description (textual | text_link | image_link | button)+>
<!ELEMENT image_link EMPTY>
<!ATTLIST image_link
    alt CDATA #REQUIRED
    src CDATA #REQUIRED
    action CDATA #IMPLIED>
<!ELEMENT only_output (textual | object | description | feedback)>
<!ELEMENT textual (text | text_database)>
<!ELEMENT text ((input_text | text_file), font_settings?)>
<!ELEMENT text_file EMPTY>
<!ATTLIST text_file
    src CDATA #REQUIRED
    type CDATA #IMPLIED>
<!ELEMENT text_database (font_settings?)>
<!ELEMENT object (image)>
<!ELEMENT description ((text?, image?) | table)>
<!ELEMENT feedback EMPTY>
A.4 Adaptation Model Schema

Listing A.4: Schema for Adaptation model

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!---- Simple Type definitions ---->
  <xs:simpleType name="rtType">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="10"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="plasticType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="user"/>
      <xs:enumeration value="platform"/>
      <xs:enumeration value="environment"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="typeOfCondition">
    <xs:restriction base="xs:string">
      <xs:enumeration value="and"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```
A.4. Adaptation Model Schema

```xml
<xs:enumeration value="or" />
<xs:enumeration value="equal" />
<xs:enumeration value="notEqual" />
<xs:enumeration value="lt" />
<xs:enumeration value="gt" />
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="actionLevel">
  <xs:restriction base="xs:string">
    <xs:enumeration value="task" />
    <xs:enumeration value="concrete" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="chNature">
  <xs:restriction base="xs:string">
    <xs:enumeration value="changeAttribute" />
    <xs:enumeration value="changeStructure" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="chOperation">
  <xs:restriction base="xs:string">
    <xs:enumeration value="sort" />
    <xs:enumeration value="add" />
    <xs:enumeration value="delete" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="StringType" />

<xs:complexType name="cParamType">
  <xs:sequence>
    <xs:element name="value" type="StringType" />
  </xs:sequence>
  <xs:attribute name="name" type="StringType" use="required" />
</xs:complexType>
<xs:complexType name="contextType">
  <xs:attribute name="type" type="plasticType" use="required" />
  <xs:attribute name="name" type="StringType" use="required" />
</xs:complexType>
<xs:complexType name="conditionType">
  <xs:choice>
    <xs:element name="ctxParameter" type="cParamType" minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="value" type="StringType" />
  </xs:choice>
  <xs:attribute name="type" type="typeOfCondition" use="required" />
</xs:complexType>
<xs:complexType name="taskType">
  <xs:attribute name="id" type="StringType" />
```
A.4. Adaptation Model Schema

```xml
<xs:complexType name="taskListType">
    <xs:sequence>
        <xs:element name="task" type="taskType" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="all" type="xs:boolean" default="false"/>
</xs:complexType>

<xs:complexType name="elemType">
    <xs:attribute name="value" type="StringType" use="required"/>
</xs:complexType>

<xs:complexType name="elemListType">
    <xs:sequence>
        <xs:element name="element" type="elemType" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="actionType">
    <xs:sequence>
        <xs:element name="elementList" type="elemListType"/>
        <xs:element name="change" type="chType"/>
    </xs:sequence>
    <xs:attribute name="level" type="actionLevel" use="required"/>
</xs:complexType>

<xs:complexType name="chType">
    <xs:sequence>
        <xs:element name="param" type="StringType" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="type" type="chNature"/>
    <xs:attribute name="operation" type="chOperation"/>
</xs:complexType>

<xs:complexType name="actionsType">
    <xs:sequence>
        <xs:element name="levelAction" type="actionType" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="ruleType">
    <xs:sequence>
        <xs:element name="condition" type="conditionType"/>
        <xs:element name="actions" type="actionsType"/>
    </xs:sequence>
    <xs:attribute name="priority" type="rtType" use="optional"/>
</xs:complexType>

<xs:complexType name="rulesType">
    <xs:sequence>
        <xs:element name="rule" type="ruleType" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<!— root description —>
<xs:element name="AdaptationRules" type="rulesType"/>
</xs:complexType>
```
Appendix B

Introduction to UML Diagrams

Each UML diagram is designed to let developers and customers view a software system from a different perspective and in varying degrees of abstraction. UML diagrams represent two different views of a system model:

- **Static (or structural) view**: This presents the static structure of the system using objects, attributes, operations and relationships. Examples include class diagrams.

- **Dynamic (or behavioural) view**: This presents the dynamic behaviour of the system by depicting collaborations among objects and changes to the internal states of objects. Examples include sequence diagrams and activity diagrams.

We present below a short summary of the main structural and behavioural UML diagrams:

- **Use Case Diagram**: It links the Actor (user or other another system) with a Use Case, which is an external view of the system that represents some actions that the Actor needs to do to complete a task. Items can be added until a complete description of the ordering system is derived to capture all of the requirements of the system.

- **Class Diagram**: They model class structure and contents using design elements such as classes, packages and objects. Classes are composed of three parts: a name, attributes, and operations. They can be used to describe three different perspectives of the system: conceptual, specification, and implementation. Class diagrams also display relationships such as generalisation, associations and aggregation.

- **Sequence diagram**: They demonstrate the behaviour of objects in a use case by describing the objects and the messages they pass. The diagrams are read left to right and descending. Conditions for communication are also inserted in the diagram between square brackets. Notice the names of the objects are followed by a colon.

- **Activity Diagram**: The diagrams describe the state of activities by showing the sequence of activities performed, which can be conditional or parallel. Activity diagrams show the flow of activities through the system. Diagrams are read from top to bottom and have branches and forks to describe conditions and parallel activities. The fundamental unit of behaviour specification in the Activity Diagram is an Action.
Figure B.1: Example of use case diagram (http://commons.wikimedia.org/wiki/File:Restaurant-UML-UC.png)

(a) Association

(b) Aggregation

(c) Generalisation

Figure B.2: Examples of class diagrams (http://en.wikipedia.org/wiki/Class_diagram)
An Action takes a set of inputs and convert them into a set of outputs. A *fork* is used when multiple activities are occurring at the same time. A *branch* describes what activities will take place based on a set of conditions. All branches at some point are followed by a merge to indicate the end of the conditional behaviour started by that branch. After the merge all of the parallel activities must be combined by a join before transitioning into the final activity state.

- **Deployment Diagram**: A deployment diagram offers a static view of the physical deployment of processing units and their configuration. The three-dimensional boxes represent nodes, either software or hardware. This may show the hardware used in the system, the software that is installed on that hardware and the middleware used to connect the various processing nodes. Hardware and software components are represented with three-dimensional boxes known as nodes. In the context of UI development, it can be used to show the various documents, libraries and source code, and also software components, such as renderers, context processors, etc.
for(A;B;C)
D;

Figure B.4: For loop using activity diagram (http://en.wikipedia.org/wiki/File:For-loop-diagram.png)

Figure B.5: Example of deployment diagram for web browsing (http://commons.wikimedia.org/wiki/File:UML_Deployment_Diagram.svg)
Appendix C

Context-sensitive User interface Profile

In this section we provide more information on the model composition of CUP and CUP 2.0 UML-based frameworks for the design of UI's. Both approaches have been briefly introduced in section 3.3.2.3.

C.1 CUP

Task Model For the task model, two abstract stereotypes are defined: genericTask and contextual. The former is similar to the original CTT notation, with the exception that they contain one additional type namely environment task which is an entity that is responsible for handling the appropriate environment action. By contrast to the original ConcurTaskTrees (CTT) which allows the specification of target platforms, the Contextual ConcurTaskTrees (CCTT) allows specification of contexts of use i.e. the different situations in which an application can be used. The latter is specified with tagged value (i.e. stereotype attribute) contextOfUse (Figure C.1). The idea is that the contextual task actually specifies a double task, once carried out by one of the stakeholders in the interaction, and one by another entity in the context of use. Therefore, all contextual tasks could be replaced by their non-contextual versions in combination with the newly defined an environment task. The resulting notation is very similar to that of the original notation. For instance, in the original notation tasks are set between brackets (when optional), with a star (for repeated task), and task properties are shown as tagged valued (between curly brackets).

Presentation Model To model the presentation model in CUP, UML deployment model is extended with a set of stereotypes to show the relationship between UI components (such as containment), and indicate the type of interaction they support (for instance, the type of data that the UI components interact with). Meta information about the UI components such as data type they interact with and description of functionality is also provided. Note that meta information is specified as attributes of the node (represented with three-dimensional boxes), and the data type and class manipulated by the user interface component is specified by an association. Four stereotyped associations are supported: select,
interact, trigger and precede. Each of these associations can have tagged attribute of the interaction to indicate for instance relatedProperty (to specify which attribute is modified by which input component), or relatedOperation to specify method is activated for the trigger action. As far as the UI components are concerned, four new stereotypes are introduced in the deployment diagram:

- **<<inputComponent>>**: It is used to allow the user to input data. Initial value is provided but not essential. It is represented by an arrow entering a square indicating data flowing out of the system.
- **<<outputComponent>>**: It is used to present data to the user. It is represented by an arrow leaving a square indicating data flowing out of the system.
- **<<actionComponent>>**: It lets the user to execute an action when it is activated. The actions should not be limited to changes in the UI. It is represented by a right-pointing arrow, within a square symbolising the system.
- **<<groupComponent>>**: It groups other UI components.

The nodes of the deployment diagram can be mapped onto multiple concrete instances by using Artifacts\(^1\).

These representations have similarities with UMLi but have three major differences. CUP extends the UML-metamodel using a profile rather than a custom-built extension which extends the activity model, by introducing six new constructors, rather than adapting a

---

\(^1\)These are libraries, executable, files, documents that you might attach to deployment diagram. They are modelled using the class symbol and an <<artifact>> stereotype.
deployment diagram. In addition, while UMLi has two types of containers (to indicate interaction and containment), CUP has only one type of container (to indicate grouping). The third difference is that in CUP metadata can be specified as attributes of the group components.

**Context Model**  CUP also considers context on two dimensions: how it is gathered i.e. manually (profiled) or automatically sensed or interpreted (detected); and what type of information it encompasses i.e. platform, user, services or environment. Context is modelled using class and package diagrams. Gathered information is described as: \(<\text{profiledContext}>>\) and \(<\text{detectedContext}>>\). Quality of context information can be described using attributes, and multiplicity of sources can be indicated by multiplicity of an association or by multiple associations. Context Collectors, the entities that gather or translate context information and deliver it to the application are modelled with \(<\text{contextCollector}>>\). It is represented as a black dot (representing the context) connected to the centre of a square (representing the system) by an arrow (see Figure C.2).

![Figure C.2: Example of CUP context model ([82])](image)

**Activity Model**  The activity model is the UML's representation of the task model. CUP uses UML 2.0 activity diagram instead of UML 1.4, because the earlier version were not deemed sufficiently expressive to describe the high level details of task models. In addition, in UML 2.0, activity diagrams support hierarchical structure and temporal relations. CUP defines four stereotypes that correspond to the four task types (Figure C.3):

- **user**, which represents an action entirely carried out by the user either cognitive or physical in nature. Note that in CIT, only cognitive actions are considered a user action.
- **system**, which represent an action entirely performed by the system
- **interaction**, which represents an action in which the user interacts with the system
- **environment**, which represents an action performed by an external entity

To integrate the presentation and the context model in to the activity model, object nodes (which are effectively class instances which represent data) and object flows (which are connectors with an arrowhead denoting the direction the data is being passed) are used. The flows link the context collector instances (from context model) and UI component instance

---

2Package diagrams are used to represent the different layers of a software system, be it the structure or the source code of the system.
(from presentation model) to action and activities (activity model). It is possible to show a different effects of the object flows to specify these relations. The source and target are indicated, than their effect and the name of the object views. The source of the object flow and target could be either actions (e.g. system, interaction, etc.), context components, or UI presentation components, and each effect is labelled accordingly to indicate the action type (i.e. activate, interact, suspend, resume and present).

C.2 CUP 2.0

**Application Model**  This model, which uses Class diagram, contains classes of the application logic that links to the UI, also context information and the interface of the context gathering services. The latter are identified using <<context>> and <<contextCollector>> stereotypes. Each property of <<context>> classes can have a stereotype that can indicate how the information is gathered: 1) <<detected>> for context information that is delivered to the application directly from sensors, the detected information is gathered from a context collection (represented by contextCollector stereotype) 2) <<profiled>> for context provided by an application or the user. This distinction is justified by the fact that UI needs to be defined to enable users to modify profiles context information, whereas detected context information requires the setting up of mechanisms to detect information and provide feedback to the user. Note that CUP 2.0 includes stereotypes from the context model of CUP, making context source closer to the application.

**Context Model**  This model is the result of the merging of CUP’s original context model and application model to avoid stereotype duplication. For each context of use, the context model contains a package with the stereotype <<contextOfUse>> Class diagram. A package contains only instances of classes that have stereotype <<context>> from the application model. Each instance specifies one value associated with context parameter. Ranges are specified using minimum and a maximum, or by listing their possible values, and multiple instance of the same class represent alternatives. Values within one context of use are combined using a logical and, and when different contexts of use are combined, a logical or is used so that multiple contexts of use can be associated with a task.

**System Interaction Model**  It describes the interactions of the systems with the user and the environment in which is it executed. It can be used to describe the tasks of the users and application and relevant interaction with the environment in more detail. It is based on an Activity diagram. A task corresponds to an UML Action which is stereotyped <<task>> or derived from it. There are four task categories, based on those found in Contextual CTT notation, with the exception of the abstract task which is represented by the generic stereotype <<task>>. The four task categories are indicated using derived stereotypes from
These are <<userTask>>, <<applicationTask>>, <<interactionTask>> and <<environmentTask>>. While application and interaction tasks are modelled as standard Action instances (represented with round rectangles), the user and environment tasks are modelled with AccepEventAction (introduced in UML 2.0 and represented with concave pentagons) meaning that they need to wait for the occurrence of an event from the user and environment respectively to be enabled. Furthermore, in contrast to CUP, where links with the abstract UI components and context collectors are represented using object flows and object nodes, the links here are implicit i.e. abstract UI component and corresponding task in the system interaction model share the same name.

The <<task>> stereotype also defines a number of tagged values corresponding to task parameters in CTT: optional to indicate whether a task is required or not, repetition to indicate the number of times a task should be executed, manipulatedObject and requiredContext are applicable to basic tasks. Figure C.4 depicts an example of a system interaction model for a museum guide application that shows the position of the user, as well as information about a nearby artwork. It is a simplified model, and will not result in a user friendly application.

**Figure C.4: Example of a system interaction model in CUP 2.0 for a museum guide application ([80])**

**Abstract User Interface Model** The abstract presentation model uses Class diagram instead of deployment model as in CUP, but maintains the original component types i.e. input, output, action and group. Input and output components can have multiple attributes. Each of the attribute has <<uiData>> stereotype, which has a tagged value propertyIn-Class that can be used in case there is a reference to a property of a class. Additional meta-data e.g. label and short description, can also be provided. All operations related to an action component must have the stereotype <<uiAction>> that allows to specify information similar to <<uiData>>. To the visibility of each property and operation with stereotype <<uiData>> and <<uiAction>>, CUP 2.0 exploits the standard UML concept of visibility which defines whether attributes specific classes can be seen and used by other classes. In
C.2. CUP 2.0

In this case, the significance of this setting has been changed to be applicable to interaction concepts. Therefore, when a UI component is set to Public, then it is visible to the current user as well as other users (e.g. viewers). When it is Protected, it is only visible to the user of the UI. When it has a Package visible, that means that it is accessible to other parts of the UI and is not shown to the user of the UI. Finally, Private visibility implies that the content of the component is shown but masked from the user (e.g. password field).

CUP 2.0 also defines some stereotypes for associations between abstract UI components to express relationships other than containment and to indicate constraints on the structure of the UI. These are: <<precede>>, to indicate that the UI component should be presented to a UI before another one, either spatially, temporally or both. It is limited to components contained by the same group to establish order; <<activate>>, to indicate that a component activates another component; and <<update>>, to indicate that the contents of the target UI component is updated by the source UI component.

Deployment Model  CUP 2.0 supports to some extent the transformation of some parts of the models into code templates that can be used to render the final UIs. The deployment model has been specifically proposed to add style to the different user interface skeletons and some design guidelines specifically for the target platform. In this respect, an example of UI generation has been demonstrated using XForms as target language by creating a mapping between AUI model stereotypes (e.g. groupComponent, uiAction and uiData) and XForms tags as shown in the table below.

<table>
<thead>
<tr>
<th>CUP-profile</th>
<th>XForms tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupComponent</td>
<td>group</td>
</tr>
<tr>
<td>- contained number of elements of same type &gt; 1</td>
<td>repeat</td>
</tr>
<tr>
<td>uiData in inputComponent,</td>
<td></td>
</tr>
<tr>
<td>- selectionType is none</td>
<td>input</td>
</tr>
<tr>
<td>- max. selectionCount = 1</td>
<td>select1</td>
</tr>
<tr>
<td>- max. selectionCount &gt; 1</td>
<td>select</td>
</tr>
<tr>
<td>uiData in outputComponent</td>
<td></td>
</tr>
<tr>
<td>uiAction in actionComponent</td>
<td>trigger or submission</td>
</tr>
</tbody>
</table>

Table C.1: AUI components and XForms tags mapping ([80])
Appendix D

Review of Java Mobile GUI Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Special features</th>
<th>Licence</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paxmodept</td>
<td>Replacement library of native JME with more flexible layout manager, improved speed and performance</td>
<td>Ability to combine multiple layout styles</td>
<td>Commercial</td>
<td><a href="http://www.paxmodept.com/paxmodept/products.htm">http://www.paxmodept.com/paxmodept/products.htm</a></td>
</tr>
<tr>
<td>TUWIK</td>
<td>Collection of packages and a set of UI components and animated visual effects (transitions, transformation and motion)</td>
<td>Favourites animation and high quality graphics-based UI's</td>
<td>Commercial</td>
<td><a href="http://www.tricastmedia.com/tuwik/">http://www.tricastmedia.com/tuwik/</a></td>
</tr>
<tr>
<td>J2ME Polish</td>
<td>Set of libraries and tools to ease the development of mobile application and enhance their look-and-feel</td>
<td>CSS support</td>
<td>GPL</td>
<td><a href="http://www.j2mepolish.org/">http://www.j2mepolish.org/</a></td>
</tr>
<tr>
<td>Tinyline</td>
<td>2D Graphics library</td>
<td>Supports vector based graphics (e.g. SVG)</td>
<td>Green, Indie and Standard Licenses</td>
<td><a href="http://www.tinyline.com/products.html">http://www.tinyline.com/products.html</a></td>
</tr>
<tr>
<td>Synclast</td>
<td>Extensible toolkit for creating colorful custom user interfaces on MIDP devices</td>
<td>UI component styling support</td>
<td>GPL / Commercial</td>
<td><a href="http://www.synclast.com/index.jsp">http://www.synclast.com/index.jsp</a></td>
</tr>
<tr>
<td>APIME</td>
<td>GUI Framework for MIDP with SWING structure</td>
<td>Skin and mouse point support</td>
<td>GPL</td>
<td><a href="http://www.java4ever.com/">http://www.java4ever.com/</a></td>
</tr>
<tr>
<td>Fire J2ME</td>
<td>Set of easy-to-use extensible components, offer similar functionality to JME in addition to themes</td>
<td>CSS support, popup panels and screen rotation</td>
<td>LGPL</td>
<td><a href="http://sourceforge.net/projects/fire-j2me/">http://sourceforge.net/projects/fire-j2me/</a></td>
</tr>
<tr>
<td>Micro Window</td>
<td>Toolkit inspired by AWT, SWING and SWT</td>
<td>Customised components, use of bitmap fonts</td>
<td>LGPL</td>
<td><a href="http://j2me-mwt.sourceforge.net/">http://j2me-mwt.sourceforge.net/</a></td>
</tr>
</tbody>
</table>

199
<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
<th>License/Cost</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thinlet</td>
<td>A single class GUI toolkit that parses the hierarchy and properties of the GUI, handles user interaction, and calls business logic. Describes presentation aspects using XML.</td>
<td>LGPL</td>
<td><a href="http://www.thinlet.com/index.html">http://www.thinlet.com/index.html</a></td>
</tr>
<tr>
<td>Lightweight Visual</td>
<td>Lightweight GUI library that supports Java (J2SE/JME) and .NET Only draft version available</td>
<td>GPL / Commercial</td>
<td><a href="http://lwccl.com/j2me.php">http://lwccl.com/j2me.php</a></td>
</tr>
<tr>
<td>Component Library (LwVCL)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>microEWT</td>
<td>Component-based, event-driven UI framework for JME Supports stylesheets and bitmap-based fonts</td>
<td>GPL</td>
<td><a href="http://www.esoco.net/content/view/7/1/">http://www.esoco.net/content/view/7/1/</a></td>
</tr>
<tr>
<td>J4ME (Java For Me)</td>
<td>JME application development framework with replacement for JME own UI framework (LCDUI) Support visual themes, connect to GPS device via bluetooth, improved logging and debugging</td>
<td>Apache 2.0</td>
<td><a href="http://code.google.com/p/j4me/">http://code.google.com/p/j4me/</a></td>
</tr>
<tr>
<td>J4ME (Java For Me)</td>
<td>J4ME application development framework with replacement for JME own UI framework (LCDUI) Support visual themes, connect to GPS device via bluetooth, improved logging and debugging</td>
<td>Apache 2.0</td>
<td><a href="http://code.google.com/p/j4me/">http://code.google.com/p/j4me/</a></td>
</tr>
<tr>
<td>Kuix (J4ME: Java For Me)</td>
<td>It provides most graphical elements to create advanced user graphical interfaces Uses an XML/CSS approach to describe the screens and the user actions in the application</td>
<td>GPL / Commercial</td>
<td><a href="http://www.kalmeo.org/projects/kuix">http://www.kalmeo.org/projects/kuix</a></td>
</tr>
<tr>
<td>TagME</td>
<td>Plugin-based architecture mobile GUI framework Supports XML for the description of the interface and script language</td>
<td>Commercial</td>
<td><a href="http://www.tagme.com/">http://www.tagme.com/</a></td>
</tr>
<tr>
<td>kUI</td>
<td>Canvas-based replacement for JME's LCDUI UI component styling support</td>
<td>GPL / Proprietary</td>
<td><a href="http://koobjects.org/kui/index.php">http://koobjects.org/kui/index.php</a></td>
</tr>
<tr>
<td>Swing ME</td>
<td>JME implementation of Swing Under development</td>
<td>LGPL</td>
<td><a href="http://swingme.sourceforge.net/">http://swingme.sourceforge.net/</a></td>
</tr>
<tr>
<td>Lightweight UI Toolkit</td>
<td>UI toolkit library for Java ME Supports theming, transitions, animation, SVG, etc.</td>
<td>GPL</td>
<td><a href="https://lwuit.dev.java.net/">https://lwuit.dev.java.net/</a></td>
</tr>
<tr>
<td>for Java ME (LWUIT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JavaFX Script</td>
<td>Scripting language to create rich internet applications on desktops and mobiles Describe the functionality of UI declaratively</td>
<td>GPL</td>
<td><a href="https://openjfx.dev.java.net/">https://openjfx.dev.java.net/</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix E

Amazon Book Browser

E.1 Final UI Model Descriptions

E.1.1 Thinlet

Details.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<panel name="infopanel" columns="1" weighty="1" gap="4" top="4" left="4" bottom="4"
right="4">
<tabbedpane name="tabbedpane" weightx="1" weighty="1">
<panel name="info" columns="2" gap="4" top="4" left="4" bottom="4" right="4" weightx="1" weighty="1">
<label text="Title:" valign="top"/>
<label name="title" weightx="1"/>
<label text="Author:" valign="top"/>
<label name="authors" weightx="1"/>
<label text="Released:" valign="top"/>
<label name="date" weightx="1"/>
<label text="Publisher:" valign="top"/>
<label name="publisher" weightx="1"/>
<label text="Average Rating:" valign="top"/>
<label name="avg.rating" icon="avg_rating.gif" weightx="1"/>
<label text="List Price:" valign="top"/>
<label name="listprice" weightx="1"/>
<label text="Our Price:" valign="top"/>
<label name="offerprice" weightx="1"/>
</panel></tab>
<tab text="Details">
<panel name="more" columns="2" gap="4" top="4" left="4" bottom="4" right="4" weightx="1" weighty="1">
<label text="Preview:" valign="top"/>
<label name="cover" icon="book_cover.gif" weightx="1"/>
<label text="Book Description:" valign="top"/>
<label name="description" weightx="1"/>
<label text="New Price:" valign="top"/>
```
E.1. Final UI Model Descriptions

<tabfolder name="infopanel">
  <tabitem label="Info">
    <scrollcontainer name="info" style="layout:inlinelayout(false, fill); padding: 5" usemarkers="false">
      <listitem>
        <text class="label">Title:</text>
        <text style="align: left">@{ title}</text>
      </listitem>
      <listitem>
        <text class="label">Author:</text>
        <text style="align: left">@{ authors}</text>
      </listitem>
      <listitem>
        <text class="label">Released:</text>
        <text style="align: left">@{ date}</text>
      </listitem>
      <listitem>
        <text class="label">Publisher:</text>
        <text style="align: left">@{ publisher}</text>
      </listitem>
      <listitem>
        <text class="label">Average Rating</text>
        <picture style="align: left" src="avg_rating.gif"></picture>
      </listitem>
    </scrollcontainer>
  </tabitem>
</tabfolder>

E.1.2 Kuix

Details.xml

<screen>
  <tabfolder name="infopanel">
    <tabitem label="Info">
      <scrollcontainer name="info" style="layout:inlinelayout(false, fill); padding: 5" usemarkers="false">
        <listitem>
          <text class="label">Title:</text>
          <text style="align: left">@{ title}</text>
        </listitem>
        <listitem>
          <text class="label">Author:</text>
          <text style="align: left">@{ authors}</text>
        </listitem>
        <listitem>
          <text class="label">Released:</text>
          <text style="align: left">@{ date}</text>
        </listitem>
        <listitem>
          <text class="label">Publisher:</text>
          <text style="align: left">@{ publisher}</text>
        </listitem>
        <listitem>
          <text class="label">Average Rating</text>
          <picture style="align: left" src="avg_rating.gif"></picture>
        </listitem>
      </scrollcontainer>
    </tabitem>
  </tabfolder>
</screen>
E.2 Amazon Book Browser Deployment Proposal

In the following sections, we will explain how the different transformation modules developed for Amazon book browser could be integrated into a complete architecture that can be used to deploy and enable the application over the network. For this, we base our recommendations on the reference architecture presented in section 5.2.1, and also on prototype software that have been partly implemented to test the feasibility of using a model-based approach as part of a distributed architecture, such as a message-oriented middleware. In fact, these modules can be adapted to suit the needs of a variety of applications beyond the Amazon book browser application.

There are three parts to the architecture: Amazon's web service provider, a client-side client to render the UI and request adaptation, and a server application this is responsible for context management and UI generation. The top-level view of the components that makes up the Amazon book browser architecture is shown in Figure E.1.

![Figure E.1: Deployment diagram of the Amazon book browser](image)

Figure E.2 shows the steps through which the client (a) and the server (b) go through to exchange context information, apply adaptation, generate final UI and transmit to the client for rendering.

E.2.1 Server side

It is responsible for the generation of the UI, management of the context and the communication with the clients. Based on the service requested and the contextual condition encountered, the corresponding task model is retrieved from the repository. Context information is acquired from the client, interpreted and managed by the context module within the server-side application. The information is passed to the context adaptation engine, which retrieves the service and user-specific adaptation rules (which are specified at design time).

The server-side application provide four categories of functionality, which are:

- **Interpreters:** It includes components used for the interpretation of content information, rules and task models.
- **Transformers:** It includes components used for the progressive transformation of the models according to context information and adaptation rules.
Figure E.2: Flow diagram of the UI management system
E.2. Amazon Book Browser Deployment Proposal

- **Adapters/Connectors**: It includes database and communication adapters.
- **Repositories**: It includes a repository for task models, client profiles and rules.
- **Core**: This concerns the business logic component that manages the other units.

Figure E.3 depicts the different software components that would make up the server side of the architecture.

E.2.2 Client side

In addition to acting as a client to Amazon's web service provider, the client side application needs to be adapted to support real-time capture of the immediate context around the device and transmits it to the UI server for processing. The context change triggers the collection of context information and transmission of complete or differential context profile. It also sends a request to generate/update its UI. To cater for situations where the device is disconnected from the network, the UI is retrieved from the local cache (device storage or a special database). That cache keeps track of the last version of the final UI, and stores data related to the last book viewed. Once it receives the new or updated final UI, it calls the GUI rendering engine and displays the information on the device. Figure E.4 depicts a potential composition of the client side of the Amazon book browser.

E.2.3 Client-server communications

For the exchange of context information and user interface descriptions between the client and server, a Message-oriented Middleware (MoM) is proposed. It operates on the principles of message passing and/or message queuing supporting both synchronous and asynchronous interactions between distributed computing processes. It is advantageous compared to standard client-server approaches that use request-response pattern in a number of aspects:
• Asynchronous delivery which means that the sender and the receiver do not need to connect to the network at the same time. This is particularly useful for our application scenario since wireless connection can be intermittent.

• Asynchronous messaging systems are highly scalable, and are capable of serve a great number of devices.

• Avoid the large overhead associate with having active clients and server requiring a persistent connection, or use some sort of polling technique.

• Decoupling between the request and reply phased on the exchange. Unlike in client/server scenarios, once a request message is set from the client, the client can be freed and continue with other processes.

• Anonymity makes it unnecessary to have direct knowledge of the producer and the consumer, beyond the messaging interactions. This feature makes it possible to keep the messaging system and change the underlying sub-system, without affecting.

We have in fact experimented with the use of a Java-specific implementation called JMS\(^1\) to enable the communication of XML-based context profiles and final UI descriptions between the client and the server side. JMS is set of specification developed by Sun to provide a common platform for Java to access different MoM systems, and is supported by a number of business and open source enterprise messaging products. JMS supports two forms of messaging: point-to-point and publish-and-subscribe, and both can be implemented as asynchronous or synchronous message transmissions. Publish-and-subscribe permits messages to be broadcast to multiple subscribers and is an excellent mechanism for many-to-many conversations. Point-to-point messaging is useful when conversations are one-to-one, and in conversations in which the receiver processes a given message just once. Receivers register their interest in those messages by subscribing to a topic or by listening to a queue. JMS supports two modes of delivery: persistent (ensures the delivery of the message regardless of failure conditions), and non-persistent ("best effort" for the delivery of the message). The advantages of using JMS\(^2\) are numerous (compared to previous middleware approaches):

\(^1\)http://java.sun.com/products/jms/

\(^2\)On the mobile side, a CLDC 1.0/MIDP 2.0-enabled library has been used called jtom v2.1.3 (http://www.j2mob.com/). Messaging functionality is provided by JBoss Application Server 4.0.0 application provider.
• Compared to exchanging XML over HTTP, JMS offers quick, easy and robust solution (despite little performance overhead).

• JMS offers off-the-shelf reliability.

• Ability to send messages and files bi-directionally.

• Ability to have multiple clients listening to a particular queue.

• Ability to send/receive message synchronously and asynchronously.

We used a queue messaging paradigm because we consider a one-to-one model whereby a message sent by the producer gets delivered to one of the interested consumers asynchronously over HTTP. Other approaches were also considered for asynchronous messaging in a mobile environment using Short Messaging Service (SMS) or Multimedia Messaging Service (MMS). However, they require a JMS-compatible plugin on the server side and is used to manage the sending/reception of messages. In addition, this puts an additional burden on the mobile device such as the need to store the message before processing them, and extracting the data from the messages before using them for adaptation. For these reasons, it has been decided to exchange the message over HTTP directly. Since JMS is used for the message exchange capabilities, no heavy processing is actually assigned to the portable device. Most of the processing is handed over to the server for the generation and adaptation of UI models. The client simply sends its context profile, issues commands and receive updates of the UI description.

The exchange between the server and the client is achieved through message passing. The exchanged message is composed of control and data parts. The control part indicates what instruction needs to be executed on the other end, and is represented by short string. In the current version, the following control instructions are supported (Figure E.5):

CON To establish connection between UI client and UI server
TERM To terminate the connection.
REQPRF To request the context profile (from the UI client).
RECPRF To notify the other end that the message contains a context profile (from the client).
REQUID To request the high-level UI description (from the server).
REGUID To notify the other side that the message contains a high-level UI description (from the server).
RECDPRF To send only a differential context profile, then the client can send only a differential context profile, with respect to the one sent earlier.
OK To acknowledge the acceptance of a connection request, and a successful reception of the profile or UI description.

The data part contains the content of the full context profile and the UI description respectively in the case of the RECPRF and REGUID instructions. A differential context profile is inserted when using RECDPRF command. The data part is in effect a serialised version of the XML file, and is not compressed.
Figure E.5: Communication protocol for context profile management