Hasselt University
and transnational University of Limburg
School of Information Technology

High-Level User Interface Models for Model-Driven Design of Context-Sensitive User Interfaces

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science
at the transnational University of Limburg
to be defended by

Jan Van den Bergh

on October 25, 2006
Supervisor: Prof. Dr. Karin Coninx

1999 – 2006
Acknowledgements

At the start of this dissertation I want to thank some people without whose support I would not have been able to complete this work. The first person to thank is my promotor, Prof. Dr. Karin Coninx, who offered me the possibility of a second chance in the human-computer interaction team when my research in computer graphics did not have the necessary success to complete a doctoral dissertation. I want to thank her for her support and guidance in scientific writing and the many suggestions she made to improve the quality of the papers and this dissertation. Thanks also to all the other members of the jury, Prof. Dr. Frank Van Reeth, Prof. Dr. Eddy Flerackers, Prof. Dr. Jan Van den Bussche, Dr. Ronald Westra, Prof. Dr.-Ing. habil. Peter Forbrig, and last, but certainly not least, Prof. Dr. Nuno Jardim Nunes.

I also want to thank prof. Dr. Eddy Flerackers and prof. Dr. Frank Van Reeth for giving me the opportunity the start my research career at the Expertise Centre for Digital Media, at that time part of the Limburgs Universitair Centrum. I want to thank Frank especially for guiding my work during the first couple of years and for offering me the opportunity to experience my first scientific conference and for guiding me towards my first scientific conference paper, which wouldn’t have been possible without the generous contributions of prof. Dr. Fabian Di Fiore and Dr. Johan Claes. Thanks to Ingrid Konings and Roger Claes for their assistance with the organization and paper work involved with the trips to the conferences to present and discuss the work of colleagues and myself.

Special thanks goes also to prof. Dr. Kris Luyten who introduced me to the exciting world of model-based design of user interfaces and thus paved the way for this research presented in this work. I also thank him and the other members of the “human-centered design group” with whom I had many discussions about the research presented in this work as well as on other topics. Of this group, I should specifically mention Tim Clerckx, Chris Vandervelpen,
Jo Vermeulen, Geert Houben and Frederik Winters.

Thanks also to all the people with whom I shared an office at the Expertise Centre for Digital Media to provide me with an inspiring and enjoyable atmosphere to perform the research during the years to prepare this doctoral dissertation.

I also want to thank Alcatel and IBBT for funding the last year of my research and thus provide me the opportunity to finish this work. Special thanks goes also to the members of the Participate team, both at Alcatel and at the Expertise Centre for Digital Media for their valuable feedback and help, especially on the creation of SpieLan and all the preliminary versions that preceded it. In this regard, Steven Huypens deserves an honourable mention because of the well appreciated feedback and help in the realization of the tool and the prototypes that were realized.

Special thanks to all people that helped in creating and evaluating the usability test of SpieLan: Steven Huypens, Mieke Haesen, Kristof Thys, Luc Adriaens and Ingrid Konings. It wouldn’t have been possible without them and of course all the people that participated in the test.

I also want to thank my parents who supported me in my studies, work, sports and other choices in my life and were always ready to help.

Last but certainly not least, I want to thank my best friend and wife, Merel Schols, for her support at the times that were the most difficult for me and for ensuring that I took the necessary breaks and performed enough sports to be healthy following the motto: Mens sana in corpore sano.
Abstract

The usage of multi-purpose electronic devices that are available to the mass public has increased significantly. The size of these devices has caused a shift in the places where these devices are used. The small size of devices such as personal digital assistants and smart phones does not require anymore that a user sits down at a desk to do tasks such as searching information over the internet or managing appointments, sending messages, ... Many users, however, do not only use this mobile device to accomplish these tasks but prefer the usage of a larger computing device, such as a laptop or a personal computer. This often means that a user has to learn different interaction methods and different user interface structures.

At the same time, communities are increasingly independent of the location of their participants; most of the communication is done through communication devices. When in the past people came together in a local pub, to enjoy entertaining activities, today more and more people do not necessarily meet at a common physical place, but rather meet online. This has increased the interest of the telecom industry to not only make entertainment more interactive, but also to introduce the social components into digital broadcasts. They want to extend the social impact that online chatservices have to other fields. Participatory television should be for watching television in group, what internet chat is for face-to-face conversations. Because the common meetingplace is gone, participants do not necessarily have to be in front of a television at home, but can be on the road, also requiring some level of context-sensitivity.

In this dissertation, some of the challenges that these new types of applications pose to design of their user interfaces are addressed, more specifically the integration of context sensitivity in general and presence for participatory television. The premise for all the work presented in this dissertation is that in order to produce interactive applications that work well in these demanding circumstances, a well-structured and systematic approach is to be taken, such
as a model-based or model-driven design methodology.

For this design methodology, we analysed the current approaches for their support of context-sensitiveness and came to the conclusion that currently context-sensitiveness and context influences are not extensively covered, especially at higher levels of abstraction. A first contribution of our work is the integration of context-sensitivity in the user-task model. This work is based on one of the most popular notation for user-task modeling, the ConcurTaskTrees notation. We extended this notation to include context tasks resulting in a notation that was a strict superset of the ConcurTaskTrees notation, the Contextual ConcurTaskTrees.

After task specification, the actual modeling of the interactive application may be started. This modeling is often divided into two parts; user interface modeling and modeling of the application core. This dissertation focuses on the former, but wants to ensure that the results can be used in a unified approach to the design of interactive systems. Indeed, it is recognized that usability problems often are also caused by problems in the design of the application core or its binding with the user interface. At this level, we introduce context-sensitivity in high-level models expressed using the Unified Modeling Language (UML). For this purpose a profile is presented that extends the UML with the necessary constructs.

While these models can provide good foundations, they are not always the most suitable notation in all situations. One of these situations is where the designers are very unlikely to have knowledge of the UML and deal with problems whose solution cannot be easily covered by the UML. The design of participatory television events is one of those situations. Participatory television also has a prominent place for context sensitiveness; awareness of the other users. In a last contribution, this dissertation describes a language that allows to describe such events. In the dissertation, the relation of the constructs with those present in the UML will also be discussed.
# Contents

Acknowledgments ........................................ iii

Abstract ................................................. v

Contents ................................................ viii

List of Figures ........................................ xii

List of Tables ........................................... xiv

I  Introduction ........................................ 1

1  Introduction ........................................ 3
   1.1  Motivation and Aim .............................. 3
   1.2  Detailed Overview ............................... 5

2  Models for User Interface Engineering ............ 7
   2.1  Introduction ................................... 7
   2.2  Qualitative Requirements ....................... 8
   2.3  Context-sensitiveness ......................... 9
   2.4  Model requirements ............................ 13
   2.5  Models in Existing approaches ................ 16
   2.6  Software engineering and models for HCI .... 26
   2.7  Discussion .................................... 32
   2.8  Conclusion ................................... 34
II User Interface Models using UML

3 Task modeling for Context-Sensitive Interactive Applications 37
  3.1 Introduction ........................................ 37
  3.2 ConcurTaskTrees .................................... 38
  3.3 Related Work ........................................ 39
  3.4 Contextual ConcurTaskTrees ......................... 41
  3.5 Example ............................................. 42
  3.6 Contextual ConcurTaskTrees and UML ............... 44
  3.7 Discussion .......................................... 46
  3.8 Conclusion .......................................... 49

4 CUP: Context-sensitive modeling 51
  4.1 Introduction ......................................... 51
  4.2 Related Work ........................................ 52
  4.3 CUP: Overview ...................................... 55
    4.3.1 Goals in designing the profile ................. 55
    4.3.2 The profile .................................... 56
  4.4 Models and diagrams ................................ 56
    4.4.1 Modeling the presentation ..................... 59
    4.4.2 Modeling context ................................ 65
    4.4.3 Modeling activities .............................. 66
    4.4.4 Integration of Models ............................ 69
  4.5 Discussion .......................................... 72
    4.5.1 Usability ........................................ 72
    4.5.2 Model Requirements ............................. 73
    4.5.3 Integration with Software Engineering ......... 74
  4.6 Conclusion .......................................... 75

5 CUP 2.0: Modeling multi-context user interface 77
  5.1 Introduction ......................................... 77
  5.2 Model Overview ...................................... 78
  5.3 Application Model ................................... 79
  5.4 Context Model ...................................... 81
  5.5 System Interaction Model ............................. 84
  5.6 Abstract User Interface Model ...................... 88
  5.7 Prototype Generation ................................ 93
  5.8 Analysis of changes in CUP .......................... 95
  5.9 Conclusion .......................................... 97
### CONTENTS

#### 6 Tool Support and Evaluation  99
- **6.1 Tool Support** .............................................. 99
- **6.2 Evaluation** .................................. 103
  - 6.2.1 Analysis of the UML-based Notations ................... 103
  - 6.2.2 Usage of the UML-based Models ...................... 106
- **6.3 Conclusion** .............................................. 108

#### III SpIeLan: A Domain-Specific Language  111

#### 7 SpIeLan: a Domain Specific Modeling Language  113
- **7.1 Introduction** ............................................ 113
- **7.2 Staged Participatory Multimedia Events** .................. 114
- **7.3 Related work** ......................................... 117
- **7.4 Overview of SpIeLan** .................................... 119
- **7.5 Scenario Model** .......................................... 121
  - 7.5.1 Requirements ........................................ 121
  - 7.5.2 Modeling Constructs ................................. 122
  - 7.5.3 Example ............................................ 124
- **7.6 Screen Model** .............................................. 124
  - 7.6.1 Requirements ........................................ 124
  - 7.6.2 Modeling Constructs ................................. 126
  - 7.6.3 Example ............................................ 128
- **7.7 Template Model** .......................................... 129
  - 7.7.1 Requirements ........................................ 129
  - 7.7.2 Modeling Constructs ................................. 130
  - 7.7.3 Example ............................................ 134
- **7.8 Conclusion** .............................................. 136

#### 8 Tool Support and Evaluation of SpIeLan  137
- **8.1 Tool Support** ............................................ 137
  - 8.1.1 Introduction ......................................... 137
  - 8.1.2 Generation of Prototypes ............................. 139
- **8.2 Evaluation** .............................................. 145
  - 8.2.1 Model Coverage ...................................... 145
  - 8.2.2 Comparison with CUP and Contextual ConcurTaskTrees 147
  - 8.2.3 Usability Test ....................................... 154
- **8.3 Conclusion** .............................................. 160
List of Figures

2.1 Three dimensional context [Schmidt 99] .................. 10
2.2 Different kinds of context influences ................ 13
2.3 HCI models and the important information types contained in them ........................................... 14
2.4 Models in Teresa ........................................... 19
2.5 Models in DynaMo-AID .................................. 20
2.6 Models in ArtStudio ...................................... 21
2.7 Models in Mobi-D ........................................ 22
2.8 Models in XIML ............................................ 23
2.9 Wisdom models and methodology .................... 24
2.10 Models in UMLi .......................................... 25
2.11 UsiXML models and methodology ................... 27
2.12 The activities in the MDA process .................... 32

3.1 Task categories in ConcurTaskTrees .................. 39
3.2 Integration of subtrees for different contexts of use in a single task model ..................................... 40
3.3 Additional task categories in Contextual ConcurTaskTrees .... 41
3.4 Contextual ConcurTaskTree model for a museum visit ........................................................... 43
3.5 UML profile for Contextual ConcurTaskTrees ........ 45
3.6 The museum visit example of figure 3.4 using the Contextual ConcurTaskTrees in UML .................. 47

4.1 Example of a location query using the ConFab infrastructure in the Lemming instant messenger [Hong 04] .......... 53
4.2 Example of a context model for a context-aware communication program [Henricksen 06] .......................... 54
4.3 Context-sensitive User interface Profile ................ 57
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>Canonical models used for context sensitive user interfaces</td>
<td>58</td>
</tr>
<tr>
<td>4.5</td>
<td>Abstract presentation specification for creating an appointment</td>
<td>60</td>
</tr>
<tr>
<td>4.6</td>
<td>Alternative concrete user interfaces based on abstract presentation in figure 4.5</td>
<td>62</td>
</tr>
<tr>
<td>4.7</td>
<td>Mapping of the model in figure 4.5 to the graphical user interface in figure 4.6(a)</td>
<td>64</td>
</tr>
<tr>
<td>4.8</td>
<td>Context specification</td>
<td>66</td>
</tr>
<tr>
<td>4.9</td>
<td>Action stereotypes</td>
<td>67</td>
</tr>
<tr>
<td>4.10</td>
<td>Activity specification for the creation of an appointment</td>
<td>68</td>
</tr>
<tr>
<td>4.11</td>
<td>Example of activity model using swimlanes</td>
<td>71</td>
</tr>
<tr>
<td>4.12</td>
<td>Model-Coverage of CUP</td>
<td>74</td>
</tr>
<tr>
<td>5.1</td>
<td>Overview of the models supported by the UML profile CUP 2.0</td>
<td>78</td>
</tr>
<tr>
<td>5.2</td>
<td>Example of user interface deployment model: A context-sensitive mobile museum guide</td>
<td>79</td>
</tr>
<tr>
<td>5.3</td>
<td>Stereotypes of the UML profile CUP 2.0 relevant for the application model</td>
<td>80</td>
</tr>
<tr>
<td>5.4</td>
<td>Example of application model: A context-sensitive mobile museum guide</td>
<td>81</td>
</tr>
<tr>
<td>5.5</td>
<td>Screenshot of iCAP [Sohn 03]</td>
<td>82</td>
</tr>
<tr>
<td>5.6</td>
<td>Stereotypes of the UML profile CUP 2.0 relevant for the context model</td>
<td>82</td>
</tr>
<tr>
<td>5.7</td>
<td>Example of context model: A context-sensitive mobile museum guide</td>
<td>83</td>
</tr>
<tr>
<td>5.8</td>
<td>Stereotypes of the UML profile CUP 2.0 relevant for the system interaction model</td>
<td>85</td>
</tr>
<tr>
<td>5.9</td>
<td>Example of system interaction model: A context-sensitive mobile museum guide</td>
<td>88</td>
</tr>
<tr>
<td>5.10</td>
<td>Stereotypes of the UML profile CUP 2.0 relevant for the abstract user interface model</td>
<td>89</td>
</tr>
<tr>
<td>5.11</td>
<td>Example of abstract user interface model: A context-sensitive mobile museum guide</td>
<td>91</td>
</tr>
<tr>
<td>5.12</td>
<td>CAP version of the group components IntroScreen and Map Display</td>
<td>92</td>
</tr>
<tr>
<td>5.13</td>
<td>Final interface based on AUIM in figure 5.11</td>
<td>94</td>
</tr>
<tr>
<td>6.1</td>
<td>Modeling Contextual ConcurTaskTrees within MagicDraw, a popular UML editor</td>
<td>100</td>
</tr>
<tr>
<td>6.2</td>
<td>Modeling CUP 2.0 within MagicDraw, a popular UML editor</td>
<td>103</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td>Models and model elements in proposed approach</td>
<td></td>
</tr>
<tr>
<td>6.4</td>
<td>Model elements used in the user interface deployment model</td>
<td></td>
</tr>
<tr>
<td>6.5</td>
<td>Artifacts created during the design process</td>
<td></td>
</tr>
<tr>
<td>7.1</td>
<td>Scenario of an auction show with screenshots from a mock-up</td>
<td></td>
</tr>
<tr>
<td>7.2</td>
<td>The different roles users participating in the auction SPME play and their locations</td>
<td></td>
</tr>
<tr>
<td>7.3</td>
<td>Screenshot of RCX Code within the programming environment</td>
<td></td>
</tr>
<tr>
<td>7.4</td>
<td>Overview of the models supported by the SpIeLan</td>
<td></td>
</tr>
<tr>
<td>7.5</td>
<td>The envisioned SPME creation process using SpIeLan</td>
<td></td>
</tr>
<tr>
<td>7.6</td>
<td>Modeling constructs used in the scenario model</td>
<td></td>
</tr>
<tr>
<td>7.7</td>
<td>The scenario model for the auction SPME</td>
<td></td>
</tr>
<tr>
<td>7.8</td>
<td>A screen model for the template Bidding or StartQuestion in figure 7.7</td>
<td></td>
</tr>
<tr>
<td>7.9</td>
<td>Chat specification - proposed notation</td>
<td></td>
</tr>
<tr>
<td>7.10</td>
<td>Template constructs</td>
<td></td>
</tr>
<tr>
<td>7.11</td>
<td>Semantics of the SpIeLan Interaction and MultiUserInteraction</td>
<td></td>
</tr>
<tr>
<td>7.12</td>
<td>Chat specification - proposed notation</td>
<td></td>
</tr>
<tr>
<td>8.1</td>
<td>Tool for creation of SpIeLan models and generation of prototypes</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>An example of a screen model</td>
<td></td>
</tr>
<tr>
<td>8.3</td>
<td>Prototype example and its structure</td>
<td></td>
</tr>
<tr>
<td>8.4</td>
<td>Generation of the templates and scenes</td>
<td></td>
</tr>
<tr>
<td>8.5</td>
<td>Generation of the participants</td>
<td></td>
</tr>
<tr>
<td>8.6</td>
<td>Generation of the participants and user interface controls</td>
<td></td>
</tr>
<tr>
<td>8.7</td>
<td>Model coverage of SpIeLan</td>
<td></td>
</tr>
<tr>
<td>8.8</td>
<td>SpIeLan models for a chat template. The full-size versions of these models are shown in figure 7.9 and figure 7.12</td>
<td></td>
</tr>
<tr>
<td>8.9</td>
<td>Cooperative part of Chat specification - Contextual (Cooperative) ConcurTaskTrees</td>
<td></td>
</tr>
<tr>
<td>8.10</td>
<td>Role specific parts of chat specification - Contextual (Cooperative) ConcurTaskTrees</td>
<td></td>
</tr>
<tr>
<td>8.11</td>
<td>Chat specification using UML 2.0 and CUP</td>
<td></td>
</tr>
<tr>
<td>B.1</td>
<td>Stereotypes for actors and alternative notations defined for Wisdom [Nunes 01]</td>
<td></td>
</tr>
<tr>
<td>B.2</td>
<td>Stereotypes on analysis and interaction model [Nunes 01]</td>
<td></td>
</tr>
<tr>
<td>B.3</td>
<td>Example presentation model in Wisdom notation [Nunes 01]</td>
<td></td>
</tr>
<tr>
<td>B.4</td>
<td>Example task model in Wisdom notation [Nunes 01]</td>
<td></td>
</tr>
<tr>
<td>B.5</td>
<td>A presentation model using UMLi [da Silva 02]</td>
<td></td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>B.6</td>
<td>A activity model using UMLi [da Silva 02]</td>
<td>197</td>
</tr>
<tr>
<td>B.7</td>
<td>A Task and domain specification using their original syntax (resp. CTT and UML) [Limbourg 04a]</td>
<td>198</td>
</tr>
<tr>
<td>B.8</td>
<td>A mapping rule between that task model and the abstract user interface model [Limbourg 04a]</td>
<td>199</td>
</tr>
<tr>
<td>B.9</td>
<td>An abstract user interface model using the graphical notation proposed by Limbourg [Limbourg 04a]</td>
<td>199</td>
</tr>
<tr>
<td>B.10</td>
<td>An concrete user interface model using the textual notation for UsiXML [Limbourg 04a]</td>
<td>200</td>
</tr>
<tr>
<td>B.11</td>
<td>An concrete user interface model using a Windows look-and-feel [Limbourg 04a]</td>
<td>201</td>
</tr>
<tr>
<td>B.12</td>
<td>The graphical notation for a subset of the UsiXML models as used in IdealXML [Montero 05]</td>
<td>202</td>
</tr>
<tr>
<td>C.1</td>
<td>CAP example: a film-clip viewer [Constantine 03b]</td>
<td>204</td>
</tr>
<tr>
<td>D.1</td>
<td>UML history</td>
<td>208</td>
</tr>
<tr>
<td>D.2</td>
<td>MDA modeling levels</td>
<td>208</td>
</tr>
<tr>
<td>D.3</td>
<td>UML diagrams [Wikipedia 06]</td>
<td>210</td>
</tr>
<tr>
<td>E.1</td>
<td>Example used to explain notation used in scenario model</td>
<td>215</td>
</tr>
<tr>
<td>E.2</td>
<td>Example used for questions about scenario model</td>
<td>216</td>
</tr>
<tr>
<td>E.3</td>
<td>Questions about scenario model</td>
<td>217</td>
</tr>
<tr>
<td>E.4</td>
<td>Examples of screen models</td>
<td>218</td>
</tr>
<tr>
<td>E.5</td>
<td>Questions about screen model</td>
<td>219</td>
</tr>
<tr>
<td>E.6</td>
<td>Task: adaptation of the scenario model shown in figure E.2</td>
<td>220</td>
</tr>
<tr>
<td>E.7</td>
<td>Task: adaptation of a screen model</td>
<td>221</td>
</tr>
<tr>
<td>E.8</td>
<td>Questions about adaptation of models</td>
<td>222</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Qualitative evaluation of existing notations .......................... 16
2.2 Splitting and linking of models in existing notations ............ 17
2.3 Expressiveness of and tools supporting existing notations ... 28
2.4 Cockburn scale. Vertical scale: defects cause loss of . Hori-

zontal scale: number of people. ................................. 29

3.1 Task properties in ConcurTaskTrees notation ..................... 39

4.1 Stereotypes for associations between user interface components
and datatypes ......................................................... 61
4.2 Relation between CUP and CAP ................................. 65
4.3 Types of object flows supported to link context collector in-
stances and user interface component instances to actions and
activities ............................................................. 70

5.1 Icons of task categories in ConcurTaskTrees, Contextual Con-
curTaskTrees and CUP 2.0 ........................................ 84
5.2 Temporal operators in ConcurTaskTrees and corresponding ac-
tivity diagram notation ........................................... 87
5.3 CUP 2.0 stereotyped Elements and XForms counterparts .... 93

6.1 Qualitative Evaluation ............................................. 104

7.1 Most common user interface components ......................... 127
7.2 Interaction component events .................................... 133
7.3 Visibility user interface components specified as source or target
of data in Interactions and MultiUserInteractions ............. 134

8.1 Mapping of model elements ...................................... 148


<table>
<thead>
<tr>
<th></th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.2</td>
<td>Answers to questions with objective results for all test participants and subgroups</td>
<td>156</td>
</tr>
<tr>
<td>A.1</td>
<td>Overview of the significance of a selection of cognitive dimensions for certain tasks as defined in [Green 98]</td>
<td>191</td>
</tr>
<tr>
<td>C.1</td>
<td>Canonical abstract user interface components</td>
<td>205</td>
</tr>
<tr>
<td>D.1</td>
<td>UML symbols deployment diagram</td>
<td>210</td>
</tr>
<tr>
<td>D.2</td>
<td>UML symbols class and package diagram</td>
<td>211</td>
</tr>
<tr>
<td>D.3</td>
<td>UML symbols activity diagram</td>
<td>212</td>
</tr>
</tbody>
</table>
Part I

Introduction
Chapter 1

Introduction

1.1 Motivation and Aim

In recent years one has witnessed several major changes in the usage of interactive applications on computerized devices. A first change is that the usage of interactive applications has reached out of the well-defined environments of the office. Interactive applications are now used by people of all ages and backgrounds and in a plethora of different environments. The changes in environments can be very dynamic. Because the small form factor of some of the electronic devices, such as mobile phones and personal digital assistants, they can be used by people in many different contexts. Their small form-factor makes interacting with these devices more difficult. To ease interaction with these devices, they need to be context-sensitive [Canny 06].

A second change is that users nowadays have more than one electronic device on which they would like to use similar applications; they would like to have applications such as calendars, web browsers, contact management software available at all times and on all devices. This means that these applications should be available on all devices and preferably have a similar look-and-feel to decrease the learning curve. Furthermore, the small form factor and the limited interaction capabilities of mobile devices require that the interactive applications make optimal use of the device they are used on and preferably can adapt in predictable ways to a changing environment.

The increasing usage of electronic communication devices and the ubiquitous availability of transport, have also made the concept of “a global village” more and more of a reality. The electronic devices are increasingly used as
a means of communication between friends, family-members and co-workers which can be located in distant places. This evolution also means that traditional local social interactions in face-to-face meetings also have to be performed using these electronic devices, which have to communicate the presence of the users.

When these changes are related back to the design of interactive applications, one can see that multiple challenges arise: applications are used in changing environments, by users with different needs on a great diversity of devices. In order to cope better with these challenges, some level of abstraction is needed during the design of these applications to make their design consistent and to avoid loosing sight of the main problems that are to be solved through the vast amount of environmental and platform dependent issues. Such abstractions are already available in current model-based techniques, both in the software engineering community as well as in the human-computer interaction community. There are, however, some pieces that are missing, or that could be improved upon.

The usage of models is gaining support in both the human-computer interaction domain and the software engineering domain, moreover this increased interest in modeling is driven by industry. This dissertation will address a limited number of issues that are involved in model-based design of context-sensitive user interfaces:

Models for context-sensitive user interfaces Although much research has been performed in the design of user interfaces using models and some level of agreement has already been reached about the contents of these models, there is still work to be done in the establishment and the detailed definition of the involved models. The first goal of this dissertation can thus be formulated as: create model definitions that allow the creation of detailed, but high-level, models for the design of context-sensitive interactive applications.

Integration in software engineering For effective creation of interactive applications, it is important that the design of both the user interface and the application core is well aligned. This dissertation investigates the usage of models for design of context-sensitive user interfaces, furthermore, models are frequently used for the design of the application core. The usage of a common meta-model for both models for user interface design and the design of the application core or the integrated design of both user interface and application logic is a second goal for this dissertation.
1.2 Detailed Overview

The structure of this dissertation can be split into four major parts, which correspond to the chronological order in which the research was performed. The first part starts with the introduction, in which the goals of this dissertation are identified. Chapter 2 introduces our definition of context-sensitivity of user interfaces. This definition and general requirements for modeling languages are used to identify issues with current model-based approaches to appropriately support the design of context-sensitive user interfaces.

The identified issues are addressed step-by-step in part II of this dissertation which aims to express the models needed to design context-sensitive user interfaces in the Unified Modeling Language [OMG 04], UML, the most universally accepted meta-model within the software engineering community, in order to make them more accessible to the software engineering community. This goal also implies that extensions to UML should be made as light-weight as possible because otherwise very specialized tool-support is required which would reduce the accessibility of the notation.

Chapter 3 addresses task modeling for context-sensitive interactive applications. It analyses one of the most popular task-modeling notations, the ConcurTaskTrees notation, for its appropriateness for task modeling and then discusses the extensions that are proposed to increase the support for modeling context-sensitive interactive applications and their limitations. Subsequently, the proposed extensions to the ConcurTaskTrees notation are introduced and explained into more detail using an example model. Finally, the advantages and disadvantages of expressing the extended task model, the Contextual ConcurTaskTrees, in UML are discussed.

Chapter 4 then introduces a first step in expressing the models needed for designing context-sensitive interactive applications using UML. This includes the introduction of a UML profile ¹, CUP, that extends a limited set of models to provide better support for modeling context-sensitive user interfaces. The scope of this profile is limited to interactive applications that only have one context-of-use and is optimized for visibility of information and familiarity of the notation for user interface designers. The chapter is concluded with an analysis of the notation that identifies both the strong and weak points of this notation.

The weak points of CUP are addressed in a second version of the profile that is introduced in chapter 5. This second version of the CUP profile addresses both weaknesses in the notation and the scope of the profile. The

¹A profile is the most light-weight standardised way to extend UML.
scope is extended to include user interfaces that can be distributed over different platforms and used in multiple contexts of use. The profile definition is also more detailed to allow more rigorous definition of the models and it provides a first step towards usage of these models to generate declarative user interface descriptions.

Part II is concluded by a discussion of the models introduced in this part of the dissertation, which all use a common meta-model: the UML.

Part III uses another approach for the specification of context-sensitive user interfaces. It proposes a domain-specific language (see also section 2.6) which can lower the mental gap between the problem and the realization of an appropriate interactive system. This can be a first step towards end-user modeling of one type of context-sensitive interactive applications; staged participatory multimedia applications. Such interactive applications have a distributed user interface and are used through television sets offering limited and very specific ways of interaction to many users which are located at different locations. Context-sensitivity in this case means for the most part awareness of the other users. Chapter 7 introduces the language designed to model these applications and specifies relationships with the models introduced in the second part of the dissertation. Chapter 8 discusses how this language can be used to generate (high-level) prototypes from these models and how it is related to the models and model elements introduced in chapter 2.

The fourth and last part summarizes the contribution of this dissertation and provides a discussion of future work in chapter 9.
Chapter 2

Models for User Interface Engineering

2.1 Introduction

The ever increasing diversity of devices and users and the parallel evolution of more personalisation and more general, the adaptation to context has renewed the attention given to model-based design of user interfaces. This evolution, however, has not produced many broadly accepted models and notations. This in contrast to the world of system design where the Unified Modeling Language [OMG 06c] (UML) has emerged as the modeling notation of choice. A short discussion of UML and its history can be found in appendix D. There also does not seem to be an accepted method for integration of user interface modeling with system modeling.

Several notations for model-based design of user interfaces, and more generally interactive systems, have been proposed [Paternò 00, Mori 03, Clerckx 04a, Thévenin 01, Puerta 97, Puerta 02, Nunes 00, da Silva 03, Limbourg 04c]. They all use several models to support the design process. The names, contents and number of these models vary greatly. Despite of this, several models or variations on these models seem to come back in most approaches.

This chapter defines what the term context-sensitive user interface means within this work and establishes a set of requirements for a notation for model-based design of context-sensitive user interfaces. Some requirements cover the qualitative aspects of a possible notation. This is because the use of a notation is highly dependent, not only on the supported semantics, but also on its usability – how it represents the information in the model as noted by Paternò in [Paternò 01]. Other requirements handle the information that should be
representable for the design of context-sensitive models. These requirements are reflected in a visual representation of the required and commonly used information that is expressed in the models of different approaches for user interface design.

To get a better overview how well current model-based techniques are fit to design context-sensitive user interfaces, and more generally context-sensitive interactive applications, a comparison between the presented requirements and the coverage of models of some recent and some well known notations is presented. The results of this comparison, together with a discussion of model-usage within software engineering, are then used to determine the areas where important additional work needs to be done regarding the notation and definition of the models and where the contributions of this dissertation are situated.

2.2 Qualitative Requirements

This section discusses the qualitative requirements that were set for a model representation for the design of user interfaces. The identified qualitative requirements are linked to properties of the cognitive dimensions framework [Green 96, Green 00], which gives a vocabulary and a framework to evaluate usability of among others programming (and design) languages and the editors that are used to manipulate the models. For reasons of clarity, the terminology of the cognitive dimensions framework that is used in this dissertation is given in Appendix A.

**understandable** The notation should be relatively easy to understand and should be perceivable by as many people as possible. It is very difficult to evaluate how “understandable” a notation is. We will split this requirement in two parts: being built on a **proved and well known basis** and having a publicly available specification. A proved and well known basis increases the probability that the resulting notation has a close mapping or a good role-expressiveness, two properties that can effect usability of a notation as presented in the Cognitive Dimensions Framework [Green 96, Green 00], which is discussed in appendix A.

**divide and conquer** This requirement relates to the properties juxtaposability and visibility of the cognitive dimensions framework, which are important to modification and exploration activities. Both properties relate to how easy it is to readily find information in the models, which can be very large for real-world applications. To support these properties well,
2.3 Context-sensitiveness

different aspects of the specification should be the focus of different diagrams in graphical notations. An abstract user interface description has different aspects that are difficult to combine understandably in a single diagram. Different diagrams, using (partially) the same symbols can be easier to understand than a single model that shows everything. Different diagrams can also be used to clarify some information that is already present in another diagram. For textual notations, a clear separation of the information of the different models is also an advantage.

**hidden dependencies** The different models represent different aspects of a user interface but the information they describe is still dependent on the relations with other models; unlinked information is a potential cause of confusion or errors.

**expressive** The requirement for the notation to be understandable should not reduce the capacity of the notation to express complex models and/or relationships. Complex behavior that is not relevant to the model should not be represented as such, but as simple as possible without losing too much information. Expressiveness relates to the cognitive properties *verbosity* and *visibility*.

**tool support** The notation should be supported by a tool. A tool can provide the necessary constraints that ensure that the information presented in possibly different diagrams is consistent. A tool can also allow the designer to temporarily hide certain information in a diagram when desired. Tool support is important to overcome many problems with *viscosity* when working with models. Different models have to be kept in sync and in iterative development, which is typical for software engineering (see also section 2.6), be modified extensively.

### 2.3 Context-sensitiveness

Multiple definitions of context can be found in literature [Schilit 94, Schmidt 99, Dey 01, Coutaz 02], each having a slightly different meaning. Schilit et al. [Schilit 94] enumerate what types of information can be part of context:

- Three important aspects of context are: where you are, who you are with, and what resources are nearby (…)

Context includes lighting, noise level, network connectivity, communication costs, communication bandwidth, and even the social
situation; e.g., whether you are with your manager or with a co-worker.

Schmidt et al. [Schmidt 99] defined context as having three dimensions: self, activity and environment as can be seen in figure 2.1. In their work, however, they only target the physical environment, which is described by a set of contexts. A context is a set of cues — cues are values abstracted from sensors at a certain point in time — all taken at the same point in time. They use context information to create intelligent devices, they don’t use context to enhance interactive applications.

Figure 2.1: Three dimensional context [Schmidt 99]

Dey and Abowd [Dey 01] explicitly state that context consists of any information that characterizes the situation of entities that are relevant to the interaction between a user and an application:

**Context:** any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity and state of people, groups and computational and physical objects.

Coutaz and Rey [Coutaz 02] explicitly include information about the past in their definition of context. Their definition of context is specific for a set of users and a certain task:

\[
\text{context}^{U,T}(t) = \text{COMPOSITION}(\text{situation}^{U,T}(t_0), \ldots, \text{situation}^{U,T}(t))
\]

(. . .) The Situation, situation^{U,T}(t), is the set of the values observed (or observable) at t of the peripheral state variables that relate to U for performing T, as well as their relations.
2.3 Context-sensitiveness

In order to avoid confusion about what is covered by the terms *context* and *context of use*, their meaning in this dissertation is defined in definition 2.1 and definition 2.2.

**Definition 2.1** Context is any information that is relevant for the interaction of a subject (person or service) with the platform. This information can contain information that was gathered before as well as during the interaction about the user, the platform and the environment (including all entities in the environment).

**Definition 2.2** A context of use is a set of objects and the relations between them that describe a class of contexts in which a certain interaction can take place.

Definition 2.1, which defines context, shares the view of Coutaz and Rey [Coutaz 02] that context can also contain information that was gathered before the interaction takes place. It shares the opinion of Dey and Abowd [Dey 01] as well as Coutaz and Rey [Coutaz 02] that context should be relevant to the interaction. It differs from the definition of Schmidt et al. [Schmidt 99] in that they consider the user’s activity to be part of the context, while in our definition the context information should be relevant to the activity. Schilit’s definition of context [Schilit 94] describes some of the information about the environment, as mentioned in our definition.

It is important to note that just as these definitions of context, our definition of context differs from the definition of context as it is frequently used in software engineering, where context identifies the business process and environment in which the designed software will be integrated [Ambler 06]. A context model in e.g. RUP (see section 2.6) thus concentrates on human activities and deployed systems rather than declarative information about users, platforms and environment.

Depending on how much influence context has on an interactive application, different categories can be defined. A first category contains the context-specific applications. These interactive applications are only supported in one context of use and therefore do not need the ability to sense changes in the context of use. When information in the context of use changes, a context event (see also definition 2.3) is sent, which can be received by the interactive application. If an interactive application can receive context events and adapt its behavior or appearance as a consequence of these events, it is said to be context sensitive. We formalize this terminology in the following definitions.
Definition 2.3 A context event is an event that signals a change of context.

Definition 2.4 An interactive application is context specific if it is designed to be used within one context of use.

Definition 2.5 An interactive application is context sensitive if it can adapt to changes in its context.

A context-sensitive interactive application that can be used in multiple contexts of use and adapt its behavior accordingly, is a multi-context interactive application. An interactive application for which each context event it reacts to results in a different context of use is statically context sensitive. In some cases, it may not be appropriate to define a separate context of use for each change in context to which the application adapts. If an application reacts to context events that do not trigger a different context of use, it is said to be dynamically context sensitive.

Definition 2.6 A context-sensitive interactive application is dynamically context sensitive if it can adapt to changes in its context that do not trigger a transition to a different context of use, otherwise it is statically context sensitive.

Many context-sensitive applications, such as Dummbo and the In/Out board [Salber 99], and myCampus [Gandon 04], directly display context information and thus are dynamically context-sensitive. Although it might theoretically be possible to define separate contexts of use for each of these cases, it would involve an exponential number of contexts of use (one for each data combination).

Context-sensitive interactive applications can be further categorized according to how they adapt to changes in the context of use. Figure 2.2 shows the different types of reactions to context events. The context of use can influence the tasks that can be carried out and how they are to be performed. Tasks involving classified information, for example, may only be executed within specified contexts of use. Other tasks may depend on the occurrence of an event within the context of use. Another example is a mobile guide in a museum that triggers a prompt for more information when walking close to a museum artifact.

Adaptation of the presentation of the user interface to the context of use is a second type of context influence. For example, a different representation can be chosen to indicate temperature depending on the context of use. A
2.4 Model requirements

Besides the notational requirements, the expressiveness of the provided models is at least as important. One way to compare approaches is to find common grounds between different approaches by defining a set of models and identifying which of these models are supported by a particular approach and how these models interrelate. This approach was taken in [da Silva 00] and in [Van den Bergh 04c]. This way of working has however the drawback that it neglects the differences that exist in the contents of similarly named models in different approaches, therefore it is important to identify the different types of information expressed in these models, which is also done by da Silva [da Silva 00]. This chapter will identify types of information that could be critical to effectively support the design of context-sensitive user interfaces and discuss the level to which different approaches support these kinds of information.

Figure 2.3 shows the different types of information that were discerned for this dissertation. An informal definition for each of these types of information is given below. These definitions are based on those found in literature and the available support for building context-sensitive applications in the form purely textual representation can be used in a large form while a graphically pleasing animation can be used in an application for children.

A third and final type of context influence is a change to the data presented in the user interface. E.g. the user’s location as visualized on a map in his mobile guide is updated while walking around in a museum.
A first topic is task information. For task information, both the tasks themselves (which includes information about subtasks) and the temporal relationships between them are considered:

**Task** A task is a piece of work that has to be performed by an actor, which can be human or a computational entity, to reach a goal.

**Temporal** A temporal relation specifies how tasks or interaction objects are related to each other with respect to time. Examples of temporal relations are sequential relations (one task is executed before another one) or parallel relations (two tasks can be executed in parallel).

The user interface can be described both at an abstract level and a concrete level, at least one of both abstraction levels should be offered by a specification. A notation should be able to express the relations between different interaction objects. Support for containment is essential. For this topic the following types of information are considered:

**Abstract Interaction Object (AIO)** An abstract interaction object is the specification of the functionality of a part of the user interface independent of platform, toolkit and modality.
2.4 Model requirements

**AIO Group** An AIO Group combines abstract interaction objects or tasks into logical groups.

**Concrete Interaction Object (CIO)** Concrete Interaction objects are specifications of user interface controls independent of the toolkit or platform on which they are realized.

**CIO Group** A CIO Group contains concrete interaction objects and can correspond with, for example, a window or a tab in a graphical user interface.

**UI Relation** A UI Relation specifies a relation between interaction objects. Examples of UI Relations are containment, precedence and navigation.

**Containment** Containment is a UI Relation that specifies that one interaction object is part of another interaction object.

For the specification of context and its usage, both the context information itself as the entities responsible for gathering it should be representable. For this topic, the following kinds of information are considered:

**ContextInfo** ContextInfo identifies a specific type of information about context.

**User** A user is a human actor that interacts with the designed system.

**Platform** A platform is a combination of software and hardware on which the modeled interactive application will be deployed.

**Environment** An environment is a description of the physical environment in which a modeled interaction can take place.

**Context Collector** A context collector is a computational entity that gives information about a specific piece of context information.

**Context of Use** A context of use is a set of platforms, users and environment information that characterizes the situation in which a modeled interaction can take place.

The last topic regards information about the datatypes (and the relations between them) that can be manipulated or accessed through the user interface, or are otherwise relevant for it:
Class A class is a type of information used in the functional core of an application.

Relation A relation identifies the relationship that exists between classes.

Attribute An attribute of a class.

Operation An operation of a class.

2.5 Models in Existing approaches

This section gives an overview of which models and which relationships between those models are supported by the various approaches for model-based user interface design. The coverage of diagrammatic notations is shown by application of the stereotype «graphical» while the coverage of textual notations is shown by application of the stereotype «textual». The discussion of graphical and textual representations will be interleaved to better show differences between the textual and the graphical notations. A discussion of the qualitative properties of the notations is provided in table 2.1, 2.2 and 2.3 with extra explanation in the text when appropriate.

<table>
<thead>
<tr>
<th>Notation</th>
<th>understandable</th>
<th>base</th>
<th>specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTT graphical (G)</td>
<td>base</td>
<td>hierarchical, LOTOS</td>
<td>Public</td>
</tr>
<tr>
<td>TERESA CTT (T)</td>
<td>XML</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>TERESA AUI (T)</td>
<td>XML</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>CTT, decision node (G)</td>
<td>CTT</td>
<td>Public (G)</td>
<td></td>
</tr>
<tr>
<td>ArtStudio (G)</td>
<td>TM: tree based, DM: UML</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>MIM (T)</td>
<td>uses MIMIC</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>MIMIC (T)</td>
<td>C++ syntax</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>XIML (T)</td>
<td>XML</td>
<td>Limited</td>
<td></td>
</tr>
<tr>
<td>UMLi (G)</td>
<td>UML</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>Wisdom (G)</td>
<td>UML, UML-ified CTT</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>UsiXML (T)</td>
<td>XML, CTT (T), UAPROF vocabulary</td>
<td>Public</td>
<td></td>
</tr>
</tbody>
</table>

G: graphical notation, T: textual notation

Table 2.1: Qualitative evaluation of existing notations
<table>
<thead>
<tr>
<th>Notation</th>
<th>splitting</th>
<th>linking</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTT graphical (G)</td>
<td>only task model</td>
<td>links through tool</td>
</tr>
<tr>
<td>TERESA CTT (T)</td>
<td>all information is organised around tasks</td>
<td>limited specification of other models</td>
</tr>
<tr>
<td>TERESA AUI (T)</td>
<td>presentation and dialogue model integrated</td>
<td>no relations with other models</td>
</tr>
<tr>
<td>CTT, decision node (G)</td>
<td>only task model</td>
<td>links through tool</td>
</tr>
<tr>
<td>ArtStudio (G)</td>
<td>separate specification of models</td>
<td>in underlying description?</td>
</tr>
<tr>
<td>MIM (T)</td>
<td>separate sections for models not applicable</td>
<td>links through design model not applicable</td>
</tr>
<tr>
<td>MIMIC (T)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XIML (T)</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>UMLi (G)</td>
<td></td>
<td>TDM $\rightarrow$ DM, TDM $\rightarrow$ AM</td>
</tr>
<tr>
<td>Wisdom (G)</td>
<td>different diagrams for different models</td>
<td>links between diagrams</td>
</tr>
<tr>
<td>UsiXML (T)</td>
<td>separate specification of models</td>
<td>links through dedicated models</td>
</tr>
</tbody>
</table>

G: graphical notation, T: textual notation

Table 2.2: Splitting and linking of models in existing notations
Many of the model-based approaches use user interface models at different levels of abstraction. Definition 2.7 informally defines what an abstract user interface model, the most abstract description of a user interface except for the task model, is. The concrete user interface model has a lower level of abstraction and is specific for a certain modality such as a graphical user interface, speech interface as stated in definition 2.8. The final user interface model (definition 2.9) has the lowest level of abstraction; it can be directly rendered using a specific platform, toolkit and modality.

Definition 2.7 An abstract user interface model is a description of the structure of a user interface independent of modalities, platform and toolkit that will be used to let a interact the specified user interface.

Definition 2.8 A concrete user interface model is a description of the structure of a user interface for a specific set of modalities (e.g. a graphical user interface). It is independent of the toolkit that will be used to instantiate the specified user interface.

Definition 2.9 A final user interface model is a description of a user interface with which a user can interact that can be directly rendered.

One of the most well-known graphical notations of model-based design is the ConcurTaskTrees (CTT) notation of Paternò [Paternò 00]. The notation is a pure hierarchical task notation that also reflects the temporal relations between tasks at the same level in the same subtree, discussed in more detail in section 3.2. The related XML notation, using TERESA XML, however allows much more information to be specified (see figure 2.4). This information can include objects that are manipulated, objects that are used to interact with the application including the platforms on which they may be available. TERESA XML also supports the expression of an abstract user interface, which can be (semi-)automatically derived from the task model and associated information. The abstract user interface (AUI) description supports more or less the same reference models as the task notation, however, it is specific for a certain platform and thus does not include platform information. The TERESA tool [Mori 03] supports editing of the CTT-model and can optionally show only tasks that are relevant for a certain platform. The AUI-model is (semi-)automatically generated from CTT and minimal editing is supported. There is no domain model but there is a minimal support for the specification of classes. Many dependencies between the models are not directly visible to the designers as can be seen in table 2.2. TERESA uses a transformational
2.5 Models in Existing approaches

development methodology [Paternò 02]: starting from one or more scenarios, a task model is created. This task model is refined for one or more platforms or modalities. A task model can be transformed into presentation task sets (PTS). These PTS correspond to screens on e.g. a mobile device or windows on a desktop system. A PTS can be transformed into an abstract, a concrete or a final user interface.

Clerckx et al. [Clerckx 04a] proposed a variant of the CTT with explicit representation of the context-sensitive parts using decision nodes (for more information see section 3.3) based on the ideas expressed in [Pribeanu 01], resulting in a presentation that has limited support for context information. The graphical notation is supported by the tool DynaMo-AID [Clerckx 04a] that has richer support for models: it allows the designer to precisely define in which context of use which tasks can be carried out using a decision tree that is linked with the decision nodes in the graphical presentation. A concrete presentation model as well as a limited description of the domain objects are also supported as is shown in figure 2.5. DynaMo-AID enables to generate a prototype of the modeled application using a variety of backends using XML-based descriptions such as UIML [Abrams 04] and SEESCOA XML [Van den Bergh 04b]. During the execution of the prototype, the context described in the models can be manipulated. No details are available on the exact notation used by the DynaMo-AID tool for the textual description of the model. They propose a prototype-driven methodology [Clerckx 06b] that also starts from a task model, from which a dialog model is generated. Combined with other models (see figure 2.5), a prototype can be generated, which can then be evaluated and refined, leading to a new prototype to be evaluated, ...

ArtStudio is a tool for model-based design described by Thévenin in his doctoral dissertation [Thévenin 01]. It supports the graphical specification
of a task model (based on CTT), an abstract presentation model (notation using interaction-task notation of CTT and rectangles to symbolise groupings, interactor specification is done in a separate model) and domain model (using UML). All models are also described in an XML format (see also figure 2.6). The approach features a separate interactor model which describes the interactors that are used to build a final presentation. The abstract user interface description has a limited scope; all interactors use the same representation (an interaction task as used in CTT) and are grouped using rectangular boxes. They also propose a methodology that starts from a task model. This task model is transformed into an abstract user interface. This abstract user interface is translated into a concrete user interface, which is transformed into a final user interface. All these transformations can be done manually or automatically.

Mobi-D [Puerta 97] supports model-based design of user interfaces and uses five models to accomplish this: user, task and domain model as abstract models and dialogue and presentation model as concrete models (see figure 2.7. Mobi-D does not support automatic conversion between the models, but rather supports the user in making the conversion. All models and mappings are specified with a single textual notation, Mecano Interface Model (MIM), using

---

**Figure 2.5: Models in DynaMo-AID**

![Figure 2.5: Models in DynaMo-AID](image)
2.5 Models in Existing approaches

MIMIC [Puerta 96a]. The model coverage of MIM is shown in figure 2.7. The tools of Mobi-D do not have a graphical representation of the supported models, although suggestions for the concrete presentation model are made graphically. There are five main steps in this methodology. The first step is elicitation of user tasks, which is followed by the definition of user-task, domain, and user model components. The third step is the integration of user-task, domain, and user models into a design model component. An interactive, knowledge-based analysis of the design model is performed to derive potential presentation and dialog designs from this design model. Finally the designer builds a task-based presentation and dialog interface. All these steps are supported by tools which help the designer to make these transformations.

XIML [Puerta 02] can be considered to be the XML-based successor of MIMIC; it has similar goals and a similar set of predefined models. Both approaches do not have pre-defined models for specification of the platform, nor the environment and abstract presentation. The domain model allows specification of complex object, having attributes, but relations between the objects are not supported. The total coverage of MIMIC is similar to that of XIML since both languages are generic modeling languages whose expressiveness can partly be defined by the user of the language, since they are meta-languages. The full potential is not shown in the figure, but one can imagine full coverage. The predefined models and relations in MIM and XIML are respectively shown in figure 2.7 and 2.8. To the author’s knowledge there is no methodology associated to XIML.
Some of the approaches are based on UML. The Wisdom-approach [Nunes 00] is a model-based design methodology and accompanying notation for software design in small software companies or small groups in greater enterprises. The notation is a UML profile, meaning that it can be expressed completely by UML, but provides several extra stereotypes 1 with a corresponding notation. For the abstract user interface description, two models are used: the dialogue model expressed by a UML-ified version of the ConcurTaskTrees notation and the presentation model that visualizes the user interface structure possibly containing five different basic elements — input element and input collection, output element and output collection, and action — organised a hierarchy of containers. The models that are used in the Wisdom approach can be seen in figure 2.9(a). In contrast with the figures for TERESA XML and the graphical CTT notation, this notation uses a larger set of models, which each play their role in the development process, which is clearly defined in [Nunes 01]. All models also have a graphical representation using UML syntax, which is serializable to XMI [OMG 05]. XMI uses XML and thus all UML-based approaches are also indicated as having a textual representation. None of these

1Stereotypes are used in UML to describe additional constraints to existing UML classes for specific purposes and are the primary way to extend the UML notation.
2.5 Models in Existing approaches

approaches, however, has a dedicated, easily human readable textual representation since XMI is too verbose to be considered human-readable.

The Wisdom methodology is actually a complete software engineering approach and is depicted in figure 2.9(b). It has great similarities to the iterative and agile methodologies discussed in section 2.6 and in fact can be considered to belong to this category of methodologies. It has separate analysis and design models for the user interface and the functional core of an application. This difference does however not exist in the creation of prototypes and the final product. The development of the actual product is not separated from the development of the prototypes, resulting in an evolutionary development methodology.

Although the notation used by the Wisdom approach has its merits, it suffers from the fact that the UML-ified notation of the ConcurTaskTrees is not as readable as the original one [Paternó 01]. TaskSketch [Campos 06] is a tool that provides special support for combining the interaction model with the analysis model. CanonSketch [Campos 05], enables the design of the abstract presentations using a graphical presentation, called Canonical Abstract Prototypes, that is closer to what developers are used to when making a user interface design using GUI builders as Qt Designer [Trolltech 06]. The Canonical Abstract Prototypes notation is also used in DialogSketch [Nóbrega 05a], which links the notation with a translation of the CTT semantics to UML [Nóbrega 05b].

UMLi [da Silva 03] is an extension of UML that introduces extra concepts to UML in order to make the standard activity diagram suitable for represent-
Figure 2.9: Wisdom models and methodology
2.5 Models in Existing approaches

ing the task model (or dialogue model) and introduces a new diagram, the user interface diagram, to represent the presentation model, resulting in the model coverage in figure 2.10. UMLi supports inputters, displayers, editors and actionInvokers as basic abstract interaction components. These components can be organized in containers and ultimately belong to a free container, which corresponds to a window or a dialogue in a graphical user interface. The extended activity diagram, which was presented as a task model, has been criticised [Paternó 01] for being too low level for this purpose, but is nevertheless very comprehensible and perhaps better suited as a dialogue model. The extended activity diagram also includes explicit links with the domain and abstract presentation model. No specific development methodology that uses UMLi is proposed.

A drawback of UMLi is that it extends the UML Metamodel and thus can only be supported by a tool especially designed for the notation. ARGOi [da Silva 05] is such a tool that is publically available. This in contrast to the Wisdom method that only uses standard UML mechanisms and extends the notation using stereotypes ensuring a broad tool support. UMLi is the only model-based approach that is discussed here, offering no explicit provisions for context modeling (although e.g. user modeling has be done using

Figure 2.10: Models in UMLi
UML [Nunes 00]). It is mentioned here, because it is the only approach that gives special care to the definition of links between the task/dialogue model and the domain and abstract presentation model.

UsiXML [UCL 06, Limbourg 04c] is an XML-based specification for the description of user interfaces in the widest sense of the term. In contrast to XIML and MIMIC, it chooses to define an exact framework instead of a more generic approach. 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. The language allows the specification of domain, task, abstract user interface, concrete user interface and context models. Mappings and transformations between the different models can be explicitly defined in separate models: mapping, transformation and rule-term model. It is one of the most complete and in-depth specifications available as can be seen in figure 2.11(a). Although UsiXML is a XML-based specification it has graphical notations for some of the models. Some examples of graphical notations of these models are shown in section B.4 and the graphical notation for all models supported by the tool IdealXML are shown in figure B.12.

The method proposed in [Limbourg 04a] is very flexible and uses models at four different levels of abstraction as can be seen in figure 2.11(b). In this approach user interfaces can be adapted to different contexts of use on all levels of abstraction except on the lowest level, the final user interface model. This means that truly context-sensitive models are not targeted by this methodology.

The great amount of tools available for UsiXML as seen in table 2.3 should be put in perspective: not all tools support UsiXML completely and most are special purpose tools built especially for UsiXML at UCL (Université Catholique de Louvain) except TERESA that only supports the concrete presentation.

2.6 Software engineering and models for HCI

Introduction

This section will give a brief overview of some commonly used software engineering practices. It is not intended as an introduction to any of the discussed approaches. It is, however, a short overview to identify where HCI models would be useful and can be integrated in software engineering practices in general. The software engineering approaches were chosen because they can deal with changing requirements, which is not uncommon for novel and com-
2.6 Software engineering and models for HCI

(a) Models in UsiXML

(b) UsiXML methodology framework [Limbourg 04a]

Figure 2.11: UsiXML models and methodology
<table>
<thead>
<tr>
<th>Notation</th>
<th>expressive</th>
<th>tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTT graphical (G)</td>
<td>unclear which task for which platform</td>
<td>CTTE (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TERESA (2)</td>
</tr>
<tr>
<td>TERESA CTT (T)</td>
<td>most information in model, context is platform</td>
<td>CTTE (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TERESA (2)</td>
</tr>
<tr>
<td>TERESA AUI (T)</td>
<td>intra-dialogue events cannot be specified</td>
<td>TERESA (2)</td>
</tr>
<tr>
<td>CTT, decision node (G)</td>
<td>decision nodes mark context dependent tasks</td>
<td>DynaMo-AID</td>
</tr>
<tr>
<td>ArtStudio (G)</td>
<td>no interactor specification in abstract UI</td>
<td>ArtStudio</td>
</tr>
<tr>
<td>MIM (T)</td>
<td>complex applications realised</td>
<td>Mobile, TIMM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U-Tel model editors</td>
</tr>
<tr>
<td>MIMIC (T)</td>
<td>tested with MIM</td>
<td>see MIM</td>
</tr>
<tr>
<td>XIML (T)</td>
<td>&gt;= MIMIC</td>
<td>N/A</td>
</tr>
<tr>
<td>UMLi (G)</td>
<td>rather low-level</td>
<td>ARGOi (3)</td>
</tr>
<tr>
<td>Wisdom (G)</td>
<td></td>
<td>UML tools</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CanonSketch (4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TaskSketch (5)</td>
</tr>
<tr>
<td>UsiXML (T)</td>
<td>limited for context specification</td>
<td>GraphiXML (6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VisiXML</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FlashiXML</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel-Tk UsiXML</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RecursiXML</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TERESA (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TransformiXML</td>
</tr>
</tbody>
</table>

G: graphical notation, T: textual notation
(1) http://giove.cnuce.cnr.it/ctte.html
(2) http://giove.cnuce.cnr.it/teresa.html
(3) http://www.cs.man.ac.uk/img/uml/software.html
(4) http://dme.uma.pt/canonsketch/
(5) http://dme.uma.pt/tasksketch/

Table 2.3: Expressiveness of and tools supporting existing notations
plex interactive applications, such as context-sensitive interactive applications.

Some of the more recent software engineering processes can be categorized as either *agile development* or *iterative development*. We will shortly discuss one typical example of each respective category: Scrum, as discussed in [Larman 03], and the Rational Unified Process (RUP) [Kruchten 03]. The choice for these processes was based on the fact that they can both be used for all critical levels and for all numbers of people in the development team according to the Cockburn scale (as depicted in table 2.4).

<table>
<thead>
<tr>
<th>Life (L)</th>
<th>L6</th>
<th>L20</th>
<th>L40</th>
<th>L100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Essential money (E)</td>
<td>E6</td>
<td>E20</td>
<td>E40</td>
<td>E100</td>
</tr>
<tr>
<td>Discretionary money (D)</td>
<td>D6</td>
<td>D20</td>
<td>D40</td>
<td>D100</td>
</tr>
<tr>
<td>Comfort (C)</td>
<td>C6</td>
<td>C20</td>
<td>C40</td>
<td>C100</td>
</tr>
</tbody>
</table>

Table 2.4: Cockburn scale. Vertical scale: defects cause loss of . . . . Horizontal scale: number of people.

**Agile method: Scrum**

Agile methods were specifically designed to cope with continuously changing requirements and to concentrate on the building of software. The properties that are common to all agile methods are effectively described in the Agile Manifesto:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Each of the agile methods has some distinct characteristics. Some distinctive properties of Scrum are described as follows in [Larman 03]:

Scrum’s distinctive emphasis among the methods is its strong promotion of self-organizing teams, daily team measurement, and avoidance of following predefined steps. Some key practices include a daily stand-up meeting (the Scrum meeting) with special questions, 30-calendar-day iterations, and a demo to external stakeholders at the end of each iteration.
Scrum does not prescribe any usage of documentation or models to design or illustrate the structure and the behavior of software. Although page 116 of [Larman 03], where Scrum practices are summarized, shows that there is a high-level design phase. Rough, and perhaps also more detailed, high-level user interface models might be appropriate in some Scrum projects. The most elaborate modeling efforts may be done in the first two of four phases in a Scrum project: planning and staging in which establishing the requirements and doing analysis and high-level design still take a larger part of the total development effort. If these high-level models can be used to generate robust source code, they might also be used more extensively during the development stage.

Iterative development: RUP

The Rational Unified Process [Kruchten 03] is a framework for software development methodologies that was created by Rational, now a part of IBM. The Rational Unified Process specifies an iterative development model, in which each iteration consists of four phases: inception, elaboration, construction and transition. Although a thorough discussion of the process is out of the scope of this dissertation, it is important to note that it is considered to be important to visually model the software. The UML is recommended as a modeling language, although it is not required. All models that are necessary at a certain stage of development are defined in the specification.

The Rational Unified Process defines a set of artifacts that are used or produced during the process. Only a subset of these artifacts are models. For this discussion the most relevant models will be shortly discussed.

The first two models are related to business modeling: the Business Use-Case Model, which identifies the intended business functions, and the Business Analysis Model, which is an object model that describes the realization of business use cases. The third model, the Use Case Model is produced by a system analyst during the requirements analysis and describes all use cases for the system. Software architects and designers produce four different kinds of models: an Analysis Model, which is refined into a Design Model, a Deployment Model and a (optional) Data Model. The Design Model describes the collaboration between the classes of the system. User Interface prototypes are created by the User-Interface Designer in addition to a Navigation Map. A last model, the Implementation Model is created by the Implementer. It is worthwhile to note that software architects, Designers and User-Interface Designers all belong to the same discipline of the nine disciplines defined by The Rational Unified
Process and are thus bound to collaborate to create consistent artifacts. The usage of unambiguous user interface models can help to create consistent artifacts by all the stakeholders as will be discussed in section 6.2.2.

**Model-Driven Architecture**

The Model-Driven Architecture [Miller 03] (MDA) is proposed by the Object Management Group (OMG) as a way to develop software through models. The ultimate goal of the MDA is to promote models, not only as an instrument to document design, but also as a means to generate fully functional applications. It specifies three types of models: Computational Independent models (CIM), Platform Independent Models (PIM) and Platform Specific Models (PSM). CIM can be roughly considered to be used to specify the requirements for the software. This CIM is transformed into a Platform Independent Model or PIM, a model that specifies a system independent of the target platform. A PIM can iteratively be refined through model transformations, each transformation refines the model until the desired level of detail and rigor is reached. At this point, a Platform Specific Model or PSM is generated for each of the target platforms to which the application will be deployed. A PSM at its turn can also be refined through several transformations until it is sufficiently detailed to generate (the source code) of the program itself from the PSM. The UML is recommended by the OMG to be used as the modeling language for expressing the different models.

As discussed by Vanderdonckt [Vanderdonckt 05], the different levels of abstraction used in the Model-Driven Architecture can be mapped on user interface models on different levels of abstraction. In his approach, the abstract user interface model is a PIM and the concrete user interface model is a PSM. The platform in this case consists of the modalities that will be used to present user interface to a user. Informal definitions of these models can be found in definition 2.7 till 2.9. Just as in the MDA, the final model is code and is called a final user interface model. It is important to note that Vanderdonckt does not use the Unified Modeling Language citeUML20super (UML), defined using the Meta Object Facility [OMG 06a] (MOF), as a meta-modeling language but uses the Extensible Markup Language [consortium 01a] (XML), extended through XML Schema [W3C 03], as a meta-model.

**Domain-Specific Languages**

Another approach to use models to generate code is to use domain-specific languages. A domain-specific language consists of a set of concepts that are
familiar to domain experts. These concepts are directly tied to code that can be used to generate complete programs. A domain-specific language is thus a kind of high-level programming language; domain-specific languages however have the disadvantage that they are very focused on a certain domain meaning that these languages are hard to reuse across domains or even between companies in the same domain. Whenever the domain evolves, the domain-specific language has to evolve with it. This means that domain-specific languages are the most useful in cases where a lot of similar programs have to be developed.

The biggest advantage of domain-specific languages is the enormous increase in development speed that can be gained (up to 300 %) [Tolvanen 05b, Tolvanen 05a], which is an order of magnitude bigger than the increases in development speed reported by the use of MDA-approaches. This spectacular increase in development speed is often explained to be caused by the fact that the translation from domain concepts to software artifacts is done by hand only once. All the other times, the translation is automated. The language designer’s knowledge of how to optimize code is thus also implicitly reused by all users of the domain-specific language. This, however, also means that the success of a domain-specific language mainly depends on the expertise and knowledge of the language designers.

Since domain-specific models are used to generate complete interactive applications, they also contain the specification of the user interface. In some cases, such a language used to define interactive applications on Nokia smart phones (see [Tolvanen 05a]), the domain-specific language uses artifacts that can still be recognized in the final user interface (see definition 2.9).

2.7 Discussion

Multiple approaches towards model-based design of user interfaces are found in literature. This chapter presented requirements for a notation that supports
2.7 Discussion

model-based design of context-sensitive user interfaces. These requirements consisted of the models that should be supported and some qualitative properties of the notations for such models. We then discussed several notations keeping these requirements in mind.

Based on the discussion one can conclude that while most notations are understandable and have tool support for the models they support, much work is needed before one can claim that model-based design can greatly ease the design of context-sensitive applications:

• While there seems a great level of agreement over the basics of a notation for the task model, the ConcurTaskTrees notation is used (and extended) in several approaches [Clerckx 04a, Dittmar 04, Van den Bergh 04a]. Additional research is required to determine whether and how to integrate context information in the task notation. Should this be left for tool support and the underlying notation or made explicit in the graphical representation? Should the presentation of context information be included in the task model and how? The integration of context into the task model is discussed in chapter 3.

• Most approaches do not have a context model, although they support the definition of context information to some degree and those that cover almost all sub-models of the context model are very constrained in the amount of context they support. A more powerful context model is needed, possibly based on the work of Henricksen and Indulska [Henricksen 04b], on the Context Ontology Language [Strang 03] or on emerging context ontologies, such as in [Preuveneers 04]. Chapter 4 and 5 introduce generic support for modeling context. This approach is orthogonal to the use of ontologies. The definition of these ontologies can also be accomplished using UML class diagrams and the appropriate stereotypes, as introduced in the Ontology Definition Metamodel [IBM 06].

• Support for graphical notations is still limited. Many approaches only have graphical notations for only one or two models and the models that have graphical notations for their models have received valid critiques on some of their notations. Some more work is needed to ensure expressive and understandable graphical notations for all the needed models including the necessary links. The use of expressive and understandable graphical notations can be important for a designer to get a clear understanding and overview of a model. Expressiveness and visibility were major concerns in the design of all the proposed notations and
will be discussed in the respective chapters. In this dissertation, graphical notations for various high-level models are proposed based on the UML [OMG 04], a modeling language with a graphical syntax. The choice to focus on high-level models is partly motivated by the fact that many current software engineering processes promote to do only limited, high-level modeling or at least include high-level modeling as a basis for further, more fine grained modeling as discussed in section 2.6.

• Experience in the software engineering community has shown that for certain tasks, a domain-specific modeling language can bring greater benefits in development time than model-driven development as is promoted by the OMG through the Model-Driven Architecture or in the Rational Unified Process. This is especially true when the modeling is done for well known environments or when highly similar tasks have to be modeled.

2.8 Conclusion

This chapter started by establishing some requirements for a model-based specification of context-sensitive user interfaces. The expression of these requirements required the definition of some context-related terminology in section 2.3. Section 2.5 discussed existing model-based development approaches and the models they used. Except for the Wisdom methodology, all of these approaches only targeted the development of the user interface. Section 2.6 then discussed some software engineering methodologies and their ability to integrate user interface models. Finally, some problems regarding the creation of models for context-sensitive models were presented using existing model-based approaches were discussed including references to were they are addressed in this dissertation.
Part II

UML-based Models for Context-Sensitive User Interfaces
Chapter 3

Task modeling for Context-Sensitive Interactive Applications

3.1 Introduction

A model-based design cycle often starts by defining a user task model. The most-used user-task model notation for the definition of high-level tasks is the ConcurTaskTrees notation proposed by Paternò et al. [Paternò 98], which can be used for both cognitive analysis and design of the interaction of users with an interactive application. The notation allows to describe the tasks that have to be performed by users and the modeled system to reach predefined goals. The task model is currently supported by the ConcurTaskTrees Environment [Mori 02], which allows the specification of a great deal of meta-information about a task that can serialized to both a binary and XML format. Part of this meta-data can be used to support cross-platform user interfaces as can also be seen in the discussion in chapter 2.

The ConcurTaskTrees notation does not offer any support for the specification of context information besides which platforms will be used by users with a certain role. Note that the latter is only supported for multi-user systems, modeled by the Cooperative ConcurTaskTrees, an extended version of the ConcurTaskTrees notation that allows the specification of multi-user systems using multiple diagrams. Although multi-platform support can be generalized to multi-context support without any profound changes to the notation, as discussed in section 3.3, this does not allow a designer to fully specify the behavior of a context-sensitive interactive application.
Some applications require to specify activities that occur in the context of use and their relation to the modeled context-sensitive application. The relations that are most interesting are when the application causes an activity to occur in the context of use and when an activity that occurs in the context of use influences the activities that are performed by the user of the modeled application or by the modeled application itself.

This chapter discusses the Contextual ConcurTaskTrees, an extension of the graphical notation, which was introduced in [Van den Bergh 04a] that provides support for modeling these relations. The proposed notation provides some additional task categories that have influence on the context in which the application is executed or that are executed in the context of the modeled interaction of a user with an interactive application.

Before introducing the extended notation in section 3.4, a more detailed discussion of the ConcurTaskTrees is provided in section 3.2 followed by a short discussion of some related work. This chapter is concluded with a discussion of the notation.

### 3.2 ConcurTaskTrees

The ConcurTaskTrees notation is a hierarchical notation that allows to specify all cognitive tasks and subtasks that need to be successfully completed to reach a goal. It is a hierarchical notation that splits up a task in several subtasks, creating a tree structure in which the tasks are the nodes. The tasks at the same level in a subtree are connected with their neighbors by temporal operators. There are different operators, each having a different priority. Starting with the highest priority, the operators are choice (\(\bigcirc\), one of the siblings will be executed), order independence (\(\bigast\), the siblings can be executed in any order, but not concurrently), concurrency (\(\bigotimes\), siblings can be executed concurrently), concurrency with information exchange (\(\bigotimes\)\(\bigotimes\), siblings can be executed concurrently and exchange information), disabling (\(\lceil\), the former sibling is disabled by the latter), suspend/resume (\(\Rightarrow\), the former task is suspended during execution of the latter), enabling (\(\Rightarrow\), the latter task starts after the former ends) and enabling with information exchange (\(\bigotimes\)\(\Rightarrow\), the latter starts after the former ends and uses information produced by the former).

The notation has different task categories (see figure 3.1(a) to 3.1(d)): abstract tasks (split up in different kinds of tasks in a single-user ConcurTaskTree), user tasks (performed by the user), interaction tasks (performed by interaction of the user with the device) and application tasks (performed by
3.3 Related Work

Integration of context in the task model has also been the topic of other related research. All of these approaches allow some parts of the task model to be

<table>
<thead>
<tr>
<th>Property</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>precondition</td>
<td>a free form specification of the preconditions that should be fulfilled</td>
</tr>
<tr>
<td>type</td>
<td>narrows the scope of the task category to define the task more precisely</td>
</tr>
<tr>
<td>platform</td>
<td>the platforms for which the task is relevant</td>
</tr>
<tr>
<td>objects</td>
<td>objects that are required for a task (part of the interface or part of the application logic)</td>
</tr>
</tbody>
</table>

Table 3.1: Task properties in ConcurTaskTrees notation
Pribeanu et al. [Pribeanu 01] propose several alternatives for the specification of interactions that are partly context sensitive. They finally conclude that an integrated task model with context-dependent subtrees – subtrees that are specific for a certain context of use – would be the best way to model user interaction with application in multiple contexts. Figure 3.2(a) shows how context-dependent subtrees are integrated into a multi-context tasktree. The context-dependent subtrees share a common parent task and have choice operators specified between them.

Clerckx et al. [Clerckx 04b] created a separate notation for this common parent task, the decision node. This node specifies the contexts of use in which the subtrees should be executed. This allowed them to create a user interface that adapts the available interactors to the active context at runtime by linking the tasks to concrete interaction objects. Figure 3.2(b) shows the same excerpt of a tasktree as figure 3.2(a) using the notation proposed by Clerckx et al. Figure 3.2(a) has the advantage that the appearance of the ConcurTaskTrees notation is kept as it was, while the notation by Clerckx et al. has the advantage of providing explicit visualization of where context-dependent subtrees start.

Rather than providing specific extensions to allow context-dependent subtrees, UsiXML [Limbourg 04d] generalizes the approach that is taken by Mori et al. in developing the ConcurTaskTrees Environment [Mori 02] and Berti et al in TERESA [Berti 04]. Both these tools allow to view partial models, showing only tasks that are relevant for a certain platform. Instead of specifying the possible target platforms as is done in CTTE and TERESA, using an unmodified ConcurTaskTrees notation, UsiXML allows the specification of the
possible contexts of use for which a task is relevant. The graphical notations provided by the tools that allow creation or modification of the task model expressed in UsiXML also do not provide any hints of which tasks are active in which context of use.

These notations [Priebeanu 01, Clerckx 04b, Limbourg 04d] are able to express task models for statically context-sensitive interactive applications, they however are unable to express dynamically context-sensitive interactive applications since they rely on changing contexts of use. Dynamically context-sensitive interactive applications, as defined in definition 2.6 also need to react to changes within a context of use.

3.4 Contextual ConcurTaskTrees

Contextual ConcurTaskTrees extends the ConcurTaskTrees notation by adding four new task categories, which can be grouped under the term *contextual tasks* defined in definition 3.1.

**Definition 3.1** Contextual tasks are tasks of which the corresponding activity triggers a context event.

Three of the four new task categories are contextual versions of categories that are part of the ConcurTaskTrees notation: application task, interaction task and user task. The fourth task category is entirely new and is a task that is executed by an entity that is not the application nor the user. The icons for the contextual tasks all contain a bold C. For the task categories that are based upon the task types available in the ConcurTaskTrees notation, the C is placed on top of the symbol of the corresponding task category. The icons of the new task categories are shown in figure 3.3.

![Figure 3.3: Additional task categories in Contextual ConcurTaskTrees](image)

(a) Contextual application task  (b) Contextual interaction task  (c) Contextual user task  (d) Environment task
3.5 Example

This section provides a short example that demonstrates the usage of the contextual tasks. The example models a museum application that has two operational modes. The first is an exploratory mode, that does not direct the museum visitor to follow a certain tour, but instead attracts the visitor’s attention to some artifacts that might be of interest when she/he is close to such artifact. The second is more traditional guided tour that guides a visitor through the museum following a certain route. In all cases a visitor is notified when another visitor is also viewing information about an artifact, thus stimulating the interaction between visitors and avoiding that one visitor blocks the view on an actual museum artifact while she/he is looking at the mobile guide.

Figure 3.4 shows a partial Contextual ConcurTaskTree of the mobile guide. The remainder of this discussion will focus on the usage of the contextual tasks. The first contextual task that is used, a contextual interaction task, is the first task in the model. The task Enter name is a contextual task because in the background, some context information about earlier visits of the current visitor is fetched. This information is then used in the task Show Tour options to preselect the option that the visitor would likely choose. This also explains the fact that the operator enabling with information exchange is used between these tasks.

The following contextual task that is used is a contextual user task, Go to artifact. This task is executed in parallel with the task Update position. The information exchange between these parallel tasks occurs because the position of the visitor is tracked within the museum by some sensors, which are not part of the mobile guide application. The mobile guide, however, is able to use this information to update the user’s position on a map embedded in the application’s user interface.

The task Show artifact information is a contextual application task, because when the information is shown, the fact that this information is shown is logged in the background by some other service. When the visitor later opts for a more free form tour, based on artifact alerts, repeated notifications about a single artifact in the neighborhood can be avoided.

The example also contains an environment task, Detect other visitor, which is executed by the museum tracking system, an entity indirectly involved in the interaction of the user with the mobile guide and thus part of its context of use. This event triggered by another system, alerts the mobile guide that another visitor is nearby. The mobile guide then offers this information to
Figure 3.4: Contextual ConcurTaskTree model for a museum visit
the visitor with a friendly message, possibly leading to human-human contact between visitors.

3.6 Contextual ConcurTaskTrees and UML

The relationship between the ConcurTaskTrees notation and the Unified Modeling Language (UML [OMG 04]) has been discussed by Paternò [Paternò 01], creator of the ConcurTaskTrees notation. The ConcurTaskTrees notation clearly has some relationships with the domain (or application) model, as was discussed in sections 2.5 and 3.2. To resolve some issues, use cases were integrated into the ConcurTaskTrees Editor [Mori 02]. This, however, means that the consistency between the domain elements in the task model and the same domain elements in the domain model, which can be specified very well using the UML class diagram [Paternò 01] have to be maintained manually. This situation only becomes worse when context is involved.

We thus propose to integrate the ConcurTaskTrees notation with UML by extending the UML metamodel which would allow to store all specifications in one repository using only a single meta-model. Two different approaches can be taken, both of which have been done before: defining a profile with the necessary stereotypes [Nunes 00] (an example of this notation can be seen in figure B.4) or extending the metamodel itself [Nóbrega 05b]. Both approaches, however define new notations that deviate significantly from the original ConcurTaskTrees notation. This can result in usability problems as discussed in [Paternò 01]. It is therefore wise to stay as close as possible to the original specification.

If one considers the Contextual ConcurTaskTrees to be a behavioral specification [Paternò 01, Van den Bergh 05], one could argue that accomplishing this with strict adherence to the semantics of the UML meta-model would probably require an extension or even significant changes of the UML metamodel. Although a task could be mapped to an Action and tasks that contain other tasks could be mapped to StructuredActivityGroup ¹, the notation that is proposed within the superstructure differs significantly; while the UML proposes visual nesting, the ConcurTaskTrees notation proposes a tree-based notation. Finding an appropriate mapping for the temporal relationships can be even more challenging [Nóbrega 05b].

¹Both Action and StructuredActivityGroup are UML [OMG 04] meta-classes. Just as for all other UML elements used in this dissertation, their graphical representations can be found in appendix D
One can, however, consider the ConcurTaskTrees notation to be a structural specification identifying the relations between classes, which happen to have behavioral aspects. Using this proposition, it is possible to express the ConcurTaskTrees notation using the light-weight UML extension mechanism, namely profiles. Tasks, in this case, naturally map to classes and temporal relationships can be expressed as a special kind of associations. The relationship between a task and its subtasks is represented using a stereotyped association. This means that one has to define one stereotype for each task category and one stereotype for each kind of relationship between the tasks.

Figure 3.5 shows a profile that defines all necessary stereotypes for the Contextual ConcurTaskTrees. In addition to the stereotypes required to express all task categories and all kinds of relationships between tasks, two abstract stereotypes have been defined to reduce the amount of duplication in tagged values. All task categories have the same properties as in the original ConcurTaskTrees notation, making the introduction of an additional abstract class that contains the corresponding tagged values a reasonable design decision. All new task categories contain only one additional property: the entity that is responsible for handling the appropriate environment action.

Figure 3.5: UML profile for Contextual ConcurTaskTrees
The types of the properties of the tasks are slightly changed, compared to those listed in table 3.1. The original ConcurTaskTrees allows the specification of target platforms, while the Contextual ConcurTaskTrees allows specification of contexts of use. This means that the tagged value contextOfUse could be replaced by the tagged value targetPlatform of type Node. Roles are now also expressed as a Class instead of a name, since in this work user roles are considered to be more than a property (with type string) of a task as discussed in chapter 2. When this profile is used in combination with the profile defined in chapter 5, additional constraints can be defined as will be discussed in chapter 6.

Note that the proposed profile does not exactly mimic the look of the original ConcurTaskTrees. The resulting notation is very similar to that of the original Contextual ConcurTaskTrees as can be seen in figure 3.6 which is made in a UML editor (MagicDraw) if one compares it with figure 3.4 which is made using a specialized editor (CTTE). The decorations that are related to certain task properties are shown as tagged values (between curly braces) instead of their original notation: the name of the task between brackets (when a task is optional), a star (for a repeatable task). Tasks in a cooperative ConcurTaskTrees diagram that are also part of a role specification do not have an ornament anymore in the UML-ified version.

3.7 Discussion

The example discussed in the previous section shows that all contextual tasks share the same pattern: there is an activity of the user (with or without interaction with the application) or the application which starts an activity in the context of use. This means that a contextual task actually specifies a double task, one carried out by one of the stakeholders in the interaction and one by another entity in the context of use. Strictly speaking, this means that all contextual tasks could be replaced by their non contextual versions in combination with an environment task.

The possibility to use contextual tasks that offer an integrated specification of both the activity by the user or application and the activity in the context of use is however kept as an additional task category because the activity is cognitively perceived as a single activity. In the example, which activities are required to capture the movement of the user in a museum is not so important to warrant a separate task in the cognitive model, but is important to identify that some activity is required and to indicate this in the extra annotations that can be made for each task.
Figure 3.6: The museum visit example of figure 3.4 using the Contextual ConcurTaskTrees in UML
Another alternative is to not introduce special notations for the contextual tasks and only adding an extra property to the existing task categories. Appropriate reaction to context influences is crucial in many context-sensitive applications and thus making tasks that influence the context of use immediately visible can be considered to be a good decision.

The introduction of the contextual tasks enables the specification of at least two of the three types of context influences on user interfaces. Influence on the task model can be modeled by using temporal operators such as enabling, disabling or suspend/resume between a contextual task and its right sibling. Influences on data can be modeled by using temporal operators with information passing between a contextual task and its right sibling.

Although the contextual tasks can be used to support platform-dependent or context-dependent tasks, as discussed in [Van den Bergh 04a], this situation is not optimal and has an artificial feel to it. One of the approaches discussed in related work can be used to support multi-context support. In [Clerckx 06a] the environment task is integrated as context task into the work presented in [Clerckx 04b]. One of the reactions to [Clerckx 06a] was why the additional context task was specified in an example illustrating the usage of the context task, providing some additional motivation that the initial idea to provide the option to specify integrated contextual tasks.

The UML-profile for the Contextual ConcurTaskTrees notation introduced in section 3.6 has the advantage over previous integration methods that it stays closer to the original look of the ConcurTaskTrees notation than other approaches discussed in literature [Nunes 00, Nóbrega 05b]. The integration of the ConcurTaskTrees notation in UML has the benefit that it is possible to express relations with other models directly taking advantage of all the features of the modeling environments and transformation technologies defined for UML. Furthermore, since there is a standardized exchange format for UML, the XML Metadata Interchange format or XMI [OMG 05], a specialized environment for task modeling could export the model to XMI and as such ensure that the results can be used in many widespread UML tools.

The ability to express the (Contextual) ConcurTaskTrees notation in UML does not reduce the need for specialized editors since directly using the UML-profile for creating Contextual ConcurTaskTrees models in a UML-editor has the disadvantage that it might prove to be impossible to completely hide many features of the editor and UML itself that are irrelevant for the designer of the task model. Further discussion of the usage of UML for editing Contextual ConcurTaskTrees can be found in section 6.1.
3.8 Conclusion

This chapter presented a new notation for designing context-sensitive applications using task models, which extends the ConcurTaskTrees notation by integrating a new type of nodes, contextual tasks. These tasks allow to specify the relation between activities that occur in the context of use and activities that are performed by the user or the modeled application with or without interaction. They do also make it possible to not only specify the tasks for different contexts in one task model, but also tasks that change the context and tasks that react on context changes. The introduced extensions allow the specification of dynamically context-sensitive interactive applications at the task level, which was not possible with previous notations. Furthermore, a UML profile is defined for expression of the Contextual ConcurTaskTrees, which opens the door to effective integration of the (Contextual) ConcurTaskTrees notation into a UML-based software engineering environment.
Task modeling for Context-Sensitive Interactive Applications
4.1 Introduction

This chapter proposes a UML-based notation that allows the specification of how context should be integrated into an interactive application. Special attention has been paid to address issues that were identified in UML [da Silva 03] and the Wisdom notation [Nunes 00], two approaches that used and extended the UML 1.x notation to express models used in model-based design of user interfaces. The choice to define a UML profile was inspired by the ubiquitous tool support available for UML, being a (de facto) standard modeling language, and the usability that can be obtained by adhering to the relevant guidelines that were used in the design of UML. In the design of the profile, the usage of diagrams in model-driven design was investigated and kept as close as possible to that usage when defining which diagrams to use and how to use them.

The rest of this chapter is structured as follows: it starts with a short discussion of the work that has been done in two areas that are strongly related to the discussed notation. The first is further integration of context into models for the design of user interfaces, already started in the previous chapter for the task model, and the second is the description of these models using UML 2.0.

In section 4.3 the goals for the profile as well as the stereotypes it groups are introduced. The following section discusses how the profile aids the specification of the different models for the design of user interfaces using an example. The chapter ends by a preliminary review of the usability of the proposed
4.2 Related Work

Support for the development of context-sensitive user interfaces is mostly established in the form of programming support. This section will shortly discuss some typical examples of such programming support and then continue by discussing the design support for context and context-sensitive user interfaces. We will conclude this section with a discussion of approaches that use UML to express models in the area of model-based design of user interfaces.

Winograd [Winograd 01] discerned three different ways to programmatically integrate context: using a widget-based framework, a services-based approach which is more flexible and a blackboard-approach in which all data is centralized for later retrieval. The Context Toolkit [Dey 01] uses the first approach. It provides a toolkit that consists of independent components: context widgets — they provide abstractions for the gathering of context similar to the abstraction widgets in graphical user interfaces provide for the low-level graphics calls and event-handling — context aggregators and context interpreters. An example of such a widget is the Location widget, as described in Dey’s doctoral dissertation [Dey 00]. This widget allows an application to be aware of the presence of users and its implementation and the application that uses it are independent of the sensor that is used to get the presence information. In the dissertation six different sensors, such as the iButtons [Maxim/Dallas 06], a graphical user interface and an infrared motion detector are listed.

In ConFab [Hong 04] a different approach is taken. It considers the user as a central point about whom information is gathered. All information about a specific user is collected into an “infospace” which can be queried using different services which allow or deny the information to be gathered based on privacy settings. A query can be about the user’s location as is shown in figure 4.1.

Henricksen and Jadwiga [Henricksen 04b, Henricksen 06] also discuss a framework for integration of context into context-sensitive applications. They propose the use of context widgets at the lowest level and aggregation of the information in a central repository which can be queried by applications. They also propose a graphical notation for modeling context. An example (including a legend) of this notation is shown in figure 4.2.

Some guidance in the development of context-sensitive interactive systems can be found in the Cameleon RT reference framework [Balme 04] and the reference framework for plastic interactive applications [Calvary 04]. The first
4.2 Related Work

Figure 4.1: Example of a location query using the ConFab infrastructure in the Lemming instant messenger [Hong 04]

defines an reference architecture for distributed, migratable and plastic (adaptable to context and still usable) user interfaces. The architecture defines three layers. The middle layer provides context sensing and adaptation. The bottom layer consists of physical hardware as well as operating system. The interactive applications and a meta-user interface are contained in the top layer. This meta-user interface provides metadata about the user interface and allows a user to control the behavior of the middle layer.

The reference framework for plastic interactive applications on the other hand provides a view on the models that can be used to develop plastic user interfaces and the different design constructs in which they can be used: user interface descriptions at four levels of abstraction. The most abstract level is tasks and concepts model. The abstract user interface model describes the user interface independent of modalities and toolkits. The concrete user interface is specific for a chosen modality (e.g. a graphical user interface or a speech interface) but independent of the toolkit. The final user interface is a user interface for a specific toolkit.

The Wisdom approach [Nunes 00] provides a UML-profile for the design of interactive applications in small teams. It has a UML-ified version of the ConcurTaskTrees notation [Paternò 00], a frequently used task model within the model-based design, as well as a special notation for the presentation model. Both representations are based on the class diagram. This means that behavior (in the UML-ified version of the ConcurTaskTrees) is specified using a diagram designed for description of structure. Paternò [Paternò 01] mentioned that enforcement of correctness for the task model does not mean it is a usable notation. The presentation model has no relation with the
way user interfaces are constructed in contemporary tools which places limits on its adoption. This was also acknowledged and therefore a separate tool was developed, CanonSketch [Campos 05], allowing interaction closer to what designers are used to. An overview of the models used in the Wisdom approach can be seen in figure 2.9(a) and a more in-depth discussion is provided in section 2.5. Section B.2 shows some example models.

Another notation based on UML is UMLi [da Silva 03], which extends the UML metamodel, breaking compatibility with most UML tools. The notation defines two new diagrams. The first is the user interface diagram, which allows to define the composition of a user interface in a way similar to what designers are used to; by graphically nesting containers and interactors. The second di-

Figure 4.2: Example of a context model for a context-aware communication program [Henricksen 06]
agram is an extended version of the activity diagram, which introduces special “components” that allow specification of optional and repetitive actions. Figure 2.10 gives an overview of the models used in UMLi and a more in-depth discussion is provided in section 2.5. Section B.3 shows some example models.

4.3 CUP: Overview

Before discussing the defined profile, it is important to say why I didn’t extend an existing notation (as discussed in section 4.2) to allow the specification of context influences and what our exact goals were for designing the new UML profile.

4.3.1 Goals in designing the profile

The analysis in chapter 2 of the state-of-the-art in model-based design of user interfaces showed that there was a lack of generic context support and that graphical notations used for the various models used in these approaches were lacking or did not support the purpose very well [Paternò 01]. Before starting the design of a new notation, a set of goals were defined for it.

The first is that it should support the modeling of context-sensitive interactive applications. Therefore an analysis was performed of what types of context integration are possible and which ones are covered by the term “context-sensitive”. This resulted in the definitions presented in section 2.3.

The second goal is to support the early design stages but take into account that results can be used in later phases of the design. Therefore all models identified to be useful in this design stage should be supported and be flexible enough to have some usage in later design phases.

The third goal is that the notation should be usable in its proposed usage environment. This means that it should be relatively easy to use for both user interface designers and by software engineers since they need to collaborate in order to create an interactive application (see also section 2.6). We therefore opted to build on an already established language for software engineering: the Unified Modeling Language (UML). Extensions of the existing language should be minimal and they should fit as much as possible into the current usage of the language; behavior is described by behavior diagrams and structure is described by structure diagrams.

When designing a new notation it is important to keep the usage of the notation in mind. Green [Green 00] gives a categorisation of different activities performed during the design and implementation of a system. The
envisioned usage for CUP falls into several categories, although the main one is exploratory design. This is the most demanding activity in the list. The notation should ideally have low viscosity (low resistance to change), reduced premature commitment (little constraints on the order of doing things), high visibility (high ability to view components easily) and high role-expressiveness (understanding what elements do). A last point is that abstractions are useful to lower viscosity, but require an extra learning effort. We will revisit these points of attention in the discussion in section 4.5.

4.3.2 The profile

The proposed profile extends the UML metamodel in three different places. The extensions are shown in figure 4.3. The profile specification is split in three parts, corresponding to their use as will be explained in the following section. Figure 4.3(a) specifies the extensions made to the deployment diagram, which are used to describe the static structure of the user interface. The dynamic structure of the user interface is specified using an extended version of the activity diagram (figure 4.3(c) 1) while the class diagram is extended to describe the context (figure 4.3(b)). The semantics of the defined stereotypes will be defined in section 4.4.

4.4 Models and diagrams

An overview of the canonical models that are used to design the user interfaces can be seen in figure 4.4. These canonical models were introduced by da Silva [da Silva 02] except for the context model, which was added as part of the research discussed in this dissertation to be able to model context-sensitive user interfaces. The figure gives an overview of the models and the relations between them using the UML syntax. Dependencies between the models represented in the figure are:

- The Application Model shows the concepts and the relations between them that are used within the application; especially those concepts that the user interface interacts with.

- The Task Dialog Model, or activity model, is in fact the composition of two commonly used models in model-based design. The task model is

---

1The profile as introduced in [Van den Bergh 05] contained less stereotypes. The additional stereotypes mentioned here are added to completely follow the UML standard.
mostly used during the analysis stage and provides an hierarchical view to the activities that need to be accomplished using the modeled user interface. Tasks are split into subtasks and sibling tasks are connected using temporal operators. The dialog model on the other hand provides another view on the tasks that can be performed through the modeled
user interface. The tasks that are active at the same moment in time are grouped. The possible sequences of tasks are the main focus of the dialog model.

- The Context Model shows the concepts that can influence the interaction of the user with the application directly or indirectly. The relations between the concepts are also important as well as the way the information can be gathered.

- The Abstract Presentation Model shows the composition of interactors in the user interface and describes the general properties of the interactors; the data they interact with and meta information about them.

- The Concrete Presentation Model describes the user interface for a specific set of contexts or platforms.

Figure 4.4: Canonical models used for context sensitive user interfaces

These canonical models was chosen because they, except for the concept model, were explicitly introduced to compare different model-based approaches to user interface design in [da Silva 02]. Furthermore, these names or small variations thereof are also commonly used to identify the models in different approaches as can be seen from the figures in section 2.5. Another option would have been to use the reference framework [Calvary 02] defined as part of the Cameleon project [ISTI CNR 04]. It defines a set of six conceptual models (concepts, tasks, platform, environment, interactors and evolution) which are used to define artifacts at four levels of abstraction: tasks and concepts model,
abstract user interface, concrete user interface and final user interface. The six models that are defined in this framework are however highly based on the approach proposed by Thévenin [Thévenin 01] and uses two models, the interactor model and the evolution model, that are only found in his approach. Furthermore, models that are used in many approaches, such as the abstract or concrete presentation model are not present as conceptual model in this reference framework.

All canonical models, except the concrete presentation model, are used in the proposed approach. CUP defines stereotypes for three of these models, namely the activity model, the context model and the abstract presentation model. The fourth, the application model, is visualized using the UML class and package diagrams. There are no extensions defined for the application model. It is however referenced in the other models. We will discuss the notations for the other models in more detail in the following sections, stating the requirements set for the model, how the model was realized and a small example illustrating the notation.

4.4.1 Modeling the presentation

Currently the presentation of a user interface is mostly designed with tools for final user interfaces. GrafiXML [UCL 04] uses this more concrete presentation too.

We want to support specification of the user interface structure and allow indication of the type of interaction the user interface components support. The abstract presentation should also be able to give meta information about the user interface components such as the type of data they interact with as well as more descriptive information (such as explanations about the functionality of the specific user interface component).

Similar to current design tools for concrete user interfaces, when a user interface component A is contained in a user interface component B, A should be visually contained in B.

We realized our goals by extending the deployment model with a set of stereotypes as shown in figure 4.3(a). Four types of user interface components are defined using stereotypes:

«inputComponent» An input component is a user interface component that allows the user to input data. An initial value can be provided, but is not required. An alternative presentation is provided: an arrow entering a square indicating data flowing into the system. An input component can correspond to a wide variety of controls in a final user interface.
An edit box, a set of check boxes, a drop-down list, a date picker and a drawing canvas (including the corresponding label and help information) are all examples of controls in a graphical user interface that are input components in the abstract presentation model.

**<outputComponent>** An output component is a user interface component that is used to present data to the user, it does not allow data input. An alternative presentation is provided: an arrow leaving a square indicating data flowing out of the system. In a graphical user interface an output component can for example be a read-only textfield or a control that can show movies.

**<actionComponent>** An action component is a user interface component that lets the system execute an action when it is activated. This action is preferably not limited to a change in the user interface (e.g. a next button in a wizard should not be modeled; it only provides a way to navigate through a concrete user interface and has no influence on the state of the application). An alternative presentation is provided: a right-pointing arrow, analogous with the play symbol, within a square symbolizing the system. In a graphical user interface an action component can be a button or a menu item.

**<groupComponent>** A group component is a user interface component that
groups other user interface components into a logical structure. No alternative notation is provided. A group component can be instantiated as e.g. a window or a panel in a graphical user interface or can even have no corresponding control.

The user interface components are extensions of the UML 2.0 Node. This allows visual nesting of user interface components as can be seen in figure 4.5. Meta information can be specified using attributes of the stereotyped Node. The datatype or class manipulated by a user interface component is specified by a stereotyped Association. The type of relation is indicated through the stereotypes «select», «interact» or «trigger». Table 4.1 specifies to which associations these stereotypes can be applied.

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Source</th>
<th>Target</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>«select»</td>
<td>input component</td>
<td>Class (collection or enumerated type)</td>
<td>input component allows selection of a value from the datatype</td>
</tr>
<tr>
<td>«interact»</td>
<td>input component, output component</td>
<td>Class</td>
<td>UI component displays or allows modification of datatype</td>
</tr>
<tr>
<td>«trigger»</td>
<td>action component</td>
<td>Class (with operation)</td>
<td>activation of action component triggers the execution of the specified operation</td>
</tr>
<tr>
<td>«precede»</td>
<td>user interface component</td>
<td>user interface component</td>
<td>Source precedes target in time and/or space</td>
</tr>
</tbody>
</table>

Table 4.1: Stereotypes for associations between user interface components and datatypes

Figure 4.5 shows the specification of a group component MakeAppointment that allows creation of an Appointment. It consists of a group component that allows specification of a name and the appropriate time information, an input component that lets the user select contacts involved in the appointment and two action components, one to confirm the creation of an appointment and one to cancel the creation of an appointment.
Two «precede» relationships specify that \textit{Start Time} should be presented before \textit{End Time} and \textit{Start Date} before \textit{End Date}. The two final user interfaces in figure 4.6 satisfy this constraint according to the conventions in Europe where there is a top-to-bottom reading order.

The «select» association shows that the input component \textit{Contact Selection} allows the selection of contacts from the \textit{ContactList}. The tagged value \textit{closed} (of type Boolean) can be used to denote whether a selection is \textit{open} or \textit{closed}. The two remaining user interface components in the example are action components that provide confirmation or cancellation of the operation.

All input components are associated to the class \textit{Appointment} through associations with the stereotype «interact». This means that they can all change the value of an attribute of the class Appointment. The tagged value \textit{relatedProperty} of the stereotype «interact» can be used to specify which attribute is modified by which input component. As mentioned in table 4.1, this stereotype (and the corresponding tagged value) can also be used for output components.

Action components can have an association with a class with the stereotype «trigger», in which case its activation would trigger the execution of one of the methods of the class. The tagged value \textit{relatedOperation} can be used to specify which method is activated.
The choice to represent abstract user interface components using stereotyped Nodes creates the possibility to create different instances from this abstract representation in which Artifacts can be used to determine the concrete components that will be used. This enables the designer to map one user interface component on multiple concrete instances. An input component for specifying a country could for example be mapped to both a dropdown list and an image map for an XHTML interface for desktop usage, and a textfield for a user interface on a mobile phone. For an instance of the abstract presentation that represents a wizard, Artifacts representing previous and next buttons, having no corresponding abstract user interface components could be added.

An example of different user interfaces based on the same abstract description in figure 4.5 can be seen in figure 4.6. Figure 4.6(a) shows a wizard interface that has additional buttons and combined labels for date and time entry, while figure 4.6(b) shows an HTML interface in a textual browser that has a label for each component, but no extra user interface elements are defined.

The mapping from the abstract user interface components in figure 4.5 to controls in the graphical user interface in figure 4.6(a) is visualized in figure 4.7.

As mentioned before, our representation has some similarities with the notation used in UMLi, nevertheless it has however some major differences. The first is that the UML-metamodel is extended using a profile rather than using a custom-built extension. The second difference is that while UMLi has two types of containers, one for specifying a dialog and one for specifying containers within a dialog, our approach only has one type of container (the group component). This difference is caused by the higher level of abstraction and thus the top-level container in a diagram can be contained in another container and does not need to present all contained user interface components at one moment in time to the user. The contained user interface components might for example be represented sequentially in a wizard or a speech interface.

The Canonical Abstract Prototypes notation (CAP) also has similar abstractions, which are discussed into more detail appendix C. An informal overview of the relationship between the stereotypes in the abstract presentation model is shown in table 4.2. This overview is not complete since the semantics of CAP is not formally defined and depends on its users [Constantine 03a] although the most important relationships should be present and be accurate.
Figure 4.7: Mapping of the model in figure 4.5 to the graphical user interface in figure 4.6(a)
4.4 Models and diagrams

<table>
<thead>
<tr>
<th>abstract component (CUP)</th>
<th>association</th>
<th>CAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>≪groupComponent≫</td>
<td>/</td>
<td>container, generic abstract material</td>
</tr>
<tr>
<td>≪outputComponent≫</td>
<td>≪interact≫</td>
<td>element, notification</td>
</tr>
<tr>
<td>≪inputComponent≫</td>
<td>≪interact≫</td>
<td>interactive material</td>
</tr>
<tr>
<td>≪inputComponent≫</td>
<td>≪interact≫</td>
<td>selectable collection</td>
</tr>
<tr>
<td>≪select≫</td>
<td></td>
<td></td>
</tr>
<tr>
<td>≪actionComponent≫</td>
<td>≪trigger≫</td>
<td>generic tool (and derivatives)</td>
</tr>
</tbody>
</table>

Table 4.2: Relation between CUP and CAP

4.4.2 Modeling context

Different approaches for modeling context do already exist. Some approaches use ontologies; CoOL [Strang 03] is an ontology language based on the Web Ontology Language and the Aspect-Scale-Context framework that is also instantiated using other languages for describing context-sensitive services. Chen et al [Chen 03] defined a specific and detailed context ontology about a university campus. Preuveneers et al. [Preuveneers 04] define a general “reference” ontology detailing what can be considered context, consciously avoiding much detail, which can be provided by other ontologies.

Henricksen and Jadwiga [Henricksen 06] define a graphical context modeling language (see figure 4.2) and focus on the properties of context data. We will now also discuss different properties of context based on an extensive literature study and gathered experience. We can organize these properties in three categories:

- **collection** context information can be manually entered into the system by the user or the designer of the software (profiled) or it can be automatically sensed or interpreted (detected) using sensors, services or other programs in the environment.

- **ambiguity** Context information is gathered from different (imperfect) sources at different moments in time and can thus be ambiguous [Henricksen 04a, Strang 03, Dey 02].

- **topic** Because context information has to be relevant to the interaction of a user with a system, the context and the application model can overlap.

The last observation about context information led to the choice to use the class (and package) diagrams to describe context information since the application model is already described using these diagrams in UML. We considered the way information is gathered to be important metadata and decided
that it can be described best using two stereotypes: `profiledContext` and `detectedContext`.

Quality information can be specified best using attributes (e.g., a sensor that detects temperature reports results in degrees Celsius until a tenth of a degree, with an accuracy of one fifth of a degree), while ambiguity can be observed without extra notational requirements when the same information is gathered from different sources indicated by the multiplicity of an association or by multiple associations. The associations between different types of information can also identify whether the information is about the user, the system, the services or the environment.

Besides the information itself, context collectors — the entities that gather or translate context information and can deliver it to the application — are described. All three types of context management techniques described by Winograd [Winograd 01] could be modeled as context collectors: context widgets, networked services or a blackboard approach. The context widgets, however, map the best onto the context collectors (`contextCollector`).

A context collector has an alternative representation. It consists of a black dot (representing the context), which is connected to the center of a square (the system) by an arrow as can be seen in figure 4.8. It shows that context information types can be related to a context collector. In this case the `contextCollector` AvailabilityChecker uses the user’s agendas to derive the available user list.

![Figure 4.8: Context specification](image)

### 4.4.3 Modeling activities

Activities performed by a user are traditionally modeled using a task model, such as GOMS, HTA, or ConcurTaskTrees (CTT) [Mori 02]. The latter is a notation that is frequently used in high-level model-based design. It uses a hierarchical decomposition of tasks (corresponding to activities) that need to
be performed to reach a certain goal. The tasks are linked using temporal operators (symbols for these operators are derived from LOTOS). CTT allows specification of four different types of tasks: user tasks (tasks performed by a user), interaction tasks (interaction from the user with the system), system tasks (entirely performed by the system) and abstract tasks (tasks have subtasks of at least two types, such as interaction and system tasks).

The goal is to have a notation that has the expressive power similar to that of the Contextual ConcurTaskTrees (see chapter 3) notation within UML, also in later stages, integration of objects that should be manipulated and links with the presentation model should be easy to specify.

The UML 2.0 activity diagrams are used, despite remarks of Paternò [Paternò 01] in 2001 that UML activity diagrams “tend to provide lower-level descriptions than those in task models” and that they “require rather complicated expressions to represent task models describing flexible behaviors”. UML 2.0 activity diagrams, however, are more expressive than their counterparts in earlier versions of UML, which were state-of-the-art in 2001. Although low-level details can be expressed using the activity model, they are also used to describe high-level design and during analysis. C3 [Foltz 01] is a slightly modified notation of the UML 1.x activity diagram to model tasks and information for cooperative work and has been successfully applied in early stages of a chemical design process and in product development in the automotive industry.

The use of the UML 2.0 activity model delivers the wanted expressive power; it has an hierarchical structure and temporal relations such as disabling (using interruptable regions), sequencing, choice and parallel operations can be specified. We opted to define stereotypes for each type of action:

**user** represents an action entirely carried out by the user (this can be both a cognitive and a physical action)

**application** represents an action entirely carried out by the application

**interaction** represents an action in which the user interacts with the appli-
environment represents an action performed by an entity in the physical environment, that is nor the user nor the application.

The usage of stereotypes also has the advantage that one can define alternative representations for these action types. Examples are shown in figure 4.9. We opted not to define stereotypes for activities, since only actions are actually carried out.

Figure 4.10 shows an activity diagram that specifies how to create an appointment (corresponding to the presentation diagram in figure 4.5). The black arrows in the figure indicate the control flow, while the lighter arrows indicate the operations on the application object, the appointment. After the system has created an appointment-object, the user can specify the name and
time-details of the appointment and select the contacts that will attend the meeting. Based on context-information, a selection of the available contacts is made before the user can select the contacts involved in the appointment. The creation of the appointment can be canceled at any moment before the user has confirmed it, after which the appointment will be added to the user’s agenda. The cancelling behavior is specified using the combination of the dashed rounded rectangle (an InterruptableRegion), used to group the actions, and the lightning-bolt edge, indicating an interrupt. The uttermost dashed rounded rectangle is a StructuredActivityNode. It is a meta class that was introduced in UML 2.0, just as the InterruptableRegion and represents a structured portion of an activity.

4.4.4 Integration of Models

Integration of the different models is very important. This section discusses the integration of the presentation and the context model into the activity model. Both models can be integrated using the same mechanism provided by UML: object nodes (which can be linked to classes) and object flows. The idea to use object flows to specify the relations between the presentation and the dialog model was also expressed in UML [da Silva 02].

Table 4.3 provides an overview of how the different effects of object flows, defined in the CompleteActivities package of the UML 2.0 superstructure specification [OMG 04] can be used to specify relations of activities and actions with user interface components (see section 4.4.1) and context collectors (see section 4.4.2). The first two columns give the source and target, while the third and fourth respectively specify the effect (which the designer can opt to omit) and the name of the object flow. The last column explains what an object flow defined by the first four columns means. Table 4.3 shows that an action can aggregate multiple interactions of a user with the system when they logically belong together. An example is the action SpecifyDetails in figure 4.10. The action is linked to an instance of a group component, appointmentDetails, that contains multiple input components as can be seen in figure 4.5.

When an activity or action is specific for a certain context (static context-sensitive or context-dependent integration, see discussion in section 4.3.1), this relation can be defined through the use of activity partitions, which can avoid a lot of clutter in some cases. No additional relations or objects need to be specified to use an activity partition; it can be indicated using a swimlane (formally an ActivityPartition) or, when this is too difficult, by marking the name of the partition between parentheses just above the name of the
Table 4.3: Types of object flows supported to link context collector instances and user interface component instances to actions and activities

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Effect</th>
<th>Stereotype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>⧋</td>
<td>group</td>
<td>create</td>
<td>activate</td>
<td>action triggers appearance of groupComponent</td>
</tr>
<tr>
<td>⧋</td>
<td>group</td>
<td>delete</td>
<td>activate</td>
<td>action triggers disappearance of groupComponent</td>
</tr>
<tr>
<td>⧋</td>
<td>⧋</td>
<td>update</td>
<td>interact</td>
<td>action updates value in outputComponent (1)</td>
</tr>
<tr>
<td>⧋</td>
<td>⧋</td>
<td>read</td>
<td>interact</td>
<td>user receives system output</td>
</tr>
<tr>
<td>⧋</td>
<td>⧋</td>
<td>update</td>
<td>interact</td>
<td>action updates value in inputComponent (1)</td>
</tr>
<tr>
<td>⧋</td>
<td>⧋</td>
<td>read</td>
<td>interact</td>
<td>action reads the value from inputComponent</td>
</tr>
<tr>
<td>⧋</td>
<td>⧋</td>
<td>update</td>
<td>activate</td>
<td>actionComponent</td>
</tr>
<tr>
<td>⧋</td>
<td>⧋</td>
<td>read</td>
<td>interact</td>
<td>system receives event from actionComponent</td>
</tr>
<tr>
<td>⧋</td>
<td>⧋</td>
<td>read</td>
<td>interact</td>
<td>action gathers context data</td>
</tr>
<tr>
<td>⧋</td>
<td>⧋</td>
<td>update</td>
<td>interact</td>
<td>user actions are sensed by contextCollector</td>
</tr>
<tr>
<td>⧋</td>
<td>⧋</td>
<td>update</td>
<td>interact</td>
<td>inputComponent updates application object</td>
</tr>
<tr>
<td>⧋</td>
<td>⧋</td>
<td>update</td>
<td>resume</td>
<td>resumes execution of action, activity</td>
</tr>
<tr>
<td>⧋</td>
<td>⧋</td>
<td>update</td>
<td>suspend</td>
<td>suspends execution of action, activity</td>
</tr>
<tr>
<td>⧋</td>
<td>⧋</td>
<td>read</td>
<td>interact</td>
<td>outputComponent shows application/context data</td>
</tr>
</tbody>
</table>

(1) initialisation in stead of update when static context integration is used
4.4 Models and diagrams

Figure 4.11: Example of activity model using swimlanes

action. Figure 4.11 shows an example activity model in which this technique is applied. The example uses three swimlanes for three contexts of use, GPS on or off, GPS off and GPS on, to specify different behavior for a routeplanner to specify the starting point. When the GPS is turned on, the current location is used as a starting point unless the user, optionally, specifies a different location. The user always has to explicitly specify the starting position when no GPS signal is found.

Figure 4.10 shows the activity diagram with integrated links to the context model and the presentation model. The link with the context model is realized between the context collector AvailabilityChecker, while the other object flows that cross the boundaries of the structured object nodes are links with the presentation model. The AvailabilityChecker gets the available contacts as also shown in the context model in figure 4.8. Note that because all links are
visualized in the diagram in figure 4.10 the diagram looks more complicated. Typically, one will be only interested in a subset of the linked information thus reducing the number of links.

The explicit integration of the model-elements from the abstract presentation model and the context model, has received some criticism from some experienced software engineers regarding semantic correctness although the author could not find inconsistencies with the UML 2.0 superstructure specification [OMG 04], indicating possible problems in usage.

4.5 Discussion

After a detailed discussion of how the profile can be applied, this section provides a short discussion of some usability issues using the cognitive dimensions framework of Green [Green 00] and discusses the coverage of the models using the model elements introduced in section 2.4. Finally, some strengths and weaknesses regarding integration with current software engineering approaches of the introduced models are discussed.

4.5.1 Usability

The viscosity of the notation was kept as low as possible. One way to establish this, is by specifying something only in one place; behavior of the system is only defined using activity diagrams, structure of the user interface is only defined using Nodes, structure of the application and context data only using class (and optionally package) diagrams. This means that when a change is required in the behavior only one model has to be changed.

A comparison of the viscosity of our task/dialog model with that of ConcurTaskTrees (CTT) shows that in certain cases the viscosity of the activity diagram is slightly higher and in other cases the viscosity of the CTT is higher. For example, changing the type of a task from interaction to system in CTT can require a type change for one or more other tasks because of the way how abstract tasks are defined. In our notation changing the type of an action does not require any other changes. On the other hand, creation of a subtask (a more granular activity) is always possible, while in our notation it can require changing the type of action and creation of an activity with the same properties.

Premature commitment can be a problem with the notation, since the links between models are in some cases created by using instances of the linked information. It seems, however, natural that relations are only made after
models have been defined, especially for the defined models. It seems natural to define the models on their own and relate them afterwards since they all describe different aspects of the modeled interactive system. Indeed, the activity model describes the functional aspects of the interaction, while the presentation model describes declarative aspects of the user interface and the application model. Finally, the context model describes the data that is involved.

The visibility of all information is made as high as possible by providing visual representations of the links between the different models in all diagrams. At the same time the number of hidden dependencies is minimized. The high visibility must, however, be relativized because the explicit inclusion of related elements in other diagrams can lead to an information overload for larger diagrams.

The number of extra non-volatile abstractions is kept as low as possible because a great number of abstractions makes initial understanding of the notation more difficult. The role-expressiveness looks quite good at first sight; similar-looking artifacts have similar meanings and many symbols are already familiar to the intended audience. The meaning of the introduced alternative representations for the user interface components and the context collector should be quite clear, although some initial explanation might be needed for the latter. Using a diagram based on the deployment diagram may however confuse some experienced software engineers and thus lower role-expressiveness. They are used to class diagrams to describe structure and deployment diagrams to describe allocation of software modules into a physical environment. Construction of a separate diagram type within a UML-editor as can be achieved in editors like MagicDraw can lower the level of confusion.

4.5.2 Model Requirements

The model coverage of the UML-profile introduced in this chapter is shown in figure 4.12. When one compares the models and the model-elements covered by CUP with the overview of all model elements in figure 2.3 on page 14, one sees that the coverage of the context information is still not complete since it is not possible to describe contexts of use. All other model-elements are however covered.

Another thing that can easily be observed is that some information is duplicated across models; application classes that are accessed through the user interface are duplicated in the abstract presentation model. The activity model can contain elements that refer to elements of all three other models
4.5.3 Integration with Software Engineering

This section provides a discussion of the strengths and weaknesses of CUP, the UML profile defined in this chapter, regarding integration in current software engineering methodologies.

In discussions that followed after presentations about CUP, some issues with the current profile were raised. One of the critiques was that the deployment diagram, extended with some stereotypes, was not a good choice to express abstract user interface descriptions because the semantics of the stereotyped `Node` would not be consistent with that original `Node`, which is a requirement set in the UML 2.0 superstructure specification [OMG 04]. These concerns can definitely hinder adoption of CUP for usage within more rigid software engineering methods.

This possible inconsistency, however, has not prevented adoption of this very model in an approach to generate interactive multimedia applications such as games [Pleuss 05].

Some doubts were expressed whether the notation of the activity model was really legal UML, although no specific details were given. Further examination
of the UML specification and questions to the UML community did not result in a conclusive answer. It did however result in some modifications of the original profile, which were already integrated in the version of CUP, discussed in this chapter. One of these modifications was the usage of instances of the meta-class `InstanceSpecification` instead of `ObjectNode` in the diagram, which was allowed in Enterprise Architect, the UML editor used to express the original CUP.

Further experiments revealed that the current definition of the stereotypes makes it unnecessarily difficult to define the context model. The stereotypes are only defined for classes and not for properties, which means that all information of a class should be marked as either `derivedContext` or `profiledContext`. This makes it impossible to define a users current location, which is derived from the location of his personal device, as a property of a class `User` when almost all other properties consist of profiled information. This makes the usage of the context model unnecessarily complex for experienced UML modelers.

This version of the profile could be used in an approach such as Scrum (see section 2.6) to quickly define some abstract models to sketch ideas. The fact that none of the defined stereotypes has tagged values can make it more difficult the create automatic transformations of the models to code because it could prove impossible for a tool to determine where some meta-data is specified. An example of such meta-data could be the text of a label in a concrete or final user interface that is the result of a transformation from the abstract presentation model.

4.6 Conclusion

This chapter introduced a first version of a UML profile for the specification of context-sensitive user interfaces at a high level. The profile integrates models used in model-based user interface design into UML. This integration is accomplished using a minimal set of extensions to UML, which are implemented in a profile in a UML 2.0 tool [Inc. 06]. Alternative images were based on abstractions already used in UML in the analysis phase (the square for denoting the system, the stick-man for a user, etc.) and to be easy to draw (possibly enabling sketched input for CASE tools). During the design, an evaluation of the effect on usability using the cognitive dimensions framework was performed and the notation was tested through the specification of a small example application.

This first version of CUP was complete enough to model some context-
sensitive user interfaces and perform an initial, limited evaluation. The experiences gained using this profile were then used to define a second version of the profile that is more refined and corresponds better to the expectations of the software engineering community.
Chapter 5

CUP 2.0: Modeling multi-context user interface

5.1 Introduction

Chapter 4 introduced the context-sensitive user interface profile (CUP). It was defined to allow high-level specification of context-sensitive user interfaces. Although it allowed to express the dynamic context influences, one could not (easily) use the profile to define multiple contexts of use for the modeled interactive applications. Furthermore, it was necessary to make some high impact changes to CUP to tackle some of the critique that the profile received from the software engineering community. Therefore, it was decided that a radically new version of the profile should be defined which had the following features:

- It should be consistent with the UML syntax and semantics.
- It should offer the capability to specify different contexts-of-use.
- Stereotypes should have tagged values for meta-information that can be useful for automated transformations.
- It should be possible to define the target platforms of the designed user interfaces.

This chapter introduces the revised UML-profile, CUP 2.0. The rest of this chapter starts with an overview of all models that can be expressed with the newly defined profile. The following sections then discuss these models into more detail. Finally, a discussion of the possibilities the profile offers
for code generation is given, followed by a discussion of the changes in the profile compared to the version of CUP discussed in the previous chapter and a conclusion.

5.2 Model Overview

The Context-Sensitive User interface Profile (CUP 2.0) is a UML 2.0 [OMG 04] profile that provides stereotypes and corresponding tagged values to increase support for the expression of the models, relevant to the high-level modeling of context-sensitive user interfaces, in a limited number of diagrams. Figure 5.1 gives an overview of the models that can be specified using the CUP 2.0 profile.

The application model specifies the data structures and functionality that can be accessed through the user interface. This includes the data structures and functionality that is not part of the modeled application but that is used to provide relevant information (context) to the application. The model is used by both 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 modeled application and in the user interface structure. The model is discussed in more detail in section 5.3.

A second model is the system interaction model. This model corresponds to the user task model, which is the core model in many model-based user interface design approaches. It is an hierarchical specification of the user’s tasks and user-observed tasks. In contrast to the most-used task model notation, the ConcurTaskTrees (CTT) notation [Paternò 00], it does not use a tree-based notation but uses the flow-based notation of the activity diagram.
5.3 Application Model

The application model is specified using a class diagram. The model minimally contains all classes of the application logic that are relevant for the user interface. In addition to those classes, also the context information and the interfaces of the relevant applications or services to get the relevant context information are included in the model. The latter classes are respectively identified using the stereotypes <<context>> and <<contextCollector>>. Both

Figure 5.2: Example of user interface deployment model: A context-sensitive mobile museum guide.

All temporal operators that are supported by the ConcurTaskTrees notation are supported in CUP 2.0. Furthermore, the system interaction model is enhanced with support for context-sensitiveness. More details about this model can be found in section 5.5.

The structure of the context-sensitive user interface is specified in the abstract user interface model. A single model represents a user interface structure that is shared in multiple contexts and on multiple platforms (see section 5.6). The deployment of an abstract user interface to a certain platform or to a set of platforms, possibly for distributed user interfaces as is discussed in [Luyten 06], can be specified in the user interface deployment model. To accomplish this, the stereotype <<contextualNode>> can be applied to a Node to specify the relation with a certain context of use as specified in the context model. Figure 5.2 shows an example of a deployment of the user interface to a PDA. Specific contexts of use can be specified in the context model, which uses the classes defined in the application model. More details of the context model are found in section 5.4.

5.3 Application Model

The application model is specified using a class diagram. The model minimally contains all classes of the application logic that are relevant for the user interface. In addition to those classes, also the context information and the interfaces of the relevant applications or services to get the relevant context information are included in the model. The latter classes are respectively identified using the stereotypes <<context>> and <<contextCollector>>. Both
stereotypes were already present in the first version of CUP, although they were used in the context model. The context model, as defined in section 4.4.2, is merged with the application model in CUP 2.0. The reason for this merge is that the context model in our modeling efforts frequently contained information that was also present in the application model, which could lead to inconsistencies between the models.

The context model in the first version of CUP contained two other stereotypes: «profiledContext» and «derivedContext» which were derived from the stereotype «context». This situation was however not ideal as discussed in section 4.5.3. Therefore these stereotypes were removed and replaced by two other stereotypes which extend the meta-class Property.

This means that in CUP 2.0, each Property of classes with the stereotype «context», can have a stereotype indicating how the modeled information is gathered since this information can be important for the further design or eventual code generation. The two stereotypes that are supported are «detected» for context information that is delivered to the application directly from sensors or from any source after being manipulated, merged or derived by some service or application. Profiled context information is provided by an application or entered by a user and is indicated by the stereotype «profiled». The difference is also clear from the tagged values of these stereotypes. While the values of profiled context information can be gathered from a resource of a certain type (e.g., a URI referencing a file), the detected context information is gathered from a context collector. The choice to categorize context in profiled and detected was motivated by the implications this difference has on the design of the application; an appropriate user interface has to be defined to modify profiled context information, while...
5.4 Context Model

The context model in CUP 2.0 specifies the different situations in which an application can be used, which was not possible in the version of CUP discussed in chapter 4. For each situation or context of use, the context model contains a package with the stereotype «contextOfUse». Such a package can only contain instances of classes that have the stereotype «context» as specified in the application model. As such the context model is more open than the context model used in UsiXML [Limbourg 04b], which uses instances of predefined classes to specify the contexts of use. Just as packages can be

Figure 5.4: Example of application model: A context-sensitive mobile museum guide.

detected information requires mechanisms to detect the information and possibly appropriate feedback to the user when problems are encountered. This categorisation of context is more extensively motivated in section 4.4.2.

The stereotypes that can be applied in the application model are shown in figure 5.3, while figure 5.4 shows an example application model. The example shows a partial application model of a museum guide. It clearly shows that many relations exist between parts of the model that are part of the context and those that are not. It also shows that the location of a user is detected by a LocationDetector, while the location of the museum artifacts is profiled.
nested, contexts of use can also be nested. When multiple contexts of use are nested within another context of use $A$, $A$ is defined to be the union of the nested contexts of use.

iCAP [Sohn 03] is a tool that allows a designer of a context-aware system to graphically create rules for context-aware adaptation. A screenshot of the tool is shown in figure 5.5. The area in the middle specifies one or more contexts of use, which express the “if”-part of the rule, the area on the right represents the output of the application. Each type of context information is represented by an image that can be dragged upon a panel that represents a context of use. Multiple panels can be used, each containing one or more images. Each image is associated with a value for that specific type of context information. The values are shown in the lower left of figure 5.5. Different panels are combined using a logical or, while the conditions represented by the images within one panel are combined using a logical and.

Each instance specifies one value to which a parameter of the context of use has to adhere. The context of use is defined by combining all specified values using the logical and condition. One notable exception to this rule is when multiple values are specified for one property or class. Ranges of values can be

![Figure 5.5: Screenshot of iCAP [Sohn 03]](image)

Figure 5.6: Stereotypes of the UML profile CUP 2.0 relevant for the context model
specified by specifying a minimum and a maximum (using the corresponding stereotypes), or by listing the possible values; when multiple instances of the same class are specified they represent alternatives. To avoid ambiguity, when both a minimum and a maximum value is provided, the instances providing these values should be linked. Figure 5.6 shows the stereotypes that can be applied to the model elements, while figure 5.7 shows a small example model, demonstrating the usage of the different stereotypes. The specified context of use is relevant for users that follow a dynamic tour through the museum and have a PDA with a certain minimal resolution.

The logical combination of contexts of use in our approach is very similar to that of iCAP: values within one context of use are combined using a logical **and**. When different contexts of use are combined, a logical **or** is used, in our approach multiple contexts of use can be associated to a task, while in iCAP they are combined when the resulting output is the same. The representation, however, is entirely different. Matching values for context information are represented by objects in our approach and by images in iCAP. In our approach a context of use is represented by a stereotyped package, in iCAP it is represented by a panel.
Table 5.1: Icons of task categories in ConcurTaskTrees, Contextual ConcurTaskTrees and CUP 2.0

<table>
<thead>
<tr>
<th>Task Category</th>
<th>CTT</th>
<th>Contextual CTT</th>
<th>CUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract task</td>
<td>📌</td>
<td>📌</td>
<td>/</td>
</tr>
<tr>
<td>User task</td>
<td>📌</td>
<td>📌</td>
<td>📌</td>
</tr>
<tr>
<td>Contextual User Task</td>
<td>/</td>
<td>📌</td>
<td>📌</td>
</tr>
<tr>
<td>Application task</td>
<td>📌</td>
<td>📌</td>
<td>📌</td>
</tr>
<tr>
<td>Contextual Application task</td>
<td>/</td>
<td>📌</td>
<td>📌</td>
</tr>
<tr>
<td>Interaction task</td>
<td>📌</td>
<td>📌</td>
<td>📌</td>
</tr>
<tr>
<td>Contextual Interaction task</td>
<td>/</td>
<td>📌</td>
<td>📌</td>
</tr>
<tr>
<td>Environment task</td>
<td>/</td>
<td>📌</td>
<td>📌</td>
</tr>
</tbody>
</table>

5.5 System Interaction Model

The system interaction model describes the interactions of the system with the user and the environment in which it is executed. It can be used to describe the tasks of both the users and the application as well as the relevant interaction with the environment in more detail. The basis for the system interaction model is the UML 2.0 activity diagram. In this model, all actions have to have the stereotype «task» or a derived stereotype applied to them.

A task corresponds to an UML Action. The task goal can be expressed using a local postcondition, if desired. Basic tasks – tasks that are not refined within the model – belong to four different categories. These categories are based on the categories of the Contextual ConcurTaskTrees [Van den Bergh 04a] notation, an extension of the earlier mentioned ConcurTaskTrees notation that allows for specification of context influences. We defined four stereotypes with the appropriate tagged values, that cover all task categories present in the Contextual ConcurTaskTrees as can be seen in Table 5.1.

One notable difference is the elimination of the task category type abstract task, which is a task that can be refined into tasks that belong to different categories. Since there are a great number of ConcurTaskTrees models that do not follow this definition and a change in semantics would only be confusing, this task category is not present in this UML profile and a generic stereotype «task» is used instead. In practise, this has the consequence that CallBehaviourActions and StructuredActivityNodes have to have the stereotype «task» and not one of the derived stereotypes.

This is contrast with the first version of the CUP-profile where these classes were not allowed to have any task-related stereotype applied to them. This
5.5 System Interaction Model

Figure 5.8: Stereotypes of the UML profile CUP 2.0 relevant for the system interaction model

was changed to achieve greater consistency and to make it possible to associate meta-data with each Action/task in the model through the tagged values that are associated with the defined stereotypes.

The four stereotypes that correspond to the non-abstract task categories are:

«userTask» A user task is a task that is performed by the user without direct interaction with the application. A user task can however have indirect impact on an application. E.g. A museum visitor might carry an electronic mobile guide while strolling, performing no direct interaction. The electronic guide can however get updates about the position of the user through the use of a positioning system in the museum. This can be modeled by applying the stereotype to an AcceptEventAction and specifying an interface to the positioning system in the tagged value contextSource. User tasks that are applied to other types of Actions are optional and will not be used during further specification of the system.

«applicationTask» An application task is a task performed entirely by the application without user interaction. Examples of such tasks are showing information to a user or performing a computation. When an application task has influence on the platform or the environment, the affected data structures or systems can be indicated through the tagged value manipulatedObject. Examples of such influences are putting information in the system paste buffer and triggering an external logger that has an
Direct user interaction with an application is modeled with an interaction task. Like the previously mentioned tasks, an interaction task can have effects on the environment which are indicated with tagged values. The type of user interaction is indicated through the tagged value interactionType.

An environment task covers all actions that have an influence on the execution of the interactive application but are performed by an entity other than the user and the application. An example of an environment task is a car accident that happens on the route calculated by a car navigation system. Similar to the user task, an environment task will be modeled through an AcceptEventAction when it has an immediate effect on the execution of the application, such as in the example of the car accident, which triggers a recalculation of the route.

All stereotypes indicating task categories are derived from the stereotype <task>, which defines some tagged values that are shared by all task categories. These tagged values are important to reduce the complexity of the diagrams: the tagged value optional indicates whether or not a certain task is required or not, while the tagged value repetition indicates the number of times a task should be executed. The tagged values manipulatedObject and requiredContext are only applicable to basic tasks and thus are required to be empty sets for the stereotype <task>. Figure 5.8 gives an overview of the stereotypes and their tagged values.

If the tagged value singleExecution is set to true for a certain task, that task interrupts all other tasks that are running in parallel until it is completed. This has as consequence that when all actions following a ForkNode have this tagged value set to true, they have to be carried out one after the other. This makes that all temporal operators supported by the ConcurTaskTrees notation can be expressed using the UML activity diagram when the stereotypes in figure 5.8 are applied as can be seen in Table 5.2.

An example of a system interaction model can be seen in figure 5.9. The example shows a partial specification of a mobile museum guide that offers different types of tours. The diagram gives only details about one type of tour: the dynamic tour. This type of tour does not offer a specified trajectory to the user, but shows the user’s position in the museum as well as information about a nearby artwork if one is available. A user can ask more information about an artifact. This additional information temporarily blocks all other information.
### Table 5.2: Temporal operators in ConcurTaskTrees and corresponding activity diagram notation

<table>
<thead>
<tr>
<th>Temporal operator</th>
<th>Symbol</th>
<th>Activity diagram constructs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabling</td>
<td>$&gt;&gt;$ and $</td>
<td></td>
</tr>
<tr>
<td>Disabling</td>
<td>$&gt;$</td>
<td>InterruptableActivityRegion with InterruptionEdge <a href="#">Diagram</a></td>
</tr>
<tr>
<td>Concurrency</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>Choice</td>
<td>$[]$</td>
<td>Decision and MergeNodes with control flows <a href="#">Diagram</a></td>
</tr>
<tr>
<td>OrderIndependent</td>
<td>$</td>
<td>=</td>
</tr>
<tr>
<td>Interruption</td>
<td>$&gt;=$</td>
<td>concurrency with tagged value singleExecution set to true for the interrupting task</td>
</tr>
</tbody>
</table>

Table 5.2: Temporal operators in ConcurTaskTrees and corresponding activity diagram notation
This example clearly shows the second major difference between the activity model of CUP’s first version of which an example is shown in figure 4.10. The links with the abstract user interface components and context collectors represented in the diagram using object flows and object nodes. The links with the abstract user interface components is now implicit; the abstract user interface component and the correspond task in the system interaction model share the same name. The links with context collectors are specified using tagged values.

5.6 Abstract User Interface Model

The abstract user interface model provides information about the structure of the user interface independent of the platform it will ultimately be deployed on. This means that abstractions from the concrete components are defined thus drastically reducing the number of components and resulting into a min-
5.6 Abstract User Interface Model

The components are differentiated according to the functionality they offer to the user. The four types of abstract user interface components introduced in section 4.4.1 are retained in this version of the profile: input components, which allow users to enter or manipulate data, output components, which provide data from the application to the user, action components, which allow a user to trigger some functionality, and group components, which group components into a hierarchical structure. These abstractions are the same as in the earlier version of the profile which is discussed in section 4.4.1 and gives a comparison with related work.

In this version of the profile, the abstract user interface model (AUIM) is represented using a class diagram, the diagram type mostly used in UML to model structural aspects. All classes in a AUIM need to have a stereotype identifying a type of abstract user interface component. There are also restrictions on the associations that can be specified between the classes, they need to indicate containment or have one of the stereotypes discussed later in this section applied to them. The definition of the stereotypes is shown in figure 5.10. Only one of these stereotypes can be applied to one class. There is one exception to this rule: a group component can also be an input component, but in this case the input component must be a selection over the contained user interface components.

One should note that the classes with the stereotypes «inputComponent» or «outputComponent» can each have multiple attributes that would each be
represented using a separate user interface component in a notation such as the Canonical Abstract Prototypes [Constantine 03a]. Each of the attributes has the stereotype «uiData». The tagged value propertyInClass can be used in case there is a reference to a property of a class. Additional meta-information, such as a label or more detailed information can be provided using the remaining tagged values. All Operations related to an action component must have the stereotype «uiAction» that allows to specify information similar to the stereotype «uiData» for each Property of an input component or output component.

The visibility specification for each Property and Operation with the stereotype «uiData» or «uiAction» is adapted to be more relevant to their meaning in the model, but remains consistent with the UML specification:

**public** Public visibility means that the associated part of the user interface is visible to not only the user of the application, but also other persons that might see the user interface. This visibility is, for example, appropriate for the part of a presentation application that shows slides.

**protected** Protected visibility means that the associated part of the user interface is only visible to the user of the user interface. This might mean that the value of an input component with protected visibility is hidden when shown on a public display. An example of user interface components for which this visibility is appropriate are the controls for moving through slides in a presentation application.

**package** Parts of the user interface that have package visibility are only accessible to other parts of the user interface, but are not shown to the users of the user interface. This visibility should be avoided in the abstract user interface model.

**private** Private visibility is used for parts of the user interface whose contents may not be seen by a user without being masked. An example of a user interface component with private visibility is a password field.

We also defined some stereotypes for associations between abstract user interface components to express relationships other than containment. These relationships indicate constraints on the structure of the user interface which are implied by the system interaction diagram and thus reduce the number of hidden dependencies within the abstract user interface model. These relationships can also be used to specify relationships between user interface components within the model that are specified in different diagrams. At the
The reduction of hidden dependencies is important to effectively support modification, a good visibility is also important for exploratory design [Green 98].

The first stereotype is <<precede>>, which indicates that one user interface component should be presented to a user before another user interface component. The precedence can be spatial or temporal. The usage of this stereotype is limited to user interface components that are contained by the same group component and can be used to establish an order in which the user interface components are presented to the user. A second stereotype, <<activate>>, can be applied to an association to indicate that a user interface component activates another component. The activated components can be added to the currently active components or can replace them. A third stereotype for associations is <<update>>. Application of this stereotype to an association indicates that the contents of the target user interface component is updated by the source user interface component.

An example of an abstract user interface model is shown in figure 5.11. The depicted model corresponds to the part of the system interaction model that shows the functionality offered in the case of a dynamic tour. The figure shows three group components that the user can interact with. The first group component contains one interaction component that allows the selection of a type of tour. When the user selects a type of tour, a second group component is activated and replaces the one that contains the interaction component, as can be seen from the tagged values on the association. This group shows
Figure 5.12: CAP version of the group components IntroScreen and Map Display

a map, the current user position and, optionally\(^1\) some information about a nearby artwork and, also optional, an option to show more information about the artwork. This information is shown within a group component Extended Info, which replaces the group component MapDisplay.

Figure 5.12 shows a subset of the abstract user interface model in figure 5.11. It shows the group components IntroScreen and View Map and the user interface components they contain. It clearly shows the different nature of both notations. While Constantine [Constantine 03a] acknowledges that the connection with the domain model is very important for an abstract user interface model, in the Canonical Abstract Prototypes notation, this connection remains limited to at the most the usage of names from the domain model for the corresponding abstract user interface components. It is entirely optimized to give an abstract indication of the layout of the screens of the final user interface as well as the functionality that that final user interface provides to the user. The Canonical Abstract Prototypes notation is actually a concrete user interface model because it is specific for graphical user interfaces and thus the modality is already fixed.

The abstract user interface description shown in figure 5.11 emphasizes the elements of the domain/application model and has no attention to the visual layout of the final user interface. This is a natural choice, since this notation is independent of the final platform and modalities.

The defined abstract user interface description assigns one type of user in-

\(^1\text{This can be derived from the multiplicity specified for the containment relations.}\)
5.7 Prototype Generation

We have identified two main areas where transformations as specified in the model-driven architecture [Miller 03] can be applied. The first is a model-to-model transformation from the system interaction model to the abstract user interface model. The second is the generation of high-level user interface descriptions from the abstract user interface model. The user interface deployment model can be used to add style to the different user interface skeletons and add some design guidelines specifically for the target platform.

To test the feasibility of the prototype generation, a XML-based description using XHTML [Pemberton 02] + XForms [Dubinko 03] was chosen as a target language for prototype generation. The mapping of the elements in the abstract user interface model to XForms tags is shown in Table 5.3. A ≪uiAction≫ is translated into a submission if a value is specified for the tagged value operationInClass, and into a trigger otherwise. In XForms each component can make references to separately defined object structure in instances. This object structure as well as its XML Schema [W3C 03] can be derived from the tagged value propertyInClass of the attributes with a ≪uiData≫ stereotype. The fully-qualified name of its class can be used to generate a meaningful hierarchy of XML-tags, while the datatype itself can be used to define the types

<table>
<thead>
<tr>
<th>CUP-profile</th>
<th>XForms tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</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>input</td>
</tr>
<tr>
<td>selectionType is none</td>
<td>select</td>
</tr>
<tr>
<td>max. selectionCount = 1</td>
<td>select1</td>
</tr>
<tr>
<td>max. selectionCount &gt; 1</td>
<td>output</td>
</tr>
</tbody>
</table>

Table 5.3: CUP 2.0 stereotyped Elements and XForms counterparts

teration to a component, similar to the approach taken for XForms [Dubinko 03], UMLi [da Silva 03] and Wisdom [Nunes 01]. TERESA XML [Mori 04] also uses this approach but defines a deeper hierarchy that contains special components for inputs of simple datatypes and selections based on the number of options. UsiXML [Limbourg 04b] only has one type of user interface components having facets that are based on the type of interaction.
in XML Schema.

The effects of the activation of components can be converted to bind tags with the right relevant settings. Precedence relations between user interface components are reflected in the order of the corresponding XForms controls in the document. The update relationships can also be translated into bindtags with the right nodeset and optionally calculate attributes. Conversion of application or context-driven updates to the user interface are more difficult since they cannot be described declaratively in XForms.

![Image](image.png)

Figure 5.13: Final interface based on AUIM in figure 5.11

An example of an XForms interface that can be obtained using XForms and CSS can be seen in figure 5.13. Notice that the interface shown in the figure is
not automatically generated. The upper part of the figure contains allows one to specify the context information, in this case the user’s location. The bottom part contains the actual user interface. The layout used for the different user interface components is similar to that specified using the CAP notation in figure 5.12. Only the group component `MapDisplay` and its contents are shown in the figure. The output component `show map` takes up the whole space of the group component. All other user interface components are shown before it. The output component `Show Location` is shown in the lower left corner (using a light-blue background), the output component `Show Artwork Info` is shown in the upper-right corner just above the action component `Ask More Info`.

The output component `Show Artwork Info` as well as the action component `Ask More Info` are only shown when the user’s location is close to some artifact; in this case location 1, the art museum. All images used in figure 5.13 are courtesy of `http://artmuseum.msu.edu/`.

### 5.8 Analysis of changes in CUP

This section provides an overview of the changes in the CUP, as discussed in chapter 4 and their relation the goals for the new version of the profile set in section 5.1.

One of the most remarkable changes is the changes in the definition of the application model and the context model. The definitions of context collectors and the types of context information are specified in the application model instead of the context model in the first version of the profile. This change was motivated by the fact that the context model, as defined in the first version of the profile, often duplicated information that was already defined in the application model. This created some hidden dependencies; one should not create inconsistencies between the two specifications, which were removed by merging the two models. The classes that are part of the context are, however, still easily noted due to the fact that they are stereotyped.

A second change was motivated by the fact that extended modeling experience indicated that the indication of whether some context information is detected or profiled was better (also) indicated at the property-level because in many cases some attributes of a class are detected and some others are profiled, causing an indication only at the class-level to be difficult to impossible.

The context model in CUP 2.0 defines the contexts of use for the tasks, which makes the model more consistent with the context model as it is found in UsiXML [Limbourg 04b] and similar to that of iCAP [Sohn 03]. The context
model thus completes the support of the model elements for context information as defined in section 2.4.

Also the abstract user interface model had some notable changes; the diagram that is used for the graphical notation is based on the class diagram in version 2.0, while the original version used the deployment diagram. This change made the notation more consistent with current software engineering practices, which use the class diagram to define structural features. Furthermore, the Wisdom approach [Nunes 01] also uses the class diagram to define the abstract user interface, while other approaches that extend the meta model use a notation [da Silva 03, Nóbrega 05a] more similar to that of the original version of CUP. The former has the most resemblance to the presentation model of the original version of CUP, while the latter is based on the Canonical Abstract Prototypes notation [Constantine 03a]. It is, however, worth noting that tree-based structures were also used in model-based user interface design to define the presentation of the user interface. A recent example of this is DynaMo-AID [Clerckx 04a].

While Wisdom also uses the class diagram for the abstract user interface model, it has some notable differences. The Wisdom notation uses stereotyped classes to define AIO groups (definition see section 2.4), but attributes for the abstract interaction objects (no type is given to any of the attributes in the examples in publications). Our approach differs at first sight; all user interface components are defined as classes turning all user interface components into reusable, top-level entities. These can be used to let all these user interface components match with exactly one action in the system interaction model. Meaning that one action can be used to indicate the manipulation or consultation various data of the application model.

The stereotypes «input element», «output element» and «action» in the Wisdom notation correspond to the combination of the stereotypes «uiData» with either an «inputComponent» or «outputComponent», and «uiAction» with «actionComponent». Splitting these stereotypes allows the specification of complicated, nested user interface elements such as tables which allow selection of a set of sub controls with one action by combining the stereotypes «inputComponent» and «groupComponent». Making all user interface components classes also allows the specification of associations between all types of user interface components where Wisdom only allows the specification of relations between views, which correspond to group components.

All stereotypes in the new version of the profile provide appropriate tagged values which allow explicit specification of relations with other models and other meta information about the entities of the model, which could enable
generation of user interface prototypes as indicated in section 5.7. Finally, the user interface deployment model can provide guidance for the generation of platform specific models from the platform independent models defined in this chapter, as specified in the Model-Driven Architecture [Miller 03].

5.9 Conclusion

One can conclude that the revised UML profile, CUP 2.0, presented in this chapter offers some benefits over related approaches and the earlier version of this profile. The current version of the profile allows a detailed description of both the behavior and structure of the user interface of context-sensitive interactive applications using a limited amount of constructs of the UML using regular UML modeling tools that allow metamodel extension through profiles.

The profile also allows a clear specification of all datatypes that are involved, allowing to make optimal use of specifically designed user interface components for complex datatypes on platforms where they are available. Finally, the fact that all information is expressed in UML makes it easier to integrate the user interface specification with the specification of the application core.
Chapter 6

Tool Support and Evaluation of Models for Context-Sensitive User Interfaces

The previous chapters in this part of the dissertation introduced extended models for the design of context-sensitive user interfaces. Chapter 3 introduced the Contextual ConcurTaskTrees notation, an extension of the ConcurTaskTrees notation [Paternò 98], for creating task models for context-sensitive interactive applications. Chapter 4 introduced a first version of UML-based support for other high-level models through the introduction of a UML profile, called CUP. It was then refined and extended in a second major version of this UML profile, CUP 2.0, as discussed in chapter 5. All models can be expressed using UML 2.0 provided that the appropriate UML profiles defined in this part are used.

This chapter presents the established tool support and presents an evaluation of the models. The evaluation will primarily concern the UML-based Contextual ConcurTaskTrees notation and the latest revision of CUP. A first part of this evaluation is discussed in section 6.2.1 and uses the same metrics used to evaluate the related work in section 2.5. Section 6.2.2 then discusses a complete model-based approach and discusses possibilities to integrate the models in the software engineering methods discussed in section 2.6.

6.1 Tool Support

The following requirements were set for good tool support:

1. The tool has to have complete support for the parts of the UML 2.0
1. It must allow the definition of UML profiles, preferably with a serialization to XMI including the option to define alternative notations for the stereotypes that are part of such a UML profile.

2. It should be as easy as possible to create correct models with the tool.

3. The created models should be viewable without paid for tool support.

The first requirement was set to minimize lock-in. When the tool is aware of the UML 2.0 meta model and can serialize the created models to XMI it is more likely that one can later use other tools to edit or transform the created models. Although in reality this is not a guarantee due to incompatibilities between the different XMI implementations, this is the best that can currently be established. Furthermore, should the UML meta model not be supported it is less likely that UML profiles can be properly supported.
6.1 Tool Support

Good support for the definition of UML profiles is the second requirement and essential for a tool that is used to create the models introduced in the previous chapters. Good support means that alternative notations should be supported for stereotyped UML elements, since these can ease the comprehension of the models by making it easier to identify what the purpose of each of the stereotyped elements is. Serialization of the profile to XMI, and especially of the models that use the profile is essential to use the created models in other tools.

The third requirement was set to minimize error proneness, as defined in the Cognitive Dimensions Framework (see appendix A), while the fourth and final requirement should ensure that the barrier to view the created models is as low as possible.

Based on these requirements, we selected MagicDraw [Inc. 06] to define the UML profiles and create models. It serializes it models to XMI 2.1 by default. It allows the creation of UML profiles, which can be exported in XMI for easy exchange with other users of MagicDraw. It allows one to create custom diagram types which have tool bars whose content can be completely optimized for the chosen models. These custom diagram types can also be exported and used by other users of the this UML editor.

The tool enforces many (but not all) constraints imposed by the UML 2.0 meta model. There are different versions of MagicDraw with varying capabilities of which the most basic one is available free of charge. This version, just as all other versions, can open all models created by other versions but can only edit a subset of those models.

Using the serialized models in other editors or transformation tools was far from optimal in the version we used to create the models shown in this dissertation (version 11). The latest version available at the time this dissertation will be published, however, will be able to export its models to the XMI generated by the tools based on or using EMF [Foundation 06] and also import models created by these tools. Checking the adherence to OCL-constraints for UML profiles is another feature that should be available in that release.

The tool however has some other drawbacks. The most important one for the work presented in this dissertation is the lack of graphical support for some meta classes used in the activity diagram such as InterruptableRegion and InterruptableEdge. The graphical appearance of these elements in the models in this dissertation has been created by using some additional stereotypes. Finally, some minor bugs were discovered (and reported) during the usage of the tool.
As can be seen in figure 6.1, the Contextual ConcurTaskTrees can be expressed using a custom diagram in UML 2.0 using a visual appearance which is very close to the original ConcurTaskTrees notation. The differences that do exist are discussed in section 3.6. The specification within the UML tool has some drawbacks and advantages in the interactive construction of the diagram. Some guaranteed properties which were by default true in the original CTT tools (see table 2.3 on page 28) no longer hold by default in the UML-based tool.

The most important property is that of the placement of the tasks: it is not possible to guarantee as it is that all temporal relations are guaranteed to have the source task on the left and the target to the right. Similarly, subtasks are no longer by default below their parent tasks. One should however note that until version 1.59 of the ConcurTaskTrees Environment, those properties are not guaranteed to hold during editing (i.e. one can change the placement such that those properties are no longer true). This also has a positive effect; namely the juxtaposability of diagram elements is better in this way. A property in the Cognitive Dimensions Framework [Green 98] that is considered to be important in both modification and exploratory-design activities.

Tools such as MagicDraw allow to show tagged values and properties within the diagram for all tasks or for specific tasks, increasing the visibility of certain information. A drawback is that often this is currently most of the time a all or nothing approach for each type of information where a type of information is for example a tagged value, an attribute or an operation.

The UML profile defined in section 3.6 does not use any of the stereotypes specified in the CUP 2.0 profile as defined in chapter 5 to enable usage independent of the latter profile. In case both profiles are used, additional constraints can be defined to ensure proper usage of the models. These constraints than include on tagged values such as contextOfUse]. All UML elements referenced in this tagged value must be packages with the stereotype <<contextOfUse>> applied to it.

For each of the models in the CUP 2.0 profile, a custom diagram type was defined resulting in the specialized toolbars shown in figure 6.2. The definition of these custom diagram types makes that the designers are exposed to the minimal number of abstractions necessary to define the models, which should lower the entry barrier to using the notation [Green 98]. Note that when desired, one can still opt to use the standard diagram types and manually apply all stereotypes.
6.2 Evaluation

6.2.1 Analysis of the UML-based Notations

This section discusses CUP 2.0 and the Contextual ConcurTaskTrees (UML-version) using the requirements set in the section 2.5 and discussed for the related models. Table 6.1 gives an overview of the qualitative properties of the notation. Our notation is supported by UML tools. The support has been defined in detail for the MagicDraw tool (see section 6.1).

The expressiveness of the notations is quite good; integration of the Contextual ConcurTaskTrees notation into UML offers some advantages; metadata for tasks, specified in tagged values (see figure 3.5 on page 45), can be shown directly in the diagram when desired. Furthermore, UML offers a formal syntax for the specification of pre-conditions via the Object Constraint
104 Tool Support and Evaluation

<table>
<thead>
<tr>
<th>Tools</th>
<th>UML-tools (MagicDraw)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expressive</td>
<td>Inclusion of Contextual CTT in UML lifts some of the drawbacks</td>
</tr>
<tr>
<td>Splitting</td>
<td>separate specification of models</td>
</tr>
<tr>
<td>Linking</td>
<td>explicit links, though not graphical</td>
</tr>
<tr>
<td>Solid Base</td>
<td>UML, CTT</td>
</tr>
<tr>
<td>Specification</td>
<td>public (UML Profiles)</td>
</tr>
</tbody>
</table>

Table 6.1: Qualitative Evaluation

Language (OCL) [OMG 06b]. The specification of pre-conditions for tasks is a part of the original ConcurTaskTrees notation, there was however no possibility to express these constraints using a formal specification. Integration in UML, means that better splitting and linking is possible; the task model can refer to specialized models for the specification of the context of use and manipulated objects. For example, the domain objects expressed in a application or domain model can be referenced from within the Contextual ConcurTaskTrees notation.

Both CUP and the UML-based Contextual ConcurTaskTrees notation, build on further on the research and industry experience that resulted ConcurTaskTrees and UML [OMG 04]. More specifically the specification of activity diagram and its semantics based on Petri Nets [Desel 01] and the specifications for structures (as provided in class and object diagrams) and deployments are used from UML.

The UML profiles and the custom diagram types are publically available for both the Contextual ConcurTaskTrees as the (models that are part of) CUP 2.0 at [http://research.edm.uhasselt.be/jvandenbergh/cup](http://research.edm.uhasselt.be/jvandenbergh/cup).

Although all models have a graphical and a textual notation as shown in figure 6.3. The textual notation is not easily human-readable since it is provided via XMI [OMG 05].

Usage of the UML and its standard light-weight extension mechanism, profiles, does not automatically imply that it is easy to use by software engineers. This is partly due to the enormous size and complexity of the UML specification. As an illustration, the UML Superstructure Specification, which is only one of the four parts of UML 2.0, contains well over 700 pages and is significantly larger than the UML version used by Nunes [Nunes 01]. He already proposed the usage of a subset of the specification for projects developed by small teams.

The specification of the Contextual ConcurTaskTrees notation and the
models covered by CUP (2.0) require the usage of a limited subset of the specification: three (out of six) structure diagram types, one out of in total seven behavioral diagram types (including four interaction diagram types). The three structure diagram types are used in modified versions for five different models (task model, application model, context model and user interface deployment model), resulting in the same amount of structural diagram types. The one behavioral diagram is also slightly modified for one model (the system interaction model). This means that our notation requires the usage of only six types of diagrams whereas the original specification contains thirteen diagram types [Wikipedia 06].

Figure 6.3 shows that the proposed models cover all information elements defined in figure 2.3 and section 2.4, except those specified for the concrete user interface model (see definition 2.8), which lies out of the scope of this dissertation. Note that this figure does not show which model elements are used when one diagram is using the other to avoid information overload in the diagram. The interested reader is referred to chapter 5 for more information about the usage of the model elements in the diagram types ¹. The diagram

¹Note that this information is not directly represented in chapter 5, but can easily be
Figure 6.4: Model elements used in the user interface deployment model

only shows the model-elements only where they are defined. If one shows
the identified model elements in all models where instances of these model
elements are defined, the deployment diagram should look in figure 6.4.

6.2.2 Usage of the UML-based Models in Software Engineering Approaches

Model usage can ease the design of context-sensitive interactive systems, while
not preventing an agile methodology. One is not obliged to use all the proposed
models, which would be prohibitive to their use in agile methods such as Scrum
(see section 2.6), certainly when code generation from these models is not
available.

When looking at integration with RUP (see section 2.6), the task model,
the system interaction model and the application model can be a part of the
analysis or design model. The user interface deployment model can be a part
of the deployment model, while the abstract user interface model can be used
to define navigation maps. Given concise definition of the models and proper
tool support, user interface prototypes could be generated from the models. In
the remainder of this section, we will discuss how the proposed models could
be used in a model-driven process.

Figure 6.5 shows an overview of the models made during the design and
the relations between them. The models with the colored background are the
ones that are discussed in this work, the ones with a white background are
not. The relations between the models are indicated using dependencies and
package imports. Whenever a dependency is used, the dependent model can
be generated or derived from the other model. A package import is used when
one model uses information specified in the other.

The process starts by specifying the users tasks, elicited from scenario’s
[Paternò 01] or from use cases [Kruchten 03], and the concepts that are im-
portant to perform these tasks. These tasks and concepts can be described
derived when juxtaposing the diagrams in that chapter with the diagram in figure 6.3.
using the Contextual ConcurTaskTrees, as introduced in chapter 3, and the application model as introduced in chapter 5.

The concepts can be annotated with context information to specify which concepts represent context information that will be used at runtime and whether it will be generated (automated generation) or profiled (manual input). Also the sources of the context information can be specified. For each of the tasks in the task model, one can specify whether context is involved; is the effect of a task noticed by an external entity (and fed back to the system) or is it executed by another system, etc. Also the context of use can be specified for each task in a context model. Based on these “context-sensitive” models, one can generate (high-level) system interaction models and abstract user interface models. These models can be expressed using the UML. When these models are refined and concisely defined, user interface prototypes could be generated from these models.

![Diagram of artifacts created during the design process]

Figure 6.5: Artifacts created during the design process

Finally, user interface designers can design the user interface guided by the information in the models, combining it with their experience and user interface guidelines. We envision that most of the model synchronization effort can be taken from the designer and done by a tool due to the way the models are defined in CUP 2.0. Especially when only high-level design is done in the models and a high-level declarative user interface description is generated by
the tool to be augmented by a stylesheet that reflects the work done by the designer.

6.3 Conclusion

This chapter presented tool for and an analysis of the high-level models presented in this part of the dissertation for the design of context-sensitive interactive applications. For all of these models UML profiles were realized using a UML 2.0 modeling tool that is aware of the UML metamodel and allows extensions to be specified, MagicDraw (version 11). Other tools that satisfy the requirements mentioned in section 6.1 can also be used. A more integrated specification of models from the software engineering world and model-based design of (context-sensitive) user interfaces using one meta-model can thus be realized. Section 6.2.2 showed how all models can be used together in a model-driven approach and provides some hints on how they could be integrated in currently used software engineering approaches.

The first model that was introduced was an extension of the ConcurTaskTrees notation [Paternó 98], the Contextual ConcurTaskTrees. It introduced contextual tasks; tasks that affect the context in which an interaction takes place (for the exact definition, see section 3.4 on page 3.4 and following) and extended the support for multiple platforms to multiple contexts of use. A UML profile for the class diagram allows the specification of the Contextual ConcurTaskTrees within a UML editor with only minor deviations from the original specification.

Support for other models was introduced in two steps; first a course grained profile (see chapter 4) with a limited scope (e.g. no support for multiple contexts of use) was defined, followed by a more detailed profile with an extended scope. The first profile focused on the visibility of information and was optimized for optimal presentation of the data (e.g. abstract user interface components are nested visually into the containing user interface component). This profile delivered some important feedback from the software engineering community which was incorporated into a second version of the profile, described in chapter 5.

CUP 2.0, the second and more rigorously defined version of the UML profile, featured an increased scope and added some important capabilities: it introduced context-sensitivity into a flow-driven behavior specification, described in the system interaction model. It introduced an application model, which also allows to identify those concepts that are part of the context. Other models included in the profile are the context model, which is more flexible
6.3 Conclusion

or powerful than the context model in other approaches that are currently available \(^2\), and an abstract user interface model that establishes a clear link with the data specified in the application model. A last contribution of this profile is the introduction of a user interface deployment model, which allows the specification of how an abstract user interface should be deployed to a platform in a certain context of use.

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 usage of a UML profile has some drawbacks. For example, the difficulties in specification of the abstract user interface diagram in a way that is both intuitive and correct according to UML syntax and semantics is exemplified by the fact that two different diagrams were chosen for this model in the two versions of the profile.

Another example is the fact that the Contextual ConcurTaskTrees notation is specified in a structural instead of a behavioral diagram. Ideally the task model and the system interaction model should be able to use the same modeling structures for the specification of tasks. This would, however, have such a major impact on the meta-model that in the author’s opinion significant changes in the UML meta-model, rather than extensions of the meta-model are needed (see also section 9.2.3. In order to be able to reap the benefits of using a common meta-model until these changes are made, the usage of the most light-weight extension model seems appropriate because they allow the usage of a diversity of UML tools which are commercially available.

\(^2\)One can check this by comparing figure 6.3 with those in section 2.5.
Part III

SpIeLan: A Domain-Specific Language for Modeling Staged Participatory Multimedia Events
Chapter 7

SpIeLan: a Domain Specific Modeling Language for Presence-Aware Multi-User Applications

7.1 Introduction

The previous chapters of this dissertation discussed models for a generic approach to model context-sensitive user interfaces. The fact they all these models can be expressed using appropriate editors for the Unified Modeling Language [OMG 04] (UML) can help them integrate into software engineering methodologies such as those that were discussed in section 2.6. UML-based models and the transformation-driven approach discussed in section 6.2.2 are however not appropriate to all types of development of context-sensitive interactive systems. This part of the dissertation will discuss the models used in another type of model-based development: development using domain-specific models.

Domain-specific models are models that are designed with one specific usage in mind. They are often specific to a vertical market or even to (a division of) a company. They often use notations that are specific for this vertical market or company and are mainly used to create complete interactive applications. The chapters in part III of the dissertation discuss a domain-specific modeling language that is used to design staged participatory multimedia events (SPME) for Participation television.

These SPME are expected to be the next step in the broadcast industry, which adds a participation aspect to the television experience. This participation experience is one step further from the interactive television as it is
Participation television will be the kind of interactive television with the highest degree of public involvement. The watchers are no longer constrained to merely passive viewing or anonymous participation, but they can become part of the show if they want to. This can be accomplished by not only using a remote control and keyboard for interaction, but by adding devices such as microphones and webcams to create input for the show and by using cross-medial aspects; e.g. users can view or interact using their remote control and television set, their PC or even their mobile phone.

An important aspect of such events is that users, or participants of such television events are aware of the other participants of the show, furthermore it is important that the screen is adapted to the current role of the participant. Understanding this, means that this type of television event is a specific instance of a context-sensitive interactive application.

The creation of participation television shows is complex, not only regarding the social, creative and managerial aspects, but the software needed for those shows is becoming very complex and more similar to traditional software engineering projects compared to the production of traditional broadcast television shows. This chapter introduces a graphical modeling language, SpIeLan (short for SPme InterfacE LANguage), that uses a high-level of abstraction for designing participatory multimedia events.

The rest of this chapter is started by the definition of Staged Participatory MultiMedia Events. After a discussion of related work, section 7.4 provides an overview of SpIeLan, the proposed modeling language. The following sections discuss each of the three model types that can be specified using SpIeLan. At the end of the chapter a conclusion is provided.

A detailed discussion of tool support as well as the evaluation of SpIeLan are presented in chapter 8.

### 7.2 Staged Participatory Multimedia Events

Broadband end users currently witness an evolution towards ubiquitous entertainment being delivered over their High-Speed Internet (HSI) line. As such the broadband pipe is used to deliver complete experiences to the home. Unfortunately, this enables the home-based users only to consume more professional content (Hollywood movies on demand) and to get some low level interaction with the broadcast television: time shifted viewing, voting, etc.
The goal of Staged Participatory Multimedia Events (SPME) is to take the interaction one step further and to actively engage end-users and turn them into true participators, thus providing a stage for users to participate in challenging interactive television applications that do not exist today. Several devices like microphones and webcams will be used in these formats to enable the different participators with true interactivity. In the future, this will lead to television shows with thousands of active participants, where complex software is needed to cope with these new formats. An example format is discussed in section 7.5.

An example of a SPME is shown in figure 7.1 using UML and some illustrative screenshots from a mock up (©Alcatel). It describes (part of) an interactive auction on television. The auction starts by an introduction by the auctioneer. After the seller has presented the item that is for sale, interested buyers can ask questions and place bids on the item. The involvement of the viewers is enhanced because all actively involved viewers can be shown through video-streams from e.g. a webcam. All viewers play a certain role in
the show; one is the auctioneer, one is the seller and the other viewers are possible buyers. All these users can participate in the show from behind their TV set as is illustrated by figure 7.2. The gray ovals represent a common location around a TV set.

![Figure 7.2: The different roles users participating in the auction SPME play and their locations](image)

In the remainder of this chapter we will use the auction to illustrate the usage of modeling language we created to design such SPME.

It is clear that creating such an interactive show requires establishing and creating a complicated software infrastructure. It is our intent to facilitate the creation of such shows by offering a set of models from which the necessary code can be generated. The generated code will use a set of pre-build networked components. The goals of the models should abstract away a lot of the details and complexity of such infrastructure. In our approach this abstraction is reached by using a set of models using different abstractions that can be considered to create different abstraction levels. The models at the highest level of abstraction do not have a direct relationship with the software infrastructure but relate to the structure of a SPME and the structure of the user interface that the participants of the SPME interact with on their television set. These high-level models can be used without the lower level details to enable early design evaluation using generated prototypes as will be discussed in chapter 8.
7.3 Related work

Multi-user interactive applications are heavily researched when these applications offer support for collaborative work. Which also involve many technical issues like synchronization, locking, floor control, etc. Boyle and Greenbergh [Boyle 05] support the design of multimedia groupware by providing a toolkit that allows to quickly prototype multimedia groupware applications. By not taking into account scalability issues and providing high-level access to often used features of multimedia groupware, they can create functional prototypes of such applications with a few lines of code.

Our work is more related to the design of the interactivity of the distributed multimedia application. van der Veer and van Welie [van Welie 99] provide an overview of more task-based notations and whether they support some features necessary for their goal; the analysis, design and evaluation of groupware. They concluded that no single notation was satisfactory and proposed a process, DUTCH, that uses multiple notations, including a task tree, a flow-diagram and an extension of UAN [Hartson 90], integrated through a shared ontology.

At the moment several tools (from companies such as Aircode, Alticast, Sofia, iTVBox and Cardinal) are commercially available for the development of interactive television applications. These tools provide a graphical environment enabling a non-technical user to easily create simple iDTV software or websites. They require no technical knowledge like Java, MHP or DVB-HTML [DVB 03] from the designer and thus ease the creation of iDTV. Most of them also offer an emulator which enables the author to preview the result of his work on his own PC, rather than deploying his output to a set-top box.

These environments are however too limited for the development of participation television. They are mostly centered to designing the graphical layout of various pages and the addition of some common iDTV modules and components. There is no support to add the interaction components needed for the participation of the viewer in a television show.

A number of tools have been created that allow early prototyping based on models. Canonsketch [Campos 05] is a tool that provides synchronized models at different levels; at the highest level, a UML class diagram is used to represent the structure of a single dialogue using the notation proposed in [Nunes 00]. The Canonical Abstract Prototypes notation (CAP) [Constantine 03a] allows to describe the functionality offered by a user interface at an abstract level using a limited set of icons that are linked to specific areas of the screen. At the most concrete level, a preview using HTML is provided.
Several model-based approaches for the development of user interfaces are taking into account some form of context-sensitiveness as discussed in section 2.5. The TERESA-tool [Paternò 01, Mori 04] allows the semi-automatic creation of multi-platform user interfaces (mostly targeted to the web) starting from a task model. Another task-based method is described by Clerckx et al. [Clerckx 05], who provide tool support that enables derivation of a context-sensitive dialog model from a context-aware task-model. In combination with a screen model, the tool is able to generate context-aware prototypes. Section 8.2.2 provides a more detailed discussion of multi-user support of notations based on the ConcurTaskTrees [Paternò 98].

Although little specific research has been performed to model multi-user (context-sensitive) interactive applications, UML activity diagrams already offer support for modeling multi-user applications. Section 8.2.2 provides a discussion of an example of a multi-user interactive application using UML.

Figure 7.3: Screenshot of RCX Code within the programming environment, courtesy of http://www.robocamp.pl
Another area of related work is end user programming. End users within this dissertation are people who are not professional software engineers, computer scientists or programmers. Different approaches have been proposed with different target applications in mind. One approach is the use of visual languages. An example of visual programming is RCX Code for programming the Lego MindStorms [Lego 06]. It is used to program robots constructed with a set of Lego building blocks. RCX Code is an event-driven, visual language. Events can be caused by sensors attached to the robot. The language has control flow structures to specify conditional or repetitive behavior. A screenshot of the tool is shown in figure 7.3. The program is build by linking Lego-like building blocks with different colors and associated behavior.

Other tools for end-user programming use different approaches, such as programming by example, simpler textual languages, or better programming environments as discussed in [Myers 06], which provides an excellent overview of the work done in the area of end-user programming.

7.4 Overview of SpIeLan

The SpIeLan consists of three different model types: the scenario model, the screen model and the template model. Each of these three models was made with a specific usage in mind. Before discussing the three models, we introduce two definitions: the definition of a template (definition 7.1) and the definition of a scene (definition 7.2) since these are the central concepts in the modeling language.

![Diagram of models supported by the SpIeLan](image)

Figure 7.4: Overview of the models supported by the SpIeLan

**Definition 7.1** A template is a reusable behavioral artifact containing a flow of actions in which a specified number of participants having different roles can be involved. A template can receive data as input and produce output data.
Definition 7.2 A scene is an instance of a template. The value of all input parameters of a scene need to be specified either explicitly in the model or implicitly through derivation from output parameters of another scene in the scenario diagram.

The scenario model describes the overall structure of a staged participatory multimedia event. It defines a sequence of scenes and how users with different roles are actively involved in these scenes. Each scene is an instance of a certain template as stated in definition 7.2. Each template (see definition 7.1) is complete defined by two models: a screen model and a template model. Figure 7.4 illustrates the relation between the models.

The screen model describes the layout of the screen for all different roles that are involved in an instance of the template, a scene. Most of the time, most of the screen will contain the same information for all users, but some parts of the screen will differ for users with different roles. The screen model allows to compactly specify this information.

The template model describes the interaction of the users through the user interface with the functionality offered by interaction components on servers. This model also defines when a user with a certain role can perform an action within a scene. When not all components are visualized from the beginning to the end in the scene, the moment when the components appear is also specified in the template model.

The creator of a show is considered to be an end user. She will create a show using pre-build templates, possibly based on an earlier show. When not all desired templates are available, she is not expected to create a complete template. Creation of templates is expected to be done by software engineers, programmers or at least people with a technical background because it requires knowledge of the underlying software infrastructure.

The envisioned design process is shown in figure 7.5. An end user creates a scenario or adapts an existing one and then adapts or creates all relevant screen models. When new screen models are created, new templates will be necessary. These templates can be created by users that understand the infrastructure and the available components. When all (abstract) screen models are ready, an abstract prototype can be generated (as discussed in section 8.1.2). This abstract prototype can be used to perform a first evaluation. When this evaluation is satisfactory, the screen models can be refined and more concrete information can be added to them. Based on this information new, more concrete prototypes can be generated. When these prototypes are satisfactory, a show (or a specification that can be used to run the show) can be created.
7.5 Scenario Model

7.5.1 Requirements

The scenario model should enable someone to effectively create the flow of a SPME and communicate this design. To reach this goal, some important characteristics were determined:

- The specification of the scenario should allow clear specification of the roles of the participants that are involved in a certain scenario. These roles can change during the execution of a scenario, so the scenario should support this.

- Interactivity is a very important part of SPMEs so whenever participants...
of the show are able to interact this should be clearly indicated.

- Multimedia and awareness of other users is an important part of the experience in a SPME. Whenever a user provides live or recorded media input this should be clearly indicated.

- The creator of a show, an end user, should be able to use generic templates, while he should be able to clearly indicate the function this template instance plays in the current scenario.

### 7.5.2 Modeling Constructs

The design of the diagram is based upon the graphical language used to program Lego Mindstorms. The language also features a horizontal bar, the *heading*, at the top that serves as a starting point for one or more parallel flows. In contrast to Lego language, SpIeLan does not support parallel flows. The flow of a program is specified from top to bottom. Whereas in LEGO mindstorms each block in the diagram corresponds to one building block, in SpIeLan, each block corresponds to a scene of the scenario. Loops and conditional behavior are specified using two separate blocks as can be seen in figure 7.6(b) and figure 7.6(c). Figure 7.7 shows a scenario model for the auction SPME discussed section 7.5.3. The different parts of the scenario-model are marked in the figure and shortly discussed in the remainder of this section.

**Heading** The heading (see figure 7.6(a)) gives some generic information about the scenario specified in that diagram. It contains the name of the scenario as well as all roles that are required to start the execution of the scenario.

**Scenes** A scene is depicted as shown in figure 7.6(d). The scene representation consists of three main areas. The upper part specifies the name of the template (optionally followed by a more precise name of the scene), while the middle part specifies the data flowing into the templates (center) and out of the template (right). The lower part shows the roles that are actively involved in the template. A role is actively involved when users having that role can cause an event that triggers or ends part of the behavior specified in a template or can give streamed input to the modeled system, such as live-video through a webcam.

**Roles** The graphical representation of each role (see figure 7.6(e)) in the diagram shows role-specific information in addition to the role name. The
minimum and maximum number of participants having that role are shown between parentheses while the actions that they can perform as well as the required input devices are represented using icons. The icons for the events correspond to the tools of the Canonical Abstract Prototype notation [Constantine 03a], which is discussed in appendix C. The most commonly used actions in SPME (based on the experience of our the Alcatel R&D team for interactive television) are generic actions (they trigger some functionality of the application), select actions (e.g. select an answer for a multiple choice question or vote for a candidate in a contest) and modify actions (e.g. change the amount for a bid in an auction). The icons for the input devices are stylized representations of a camera 📹 and a microphone 🎤.

Two predefined role types can be used in scenario models: all, representing all participants of a SPME, other, all participants except those having roles explicitly specified in the template.

**Data** The only information visible in the diagram about the parameters and the results of a template, are its name and an indication of the data type using the representation shown in figure 7.6(g). A stick-figure is
shown for role-related data (e.g. when the role of a participant changes within a template), a stylized camera for live streaming video, a stylized microphone for live streaming sound, and a stylized clapper board for media recorded in advance. For all other types of parameters no type indication is provided.

7.5.3 Example

An example of such specification for the auction on television in figure 7.1, is shown in figure 7.7.

The auction SPME could start when a regular auction is being broadcast and at least one of the viewers has offered an item for sale. When enough interested viewers are registered (at least two in the specification in figure 7.7, an auctioneer can decide to start the auction. After a short welcome message, he introduces the seller to the interested viewers. The seller then has the opportunity to promote the item, after which the auction starts. Any of the registered viewers (who all have the role Buyer) can then make a bid on the item or ask questions to its seller. When a satisfactory price is reached, the auction is concluded.

7.6 Screen Model

7.6.1 Requirements

For the screen model, the following requirements were gathered:

- The specification should allow specification of the structure of the user interface independent of the physical devices and the physical interactions that participants will use. E.g. one should not be limited to wide-screen television sets and the model should not specify whether color buttons or navigation keys should be used to trigger functionality.

- Screen organisation for each of the participants during a scenario is similar although it can differ for each role.

- Multimedia and awareness of other users is an important part of the experience in a SPME. It should be clearly indicated whether a certain part of the screen contains video or not and whether some participant is represented on a certain part of the screen.

- The language should be useable by end users.
Figure 7.7: The scenario model for the auction SPME
Based on these requirements, the decision was made to use the Canonical Abstract Prototypes (CAP) notation as a starting point for the interaction, but to adapt it to the specific needs for the screen model. The following section provides a short introduction to the CAP. It is followed by a more detailed discussion of the resulting notation.

### 7.6.2 Modeling Constructs

![Figure 7.8: A screen model for the template Bidding or StartQuestion in figure 7.7](image)

Our notation is shown in figure 7.8. It has heading which lists the basic features of the template: its name, its input and output values and all roles that can participate in an instance of the template. Abstract symbols for the user interface components that are visible to participants with a certain role
Table 7.1: Most common user interface components

<table>
<thead>
<tr>
<th>Icon</th>
<th>Name</th>
<th>Example/explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>![ ]</td>
<td>Element</td>
<td>a video, an image, text</td>
</tr>
<tr>
<td>![ ]</td>
<td>Collection</td>
<td>a set of smileys</td>
</tr>
<tr>
<td>![ ]</td>
<td>Notification</td>
<td>a confirmation message</td>
</tr>
<tr>
<td>![ ]</td>
<td>Participant element</td>
<td>a buddy icon or video of a participant</td>
</tr>
<tr>
<td>![ ]</td>
<td>Active Participant collection</td>
<td>a set of buddy icons or videos of participants</td>
</tr>
</tbody>
</table>

are represented in the corresponding role representation in the heading, just as the relevant input sources and actions.

The user interface components that are visible to all participants of a show are not repeated in each role representation, but are grouped in the representation of a special role All. Each user interface component is represented using the icons of the Canonical Abstract Prototypes notation, with a number or letter and an informal description next to it. The most commonly used user interface controls, actions excluded, are shown in table 7.1. We created the last two icons in the table to identify user interface components that show a participant (participant element ![ ]) or a collection of participants of which zero or more can be emphasized (active participant collection ![ ]). These icons were created to identify components that show presence of other participants in a staged participatory multimedia event.

The bottom part of the screen model, shows an abstract representation of a television screen. The area of the television screen that should be visible on all televisions is indicated using a dashed line. All icons for user interface components are allocated to a region of the screen by placing them in a rectangle with a semi-transparent background. Multiple user interface components can be allocated to one region of the screen. When one or more user interface components are visible for a single participant, these components will be shown one at a time to the user in the generated application. A participant will be able to navigate between these user interface controls. The link between the allocation of a user interface component on the screen in the bottom part of the model and the participants with a certain role in the top part of the model are visualized using two mechanisms: color and symbols.

The usage of color allows the users of the diagram to quickly see which user interface components are visible for a participant with a certain role (see section 8.2.3) when color information is available. When no color is available, for example in a black-and-white hard copy of the model, the characters provide
a back-up mechanism to visualize this link.

Because multimedia assets are an important aspect of a SPME, icons referring to the origin of multimedia materials are added to the icons in CAP-notation. The sources can be either live multimedia streams or recorded multimedia. The sources are linked to users with a certain role with means identical to the ones used to link user interface components with participants with a certain role.

### 7.6.3 Example

![Chat specification - proposed notation](image)

Figure 7.9: Chat specification - proposed notation

Figure 7.9 shows the screen model that corresponds to the template model, specified in figure 7.12. All user interface components that are used in the template diagram to specify which data they contain, are allocated to a certain
region on the screen. Notice that a single region can contain more than one symbol for a user interface component. This enables us to show the user interface specification for multiple roles at different moments in time in a single diagram.

The user interface components are shown on the screen whenever an interaction could retrieve data from it or push data to it. In the chat example, this would result in the appearance of the user interface components which allow entering login and password from the start of the template for users with the role LoggedIn and the contactlist (user interface component \(1\)) would be visible from the moment at least one user is logged in. The designer can however override this behavior if desired.

7.7 Template Model

7.7.1 Requirements

The template model specifies the interaction between participants of a staged participatory multimedia event and the system as well as the temporal relations between these interactions. The requirements we set for the language based on discussions with domain experts are as follows:

- The language should support interaction of multiple users with a single system. Each of the users has one role, but this role can change at runtime. A role change is established by the system. A user’s role determines the user interface that is displayed to the user and the interactions that are possible.

- The usage of multimedia and awareness of other users are very important for the establishment of staged participatory multimedia events and thus they should be represented in the language.

- A template and the interactions specified in it should be useable in a number of contexts, so a certain degree of abstraction is desired. E.g. it should not specify which concrete hardware features (such as for example color keys on a remote control) should be used to trigger some interaction, however the fact that interaction should take place had to be shown clearly.

- It should be possible to map the specification onto a framework of server-side interaction components which uses an asynchronous messaging system for establishing interaction.
• The integration with the other models should not be too confusing.

• The target users of the template model are technical people such as programmers, software engineers, etc.

7.7.2 Modeling Constructs

The template model consists of two main areas, a header and the main content. The header, shown in figure 7.10(a), specifies the name of the template, the data flowing in (parameters) and flowing out (results) of the template and all roles that are involved in the template. For each role, all parts of the user interface that are visible to that user are specified as well as the media-sources they provide. A special role, All, exists to avoid unnecessary double specification of user interface elements that are visible to all users.

The second area describes the functionality of interaction components that is used in the templates, when this functionality is used and how it is triggered by the SPME participants through actions in the user interface on their TV set. The structure of this part is based on the UML activity diagram. The semantics of the constructs shown in figure 7.10(d) is the same as their counterparts in UML. From left to right in the figure these UML elements are InitialNode, ActivityFinalNode, ForkNode/JoinNode\(^1\) and ControlFlow.

Invocation of functionality is described using two special constructs instead of actions as in UML: Interaction and MultiUserInteraction. Examples of these constructs are shown respectively in figure 7.10(b) and figure 7.10(c). Both contain four sections from top to bottom:

• The name of the function of an interaction component that is triggered

• The name of the interaction component offering the function

• The parameters of the function and its results.

• Events that can start (In:Start) or stop (In:End) the execution of the function, or that are sent during the execution of the function (Out).

Figure 7.11 shows the generalized server-side behavior specified by these constructs using UML syntax. The semantics of an Interaction is shown in figure 7.11(c). It specifies that the execution of the server-side functionality is started when an event event1, specified in the section In:Start, is received by an interaction component. This interaction component then starts executing

\(^1\)ForkNode and JoinNode have the same graphical representation.
the specified functionality Function. In the UML diagrams, the actual execution of the server is represented by the unlabeled actions. This might cause sending an event, specified in the section Out to another interaction component. The execution of Function is stopped when an event event2, specified in the section In:End is received. A cleanup action can still be executed, which can be used to send the results to the participant’s television set. The specification of events in the sections In:Start, In:End and Out is entirely optional. In this case, the Interaction is started when the previous one ended and it stops when Function has finished execution.

The semantics of example Interaction in figure 7.10(b) is thus as follows. The function Login of interaction component Authorization is acti-
Figure 7.11: Semantics of the SpIeLan Interaction and MultiUserInteraction using UML 2.0 syntax. Corresponding events in figure (a) and (c), and (b) and (d) have corresponding labels.

vated when a participant executes action a. This function has two parameters, Login and Password, whose values are provided respectively by the user interface components 5 and 6. It has one result add which is displayed in user interface component 1. When the execution of Login is finished and the results are displayed in the participants’ user interfaces, the Interaction has
7.7 Template Model

finished execution. The execution of the Interaction also finishes when event d is received.

The semantics of a MultiUserInteraction is similar to that of an Interaction, but differs in the fact that it can be started and execute multiple times before completion. This means that at least one event needs to be specified in the section In:Start. When this event is received by an interaction component, it starts executing the specified functionality. In the mean time, however, it can keep on receiving the specified event. The execution of a MultiUserInteraction ends, when no more events can be received because the maximum amount of incoming events has been reached, specified in the section In:End is received or no more events can be received because all possible event sources are exhausted. This can happen when all participants of the roles that can cause the events can only send the event a limited number of times and they have all reached that limit.

The MultiUserInteraction in figure 7.10(c) specifies that the function ProcessMessage of interaction component MessageServer is triggered when action c is performed by a participant. The function takes the data of user interface component 3 as value for its parameter Message and the result is shown in user interface component 2.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗</td>
<td>At start of execution</td>
</tr>
<tr>
<td>✗</td>
<td>During execution</td>
</tr>
<tr>
<td>✗</td>
<td>At end of execution</td>
</tr>
</tbody>
</table>

Table 7.2: Interaction component events

Functionality can be activated or stopped by other interaction components through the specification of interaction component events or events generated by the users (which are discussed in the following section). The interaction component events are displayed as sand glasses with different time indications depending on the moment they are sent as can be seen in table 7.2. Note that interaction components can only sent events to interaction components that are currently executing or waiting for an event to be executed. Events sent to interaction components that are not in one of these situations are ignored. Exchange of data can take place between all functions specified within a single diagram, taking into account the control flow that is specified. The control flow will be explained into more detail in section 7.7.3.

When desired, user interface components that are used as data sources or display results from certain functionality can automatically appear on or
Table 7.3: Visibility user interface components specified as source or target of data in Interactions and MultiUserInteractions

disappear off a participant’s screen during a scene based on there usage as data sources or result displays. The resulting behavior is summarized in table 7.3. This behavior can be used to create notifications that appear after an action of a participant and disappear again after a second action.

7.7.3 Example

Figure 7.12 shows an example of an interaction specification of a simple chat application. The heading shows the name of the template, Chatting in addition to the roles that are relevant: LoggedIn, LoggedOut and Director. No parameters or results are specified. Below the heading, one can see the control flow of the template.

The control flow can be explained as a token that starts at the start node. One token than flows through the edge to the following node in the diagram. In figure 7.12 this is a fork node. In a fork node, the token splits in one token per outgoing edge. When all local preconditions are met (all specified events are received by the interaction component), the interaction component starts executing the specified behavior. In the example, this means that once a user has requested to log in with a valid login and password, that user is logged into the system and the contact-lists of all users get updated. In parallel, the MessageServer waits until it receives an event e from the Director that will start the shut down process. When this process starts, the event d is sent which stops all processing of messages, logs out all logged in users and stops the ability of users that are not logged in to use the chat service.

Once one user is logged into the system, a token leaves the Interaction Login, splits at the fork node and three tokens flow to the specified MultiUserInteractions that enable sending and receiving of messages, the login of other users and the logged in users to log out. Once all users have logged
out, the execution of the MultiUserInteraction Logout stops. This triggers the event \( f \) which also stops the other MultiUserInteractions. Once these have all successfully stopped execution, tokens have flown over all incoming edges of the join node, starting the flow of a token over the outgoing edge, which again starts the Interaction Login when a valid login and password are specified by a user (in user interface components 5 and 6).

The interaction stops when the final node is reached (a circle with a black dot in it). In the example in figure 7.12 the execution ends when the Director has shut down the MessageServer.
7.8 Conclusion

This chapter introduced a domain-specific modeling language for the specification of staged participatory multimedia events (SPME). The modeling language uses three models that complement each other. The scenario model describes the overall structure of a SPME using scenes. These scenes are instances of templates which are described using two models: the screen model and the template model. The screen model describes which user interface controls are visible to which participants of a SPME, while the template model describes the interaction between the user, his or her television set and server components that provide the functionality.

The proposed modeling language features a compact notation for this type of multimedia multi-user and multi-device interactive application. The use of a domain-specific language for the specification of SPME (instead of a generic language such as a UML) provides the benefit that special attention can be paid to concepts which are important to this domain, such as multimedia, both live and recorded, representation of the participants on the screen and the interaction of participants with different roles. The usage of abstractions that should be natural to the final users of the language (such as scenes) and usage of proved abstractions such as the Canonical Abstract Prototypes notation [Constantine 03a], could make it easier to specify such systems.
Chapter 8

Tool Support and Evaluation of the Domain Specific Language SpIeLan

This chapter discusses tool support and presents an evaluation of SpIeLan, the domain-specific modeling language introduced in chapter 7. The goal of this domain-specific language is to enable the generation of prototypes at various levels of abstraction and fidelity. Section 8.1.2 discusses how high-level prototypes can be generated from two of the three models: the scenario model and the screen model. This is the first prototype to be generated following the proposed design process shown in figure 7.5.

The second part of this chapter presents the evaluation that was performed for SpIeLan. The evaluation consists of four parts. The first part discusses the models of SpIeLan according to the model elements for context-sensitive user interfaces, proposed in section 2.4. Section 8.2.2 compares the specification of templates in SpIeLan with the specifications presented in part II. After a theoretical evaluation of SpIeLan using the Cognitive Dimensions framework [Green 96], the results of a usability test are presented in section 8.2.3. Finally, conclusions are drawn about SpIeLan.

8.1 Tool Support

8.1.1 Introduction

The envisioned development process for staged participatory multimedia events (SPME) explicitly includes the generation of prototypes at various levels of
abstraction from models expressed using SpIeLan. In order to reach this goal we built a tool that allows the creation of models using a Visio [Microsoft 06] plug-in and the automatic generation of abstract interactive prototypes based on these models. Figure 8.1 shows a screenshot of this tool. At the left side of the tool, the Visio toolbar and a SpIeLan diagram is visible. The right side of the tool provides access to some tools among which the most important one allows completely automatic generation of the interactive prototypes as discussed in section 8.1.2.

It is however clear that this tool is not the type of tool that would be offered to end users, instead it is intended to test the specification and to proof that prototype generation from the proposed models is possible. Such a tool would have a completely different interface. The user test discussed in section 8.2.3 already resulted in some requirements for a tool.
8.1 Tool Support

8.1.2 Generation of Prototypes

This section discusses how the models of SpIeLan can be used to generate high-level prototypes. Before introducing the algorithms used to generate the prototypes the specification of the prototypes themselves is introduced.

Specification of the prototypes

The dynamic abstract prototypes are expressed using a combination of XHTML [Pemberton 02], XForms [Dubinko 03] and CSS [Bos 06]. XHTML merely serves as a container for embedding the dynamic abstract prototype. XForms and CSS are respectively used for expressing the structure of the dynamic abstract prototype and the styling and positioning of the abstract components. This combination was chosen because the tools to display these specifications are freely available and the specifications are relatively easy to read. Furthermore, the style and layout, the structure of the show, the user interface and the runtime data are all cleanly separated. The choice for XForms is also motivated by the fact that it is designed to be embedded in another XML-based language and is completely declarative (including event handling). This enables reuse for more concrete prototypes, for which the XForms-structure could be largely reused in for example a SMIL [Koivisto 05] document. X-Smiles [X-Smiles.org 06] is being ported to the MHP platform [DVB 03] and will be able to show content expressed using XForms and SMIL.

An example of a prototype corresponding with the Bidding template in the scenario in figure 7.7 and the screen layout in figure 8.2 is shown in figure 8.3(a). It shows a typical screen during prototype simulation. The upper part contains controls for navigating through the abstract prototype. At the top left one can select a participant with a certain role. In this case the participant’s name is Mieke and has the role buyer. At the next row, the values of all parameters of the current template are shown. The next line shows input fields for the corresponding output, followed by triggers for navigating through the abstract prototype, including a trigger for restarting a simulation and triggers for all transitions to other scenes that are possible in the active context (selected participant and parameter values). The last line of the top part displays the current template and scene. The rest of the screen shows the actual prototype using the CAP notation. User interface components in the abstract prototype can show a tooltip when hovering over them giving concrete information about its function. The tooltip for the abstract component Ask Question in figure 8.3(a) shows a button, Question, that triggers a transition to the corresponding scene.
The remainder of this section provides more detail about how XForms and CSS are combined to create the dynamic abstract prototypes. The overall structure of a document describing a prototype is shown in figure 8.3(b). The document consists of three major parts: (1) simulation related data, including the participants of the simulated show, scenario structure and the applicable constraints, (2) the prototype manipulation controls and (3) the description of the user interfaces associated with the templates.

**Templates and scenes** The template structure is coded into XForms instances. Listing 8.1 shows the document type definition (DTD) [consortium 01b] for the XForms instances. It lists all templates in the same order as they appear in the scenario model described in section 7.4. Each template has a name and all corresponding scenes appearing in the scenario are described in nested tags. Each scene also has a name and a next scene specification. Notice that
8.1 Tool Support

(a) An example prototype

(b) The document structure of a XForms prototype

Figure 8.3: Prototype example and its structure
the elements *instances* and *next* are used because the XForms processor requires an element to only contain one type of sub-element in order to iterate over them.

**Listing 8.1: DTD for template and scenes structure**

```xml
<!ELEMENT scenario (template+)>  
<!ATTLIST scenario name CDATA #REQUIRED>  
<!ELEMENT template (instances)>  
<!ATTLIST template name CDATA #REQUIRED>  
<!ELEMENT instances (inst+)>  
<!ATTLIST inst name CDATA #REQUIRED>  
<!ELEMENT next (option+)> <!ELEMENT option EMPTY>  
<!ATTLIST option templ CDATA #REQUIRED>  
<!ATTLIST option inst CDATA #REQUIRED>  
<!ATTLIST option conditional CDATA #REQUIRED>  
<!ELEMENT params (param+)>  
<!ATTLIST param (#PCDATA)>  
<!ATTLIST param name CDATA #REQUIRED>  
<!ATTLIST param input (true|false) #REQUIRED>  
```

Choices and iterations are not directly coded into the templates although each instance can have multiple following templates, instead all possible next templates following a specific scene are mentioned. The template *Bidding* in figure 7.7, for example, can have both *Buyer Question* and *End Bidding* as next scenes. The figure however only shows *Question* as next scene, because no satisfactory bidding is reached yet. Note that also *Bidding* is not listed as an option because a navigation element to the currently active page can be confusing. Furthermore, this is a prototype that has no link to program logic. When the current user in the simulation is no buyer, none of the next options would be shown, because only buyers can trigger a transition to another scene. A warning (on black background) is displayed whenever some transitions could be hidden due to unsatisfied constraints (see figure 8.3(a)).

**Bind expressions** This section of the document contains mainly bind-tags that indicate relevancy of navigation controls (the “next triggers” for navigating to other scenes), or user interface components that are only relevant for a certain role. The generation of bind-expressions for the “next-triggers” results in a number of bind tags for unconditional transitions that equals the maximum number of unconditional transitions for a single scene and in one
8.1 Tool Support

bind tag for each conditional transition. Additional bind expressions are provided to ensure that input and output values for scenes are always displayed correctly.

**Simulation related information** All simulation-related data, is also encoded using an XForms instance. The show related information contains all runtime information about the show and a list of participants (with name, role and other related data such as media streams). The runtime information includes the information about the currently active participant, the currently active scene and tags that can be referenced to by controls that are shown conditionally (such as the possible transitions to another template). Among these tags there is one tag for each bind related to a “next trigger”. The relevance, combined with a CSS rule indicating that disabled XML-elements should not be displayed, allows hiding of all irrelevant items.

**User interface specification** The user interface specification is entirely contained within one XForms group (from now on referred to as group), representing the screen. This group contains a set of groups, one for each template. These groups are shown one at a time, depending on the currently active scene as described in the XForms model.

Each group contains XForms controls for each user interface component in the screen model. When controls are only visible to a certain role, they are only made relevant to this role, and consequently only shown to the relevant users. Additional information is specified in an XForms hint. This causes the information to be shown when a user hovers over the controls in the CAP-notation. The FormsPlayer plug-in for Internet Explorer allows embedding XForms controls in hints (displayed in most browsers as tooltips) figure 8.3(a). This enables using XForms output tags, combined with appropriate CSS-rules to show the CAP-notation of the component in the HTML-page and to show low-level interface components in tooltips for establishing real user interaction. In this way, the abstraction can be used to spark creativity, while keeping the interactivity of low-level controls.

**Generating Dynamic Abstract Prototypes**

The prototypes can be automatically generated from the models specified in section 7.4. We will shortly discuss the main aspects of this algorithm in this section: the generation of the templates and template instances section in the
Templates and template instances  For each scene in the scenario, the template is added. When a scene is followed by a choice-construct, all possible next scenes are added as options to the list of scenes that can follow the current scene except when the next scene is the current scene. The generation of the scenes is illustrated in figure 8.4 for the template bidding of the scenario in figure 7.7.

Participants  The initial list of participants is generated based on the roles present in the scenario heading, while the remaining participants are generated based upon the roles present in the screen models that are linked to the templates that are used in the scenario model. In this case, only roles that can add participants are considered. This means that the following role-types cannot cause the creation of new participants: roles that are already represented in the list, the meta-role all, and roles that are created during the scenario (i.e. they are mentioned as output of one or more scenes.) The generation
8.2 Evaluation

8.2.1 Model Coverage

The coverage of the models that are part of SpIeLan using the model elements introduced in section 2.4 is shown in figure 8.7. It is immediately clear from this figure that SpIeLan does not fit into the standard model-based way of thinking; a lot of model elements appear in all diagrams. Since the mapping of the model elements is not as straight forward as in the other approaches, table 8.1 provides an overview of the mappings. The coverage of the models shows that the context sensitivity of the SPME is very specific to awareness of users. In the current version, the other aspects of context are not represented in the models. Future versions of SpIeLan, might target other platforms than television, such as PCs or mobile phones. One part of the context sensitivity is not as apparent in this overview, namely the extended CAP controls which
Figure 8.6: Generation of the participants and user interface controls: (1) generation of media links for participants, (2) generation of repeated elements which contain media from participants with role buyer (3), and (4) trigger that is only visible to buyers and causes a transition (5)
8.2 Evaluation

8.2.2 Comparison with CUP and Contextual ConcurTaskTrees

The following sections compare the specification of a chat application using SpIeLan’s screen and template model with a specification using the Contextual ConcurTaskTrees and CUP. The comparison is limited to the specification of the interaction of the user with the server-side behavior and the specification of the user interface because these are the two areas that were important for the specification of SPME.

SpIeLan

Figure 8.8 shows the screen and the template model for a chat application. A complete description of what the contents of these models is given in section 7.6.3 and section 7.7.3. A short summary is provided here for convenience.

This chat application has a contact list, which is visible to all users of the application. Once a single user has signed in, he can start sending messages. Meanwhile other users can log in and once logged in can also start sending messages. Users that are logged in can log out again. Once they have done
### Model Elements vs. Elements of SpIeLan

<table>
<thead>
<tr>
<th>Model Elements</th>
<th>Elements of SpIeLan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>CAP tool, scene</td>
</tr>
<tr>
<td>Temporal</td>
<td>connections between scenes and between Interaction and MultiUserInteraction</td>
</tr>
<tr>
<td>CIO</td>
<td>CAP and extended CAP controls</td>
</tr>
<tr>
<td>CIO group</td>
<td>CAP groups</td>
</tr>
<tr>
<td>UI Relation</td>
<td>containment and position of CAP</td>
</tr>
<tr>
<td>Class</td>
<td>interaction component</td>
</tr>
<tr>
<td>Operation</td>
<td>Name of Interaction and MultiUserInteraction</td>
</tr>
<tr>
<td>User</td>
<td>Role</td>
</tr>
</tbody>
</table>

Table 8.1: Mapping of model elements

This, they can no longer send messages, nor see sent messages. An operator can shut down the service at all times.

The notation has a rather high viscosity since the execution of a single task can require several actions spanning multiple diagrams. Tool support can however heavily reduce the number of required actions. The high viscosity was also clearly shown during the usability test discussed in section 8.2.3, examples of tasks suffering high viscosity are discussed in that section.

When designing the language, much attention was paid to visibility, which caused some information to be present in several diagrams, causing the high viscosity for some tasks. Not all information is specified completely in each diagram. The names of actions, for example, are only specified in the screen model, while the symbols for these actions are also present in the scenario model and the template model. This means that a tool supporting the language should allow juxtaposing the different models. Something that was also noticed in the usability test.

A good understanding of a scenario or a template model depends on the names that are chosen for the scenes (in the scenario model) or the model elements. A secondary notation is therefore a welcome addition. Although the number of hidden dependencies between the different models is limited, there are some important hidden dependencies within, for example, the screen model: removing a user interface component from a screen model, may imply also removing its source and vice versa.
8.2 Evaluation

(a) Screen model for chat template

(b) Template model for chat template

Figure 8.8: SpIeLan models for a chat template. The full-size versions of these models are shown in figure 7.9 and figure 7.12

**Contextual ConcurTaskTrees**

The Contextual ConcurTaskTrees were introduced in chapter 3 to allow the specification of task models for context-sensitive interactive applications. Since its underlying specification the (cooperative) ConcurTaskTrees (see section 3.2) allows the specification of task models for multi-user applications, our extended notation can also be used for this purpose. This section discusses the task model for the example of a chat application discussed in sections 7.6.3 and 7.7.3.

Figure 8.9 and figure 8.10 show these diagrams, applied to the chat example. In this case, the number of roles has been limited to two: chatters and operators. The difference in the user interface between chatters that are logged in and those that are logged out, can be derived from the diagram and is therefore eliminated. Some other properties are, however, less visible in this diagram. An example of such property is what information is displayed to whom and which user interface components will be used to show the data. It is also not clear from the diagram what information is manipulated or passed around between different tasks. The use of temporal operators and their prece-
Figure 8.9: Cooperative part of Chat specification - Contextual (Cooperative)
ConcurTaskTrees
8.2 Evaluation

dence sometimes makes it difficult to construct correct specifications, although tool support for the notation helps to find and identify problems.

The use of three diagrams instead of one for specifying the behavior, raises the viscosity; the same information often has to be specified twice: once in the cooperative part and once in the role specific part. The specification however allows to specify behavior for each role that is not present in the overview diagram, which enables the creation of richer clients. This feature is however not useful in this situation because all information processing is done at server-side. Roles in CCTT are allocated to different diagrams which makes getting an overview in an application in which roles are dynamic more difficult.

The notation offers no obvious way to specify which interaction component is responsible for executing which behavior. This is natural because specification of such implementation related things is out of the scope of the
Collaborative ConcurTaskTrees, which is a cognitive model. The model, however, shows when context (in this case the availability of the chat contacts) is manipulated.

The notation has some hidden dependencies; the temporal relations between tasks in the different parts of the model should not conflict.

**CUP 2.0**

The System Interaction Model, presented in section 5.5, can also be used to specify the chat example. Figure 8.11(a) shows a possible specification. For each ≪userTask≫ or ≪interactionTask≫ in the diagram one can specify the user that executes the task using tagged values. Another possibility in activity diagrams in general is the usage of so called swimlanes. For the system interaction model this option was not chosen in order to be able to specify the source of an event for AcceptEventAction as the actor. Figure 8.11(a) shows two diagrams; one for specifying the behavior of the chat-client and one for the specification of the chat-server.

Two diagrams were chosen for this specification because the author was unable to clearly specify both the client behavior and the server behavior in a single diagram. These two diagrams were created from scratch and not by transforming the specification discussed in section 7.7.3 using the mapping shown in figure 7.11. The specification in UML however relies heavily on the usage of some new syntax for UML 2.0, such as the interruptable regions (the dashed rounded rectangles in the diagram), which stops all execution within that region when an event has been received (such as for example the server shutdown element in figure 8.11(a). A drawback of the specification in the figure is that the interaction between the client and the server is not directly clear from looking at the diagrams. To make this interaction a bit clearer, additional actions of the types SendSignalAction or AcceptEventAction were added to have pairs for the interaction between client and server. This pairing is not needed in the Contextual ConcurTaskTrees as demonstrated in section 8.2.2.

Furthermore, there is no directly visible relationship between the actions and the user interface components that use them. Although these user interface components have the same name in the abstract user interface specification as can be seen in figure 8.11(b). This user interface specification shows that Chatters are visualized and gives an indication about the structure of the user interface. Something that isn’t as clear from the abstract user interface specification is which users can see what part of the user interface. Note that this figure only shows the user interface for the chatter; the struc-
8.2 Evaluation

(a) Chat specification - System Interaction Model

(b) Chat specification - Abstract User Interface Model

Figure 8.11: Chat specification using UML 2.0 and CUP
ture of the Operator is trivial (one group component that contains one action component). Note that the stereotypes ≪uiData≫ and ≪uiOperation≫ on the attributes and operations are not shown in this diagram, which is entirely legal to do in UML. The fact that one relies on the naming of different components to establish relationships introduces a hidden dependency, which is not present in SpIeLan where these conventions are not necessary.

8.2.3 Usability Test

A domain-specific language, such as SpIeLan, must use a vocabulary that is easy to understand for the target public. One can only get an accurate image of how good a language is understood by performing a usability test. The conducted test purely evaluated the two models of the language aimed at end users and was conducted entirely on paper. It consisted of two parts: the first part evaluated how easy it was to understand this language while the second part evaluated how easy it was to make changes using the language. Appendix E offers an overview of the test which was conducted in Dutch. The explanatory texts, however, are not included in the appendix because of space constraints.

There were a total of nineteen participants, spread over three days. There were five women and fourteen men participating in the test, which had ages varying between 22 and 61. Five of these had no or almost no programming experience, half the people participating in the test did not consider programming (or designing) to be a major goal in their job and can thus be considered end users. One of the people without programming experience, also the oldest participant, failed to complete the test. He claimed that it was too difficult and he was too old to understand the language and or test. Because he already quit after answering one question, we will further discard his results in the test. By doing this, the oldest person whose results were evaluated was aged 45. Nine of the eighteen participants that completed the test performed the second part of the test using a think-aloud session which was videotaped. Seven participants were involved with a Participation TV project, three of them had no or very limited programming experience.

Table 8.2 gives an overview of how the users performed on the questions with objective results and the two adaptation tasks in the second part of the test. The two series of questions in the first part of the test were all multiple choice questions. A short description of the questions with objective answers can be found in table 8.2, followed by the percentage of test participants that chose the right answer. The percentages are given for four categories: all test
participants that completed the test (18), all participants that completed the
test and had no or very limited programming experience (4), participants with
programming experience and knowledge of the domain (4) and participants
with programming experience and no knowledge of the domain (10). It is
important to note that three of the four test participants without programming
experience had knowledge of the domain. No significant differences were noted
in their results.

Most users gave correct answers to all questions in part 1 of the test.

One remarkable result in the table can be (partly) explained by a problem
in the test. For example, the significant difference in results between the last
category and the rest regarding the question 3 (complete question in Dutch
see figure E.3) of the table can be explained by the fact that this question
was unclear to many participants and a wrong explanation was given to the
participants of the last category.

A remarkable result that cannot be explained through a deficiency in the
test is the result for question 24 (complete question in Dutch see figure E.5) in
the table, although the number of users is probably too low to generalize the
difference in results. It is remarkable that all non-programmers answered that
question correctly, while less than half of the test participants with program-
ning experience gave the correct answer. This question involved the inter-
pretation of the symbol used for indicating repetition (\texttt{>>>}) in the Canonical
Abstract Prototypes notation [Constantine 03a] (CAP). It is unclear what the
cause of this difference is.

The questions asking the participants about their opinion in the first part
of the tests gave some other interesting results. These questions are also
multiple choice with five possible answers: completely disagree (1), disagree
(2), neutral (3), agree (4), completely agree (5). Although all test participants
without programming experience answered the questions regarding the usage
of parameters correctly, none of them agreed (4 or 5) to the statement that
their usage was clear (just above 70 percent of the other users agreed) and 75
percent of these users disagreed (1 or 2) in the series of questions about the
scenario model. A question in the series of questions about the screen model
confirmed this result. The open remarks section did not reveal what caused
this answer. The minimalistic explanation of the usage of parameters at the
start of the test might have caused this reaction.

A second remarkable result is probably explained to a lack of explana-
tion: only 25 percent of the non programmers said that the choice structure,
which wasn’t explained using an example in the introduction, was clear to
them. Almost all other participants said they understood the choice struc-
<table>
<thead>
<tr>
<th>Question/partial solution</th>
<th>Correct answers (percentage)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All</td>
</tr>
<tr>
<td><strong>Part 1: scenario model</strong></td>
<td></td>
</tr>
<tr>
<td>1 Minimal number of participants</td>
<td>89</td>
</tr>
<tr>
<td>2 Roles (seller missing)</td>
<td>94</td>
</tr>
<tr>
<td>3 Seller is actively participating</td>
<td>39</td>
</tr>
<tr>
<td>4 scene “Bidding” can be repeated</td>
<td>89</td>
</tr>
<tr>
<td>5 Scenes that change New price</td>
<td>89</td>
</tr>
<tr>
<td>6 Scenes that use New price</td>
<td>94</td>
</tr>
<tr>
<td>7 Correct sequence of scenes</td>
<td>83</td>
</tr>
<tr>
<td><strong>Part 1: screen model</strong></td>
<td></td>
</tr>
<tr>
<td>22 Who can make a bid</td>
<td>100</td>
</tr>
<tr>
<td>23 Who is being filmed</td>
<td>100</td>
</tr>
<tr>
<td>24 How many “buyers” on screen</td>
<td>56</td>
</tr>
<tr>
<td>25 Who can see auctioneer</td>
<td>83</td>
</tr>
<tr>
<td>26 Who can see current bid</td>
<td>94</td>
</tr>
<tr>
<td>27 Who can ask question</td>
<td>94</td>
</tr>
<tr>
<td><strong>Part 2: scenario model (partial solutions)</strong></td>
<td></td>
</tr>
<tr>
<td>51 Delete old scene</td>
<td>72</td>
</tr>
<tr>
<td>52 Add scene 1</td>
<td>83</td>
</tr>
<tr>
<td>53 Add scene 2</td>
<td>22</td>
</tr>
<tr>
<td><strong>Part 2: scene and screen model (partial solutions)</strong></td>
<td></td>
</tr>
<tr>
<td>54 Scene (remove camera from role)</td>
<td>94</td>
</tr>
<tr>
<td>55 Scene (remove role)</td>
<td>33</td>
</tr>
<tr>
<td>56 Add parameter Original price</td>
<td>56</td>
</tr>
<tr>
<td>57 Remove camera (screen model)</td>
<td>83</td>
</tr>
<tr>
<td>58 Keep role in screen model</td>
<td>56</td>
</tr>
<tr>
<td>59 Remove video images seller</td>
<td>33</td>
</tr>
<tr>
<td>60 Add Original price screen</td>
<td>61</td>
</tr>
</tbody>
</table>

Table 8.2: Answers to questions with objective results for all test participants and subgroups: participants without programming experience (No progr.), participants with programming experience familiar with participation TV (TV) and participants with programming experience and not familiar with participation TV (Progr.)
8.2 Evaluation

ture. The repeat structure, however, was explained using an example and was understood by 75 percent of the non-programmers.

A third remarkable result that we saw was that the symbols used for the actions got a negative feedback of all users (both in scenario and screen model). A closer examination of the open comments added by the test participants showed that there were several reasons for this result. One of the reasons was that the figure which was used to explain the scenario model (shown in figure E.1) was not completely up-to-date and did not have numbers next to the tool symbols of CAP. This resulted in the fact that these numbers were not explained to the test users (they are used to identify the actions in the screen model). This interpretation is supported by the fact that 75 percent of the first two subgroups found the usage of letters and numbers next to the icons unclear when asked during the questions about the screen model in part 1. A second reason was inherent to the current version of SpIeLan; contrary to the tools in CAP, the corresponding icons in the scenario model do not have explanatory labels. Tool support can improve this situation by juxtaposing the scenario model and the screen model since in the screen model, the icons have an explanatory label (in the screen visualization).

A fourth remark about the icons for the actions was that the symbols for selection (✓) and editing actions (✗) were too similar. When using these are hand-drawn, they can easily be confused because straight lines may not end up being completely straight.

All test participants were neutral or positive about the newly introduced icons ParticipantElement (I) and ActiveParticipantCollection (II), with about 50 percent being positive, 39 percent neutral and some participants did not answer that question.

The usage of multiple icons within one region of the screen as in figure E.4(b) was not unclear to most users although no explanation was given about their meaning.

A last remarkable result was that three of the four participants with programming experience and experience with participation TV found the usage of colors in the screen model superfluous, while most of the other fourteen test participants disagreed with this statement. This result was consistent with a similar question in the second part of the test that asked the participants whether they found information more quickly because of the color usage. 1

This second part of the test consisted of two tasks, one 2 involved making

---

1 Different colors are used for each role and the icons for the corresponding media sources and abstract user interface components.
2 The complete description of the task is given in Dutch in figure E.6.
changes to the structure of the auction TV scenario in figure E.2 and one involved making changes to a scene shown in figure E.7, both in its representation as it is used in the scenario model and the screen model, followed by some questions about making modifications to existing specifications in SpieLan and tool support as shown in figure E.8.

As can be seen in table 8.2, the participants needed to do three actions to complete the first task: remove an existing scene, add a new scene in the same place and add an additional scene. The correct placement of the second scene required a good understanding of the scenario and the involved scenes, and the choice and repeat structure. Most users completed the first two actions successfully, but failed to perform the third action correctly. This can have multiple causes such as an incomplete understanding of the choice structure or an incomplete understanding of the scenario and the scenes. The think-aloud method used to perform this task for half of the participants in most cases did not unambiguously resolve this issue for all participants. Although in one case, it was clearly due to a misunderstanding of the choice structure; one test participant thought that the correct placement of the scene (within the choice structure) would result in a parallel execution of two scenes. Something that was not desired. A second test should be performed to show whether a better explanation of the choice structure during the start of the test will lead to better results.

The results of the execution of the second task clearly showed that tool support is required to add a user interface component to the screen (one sub task) shown during a scene or remove one from a scene (second sub task). Although all users performed at least one part of each sub task correctly, none of them performed all necessary actions. Which shows that good tool support is desirable for the notation, although more tests will be necessary to draw definitive conclusions. The test showed that these tasks have a high viscosity in SpieLan (on paper); the first sub task (the removal of a video of the seller from the screen), requires five or six individual actions (depending on how the task is performed). The second sub task, adding the original price to the screen, requires five actions. Tool support can reduce the number of required actions for both sub tasks two.

Most users agreed with the statement that making changes to the models in SpieLan was difficult, especially those of the first two sub groups shown in table 8.2. 44 percent thought it would be easier with tool support. There did not seem to be a correlation between the answers to the two questions. About 3

One can directly remove the role in the scene representation, or first remove the camera icon and then remove the role.
one of three test participants agreed with the statement that they could create a complete scenario based upon a written text describing the scenario when the scenes would be created in advance. 44 percent would need more explanations to make such a transcription. One test user agreed to the statement they would not be able to accomplish such a transcription task even after more explanations, while 6 (33 percent) answered neutral to this statement.

39 percent agreed with the statement that they did completely understand the notation after the test, which lasted between 45 minutes and one and a half hour. 39 percent more were neutral to this statement.

Most test participants agreed that a tool supporting the notation should automatically propagate changes in the screen model to the scenario model. This was reconfirmed in the last question of the test which asked the test participants about what functionality they expected from a tool supporting SpHeL.

Other often requested features for tool support were:

- drag and drop
- juxtaposition of the scenario model and the screen model with highlighting of selected items in both models
- a toolbar/legend with all modeling elements with all non-relevant elements grayed out
- error prevention and correction
- immediate validation, testing of the scenario; (live) preview
- a number of pre-built scenario’s with most common things already specified other details left open to be specified

It was clear from the questions that were asked during the think aloud tests that some details that were not given in the evaluated models were desired. For example, how can bids be changed (via fixed increments or free), whether actions are mutually exclusive or in what sequence they can be used.

Based on this experience gained in this test, the following conclusions can be drawn:

- The usage of parameters and actions as conditions for repeat and on connections should be made clearer. This can possibly be accomplished by not only using their names in these situations, but also use the corresponding symbols.
• The numbers next to the actions should be explained and can for example be used to indicate the sequence in which they can be used. Multiple characters can be used for alternate actions: e.g. two mutually exclusive actions in one scene could be name 2a and 2b while actions for which a chronological order is defined can be numbered accordingly. Action 3 is executed after action 2.

• The usage of user defined symbols can make the meaning of parameter names clearer.

• When the scenario model is edited, the screen model should be juxtaposed.

• Symbols for selection and editing actions can best be made more visually distinct.

• Tool support should guide and assist the user as much as possible to a working scenario and allow early evaluation, even of incomplete models.

• Details, such as the order in which user interface components are shown within a role in the heading of the screen model, are important and can cause a lot of confusion. The naming of all elements should be consistent and small non-meaningful differences between diagrams as well as within a diagram should be avoided.

• The usage of CAP in other circumstances has an impact on its usability.

• The test showed no significant differences between non-programmers and programmers to solve the questions and tasks in this test. They, however, found concepts such as parameters and (some details of the) choice-structures more difficult to understand.

8.3 Conclusion

This chapter provided a discussion of tool support for as well as an evaluation of a domain-specific modeling language for the design of staged participatory multimedia applications, SpIeLan, introduced in chapter 7. The language builds on a diversity of specifications, such as the graphical language for programming Lego Mindstorms [Lego 06], the Canonical Abstract Prototypes notation [Constantine 03a] and the UML activity diagram [OMG 04]. These specifications were however modified and molded into three different models
which are built around well known abstractions for this domain: scenes and screen layouts, complemented by a specification of the interactions.

Chapter 8 demonstrated that generation of abstract interactive prototypes is possible based on the scenario and the screen model, which several participants of the usability test said was required for a tool supporting SpIeLan. The screen model and the template model were compared with a subset of the notations discussed in part II. In addition the model coverage was discussed using the model elements introduced in section 2.4 and finally the results of a user study were discussed. This study showed that user with and without experience were able to interpret the language within a short time period and even make some adaptations, although the viscosity of the language due to repetition of several symbols and some hidden dependencies (for example, removal of the source of a component should also remove the component) poses problems to solve the problems completely accurate. Many of the problems discovered in the user test can be explained through the cognitive dimensions framework. This means that an thorough analysis using the cognitive dimensions framework can already bring to light many problems that users will face without the huge time investment posed by a user test. User tests are, however, still necessary because not all problems will be discovered this way and assumptions made during a theoretical evaluation using the cognitive dimensions framework can be wrong.

Furthermore, the comparative evaluation showed that the proposed notation offers some advantages over more generic specifications, such as those offered in part II for the modeling of staged participatory multimedia applications. More elaborate tool support and prototype or code generators will have to be realized to demonstrate the full potential of the introduced language and to be able to state that end users can generate shows using SpIeLan.
Tool Support and Evaluation of SpIeLan
Part IV

Conclusions and Future Work
Chapter 9

Conclusion

9.1 Contributions

In chapter 1 two overall goals were formulated for the work performed for this thesis: the definition of models that allowed to specification of models for the design context-sensitive user interfaces and the usage of models or a common meta-model for the models which would ease the creation of a model-based approach for the design of both the user interface and the functional core of the application.

In this dissertation model definitions were introduced for two different approaches to eventually reach a well integrated design: the definition of models for the design of context-sensitive user interfaces using the Unified Modeling Language [OMG 04], UML and the definition of a domain specific language for the design of staged participatory multimedia events.

The model definitions resulted in the following contributions to the field of model-based design (of user interfaces):

- In chapter 3, the ConcurTaskTrees notation was extended with contextual tasks enabling the creation of task models for dynamically context-sensitive interactive applications resulting in the Contextual ConcurTaskTrees. The environment tasks were adopted as context tasks in [Clerckx 06a], which also defined other extensions to the ConcurTaskTrees notation. Based on this model (and some additional information about the context and the presentation of the user interface) this latter approach enables the generation of complete working prototypes from
these models. The defined UML-profile allows the definition of task models which have a look that is much closer to the original Concur-TaskTrees specification than earlier proposals.

• CUP 2.0, the UML profile proposed in chapter 5, provides the some stereotypes that allow the UML 2.0 class diagram to be used for the definition of context information types. These context information types can be used to define contexts of use employing all the necessary types of context information. When the defined stereotypes are combined with those of the Ontology Definition Metamodel [IBM 06] (ODM), could even enable the usage of the class diagram for appropriate context ontologies. This would, however, require some additional research to investigate the compatibility of the relevant stereotypes of CUP 2.0 and ODM.

• CUP 2.0 also provides the necessary stereotypes to define abstract user interface models using class diagrams which includes precise bindings to datatypes, classes and operations in the application core as well as some meta-information which could allow automated generation of user interface prototypes functional prototypes (see section 5.6). The defined model allows the specification of how private information and functionality presented in the user interface is. This can be important when defining user interfaces that can take advantage of visualization resources in the environment whose (semi-)automated usage could lead to privacy issues e.g. when private data is shared on public displays.

• As part of CUP 2.0, the usage of the deployment model is proposed to specify the exact technology to be used for the definition of the user interface in a certain context of use is proposed. This specification can be a useful tool in documenting the made design choices and can potentially be used to identify possible points of attention with the chosen technology in a specific context of use.

• The System Interaction Model, introduced in section 5.5, provides an alternative view on the information modeled in the Contextual Concur-TaskTrees. This view is flow-based instead of hierarchical and is application centric. Both views on the interaction between user, application and context are complimentary. Both flow-based and hierarchical (tree-based) models are currently actively used by different stakeholder in the design of interactive applications in industry. For example, information architects frequently use flow-based specifications, while in, for example,
the automotive industry hierarchical task models are frequently used for task analysis.

- The definition of the models, leads to the fact that there are one to one mappings of unit tasks in the Contextual ConcurTaskTrees to tasks in the System Interaction Model and user interface components in the Abstract User Interface Model. The existence of such one to one mappings could potentially ease the definition transformations between these models.

- Although the proposed profiles are powerful and allow a model-based description of user interfaces at an abstract level using currently available tools, they are suboptimal for the expression of some models. Exploration of more invasive extensions or modifications of the UML for the expression of the models can be worthwhile. A tool in which the metamodel can also be edited, such as MetaSketch [Nobrega 06] can be useful in this respect.

- SpIeLan, as defined in chapter 7, introduces the usage of scenario-like specification of multi-user interactive applications, and more specifically staged participatory multimedia events.

- SpIeLan introduces a compact definition of event-driven interaction of users with a component-based application core for multi-user applications.

- SpIeLan also introduces a compact way to define high-level variation in user interfaces for multiple roles within a single screen or window.

- SpIeLan is the first language to the authors knowledge to introduce special notations for parts of the user interface that can trigger user awareness for multi-user applications.

- Testing a design language, such as SpIeLan, with users even without tool support can give valuable results. Optimal results can only be achieved when the test is prepared very carefully with much attention to detail.

9.2 Future Work

The areas for future work can be split into three major fields: enhancing tool support, creation of model transformations and enhanced/improved model
definition. The following sections will further introduce some of the next steps in develop

9.2.1 Tool Support

UML-based tool support

Part II introduced some UML profiles that allow the specification of different UML models for the design of context-sensitive user interfaces. These models have contents that have overlapping content, specified using different metaclasses. Section 2.6 showed that many software engineering approaches have an iterative design and development methodology. It is thus very likely that these models will be modified frequently. Given these two facts, it is important that model synchronization will need to be further explored. It is very probable that this synchronization cannot be done completely automatic. It is however desired that designer input will be minimal [Puerta 96b].

Another important issue is the integration of the UML tools with specialized user interface design tools. This integration can be established in two ways: writing plugins for existing UML editors ¹ or by providing import and export functionality to popular specifications within the model-based user interface community such as Teresa XML [Berti 04] or UsiXML [Limbourg 04b].

Other issues are layout algorithms that produce readable diagrams of the models and visualization techniques that support the exploration of large models.

Tool support for SpIeLan

Although SpIeLan models can currently be designed using a proof-of-concept tool (see section 8.1). The models that are created are not guaranteed to be even syntactically correct. In a first phase, a proof-of-concept tool that can only produce syntactically correct models can be created based upon a tool such as ATOM³. The creation of a tool for end-users that can effectively be used to generate staged participatory multimedia events (SPME), however, is also necessary to do a complete evaluation of the usability of the language. Some requirements for such a tool were already formulated in the discussion of the tool support for SpIeLan in section 8.2.3.

¹Many of these editors provide API’s to create plugins.
9.2 Future Work

9.2.2 Model Transformations

Complete and detailed definition of model transformations between the proposed models and toward platform specific models (and code) presented in part II can be another area of future work. Some important transformations and the problems they face are transformations between

**task model and system interaction model** Both models allow specification of more or less the same information with significantly different representations (tree-based versus flow-based representations). For example, the ConcurTaskTrees notation uses typed connections between tasks to specify temporal relations while the system interaction diagrams uses a combination of flows and control structures (such as InterruptableRegions, and, merge and decision nodes as can be seen in table 5.2 on page 87). Furthermore, automated construction of system interaction models also requires advanced graph visualization algorithms to present the model in a human-readable format.

**task model and abstract user interface model** While it is possible to derive abstract user interface models from task models [Luyten 03, Mori 03], the additional constraints posed by context-sensitivity creates additional challenges. The inverse transformation is even less explored.

**abstract and concrete user interface model** The transformation of an abstract user interface model into a concrete user interface model poses some problems because one abstract user interface components might correspond to multiple concrete user interface components. Furthermore, usage frequency and other meta-information can be used to produce results with a totally different look-and-feel. For example, usage of a wizard-like interface or a single, but larger form (as in figure 4.6 on page 4.6) can both be optimal in different contexts of use. The inverse transformation poses even more challenges and can probably be not be reached in the generic case since the necessary meta-information is often lacking. Although usage of artificial intelligence in combination with knowledge of user interface guidelines and user interface design patterns might lead to some meaningful results in limited cases.

Implementation of such transformation can also lead to additional feedback about the usefulness of the chosen abstractions.
9.2.3 Model Definition

Further work on the refinement of the defined models is a last path for future. This refinement can take multiple forms for the models defined in part II. One way to refine the models is to finetune the currently defined UML profiles, which would also be appropriate for SpIeLan. Another way is to define a more integrated meta-model, similar to the common ontology defined for all models used in Dutch [van Welie 99]. Currently a single concept does not only have multiple visual representations, but also multiple representations in the meta-model, which is less than ideal since it introduces much complexity both for the implementer and the user of the supporting tools.

For example, a single task is currently represented as a stereotyped class in the UML-version of the Contextual ConcurTaskTrees and as a stereotyped action in the system interaction model. Furthermore, many of the properties of the class are duplicated in the abstract user interface components in the abstract user interface model. This unification will require changes to the UML meta-model. These changes will require significant changes in the UML metamodel [OMG 04], which may be too invasive to be ever realized in UML: to the author’s knowledge UML currently defines different subclasses of one class to specify different visualizations in different diagrams or models. It should, however, be possible to define a meta-model that is close to UML in both representation and semantics, but extends the model for increased expressiveness regarding the specification of high-level user interface models with meta-classes shared by the different models where appropriate and provides alternative representations for different models / diagrams types.

9.2.4 Formalization of temporal and spatial relationships

A last area of future work can be a formalization of the temporal operators of the (Contextual) ConcurTaskTrees and the system interaction diagram using the relationships on temporal intervals defined by Allen [Allen 83]. Similarly, the meaning of the relations \(<\text{precede}\)\> and contain as well as their inter-relations can be formalized using “Allen relations” or their extension into two-dimensional worlds as presented by Limbourg [Limbourg 04a] for respectively one dimensional (e.g. speech) or two dimensional (e.g. graphical) user interfaces. Even a formalization into three dimensions could be considered.
Part V

Bibliography and Appendices
Bibliography

[Abrams 04] Marc Abrams & Jim Helms. *Retrospective on UI Description Languages, Based on 7 years Experience with the User Interface Markup Language (UIML).* In Luyten et al. [Luyten 04], pages 1–8.


BIBLIOGRAPHY

Dynamic Model-Based User Interface Development. In Bastide et al. [Bastide 05], pages 77–95.


[Clerckx 05] Tim Clerckx, Frederik Winters & Karin Coninx. Tool support for designing context-sensitive user interfaces using a model-based approach. In Sikorski [Sikorski 05], pages 11–18.


[Dittmar04] Anke Dittmar & Peter Forbrig. *The Influence of Improved Task Models on Dialogues*. In Jacob et al. [Jacob 05], pages 1–14.


[Markopoulos 04] Panos Markopoulos, Berry Eggen, Emile H. L. Aarts & James L. Crowley, éditeurs. *Ambient intelligence: Sec-


[Preuveneers 04] Davy Preuveneers, Jan Van den Bergh, Dennis Wagelaar, Andy Georges, Peter Rigole, Tim Clerckx, Yolande Berbers, Karin Coninx, Viviane Jonckers & Koen De


Appendix A

The Cognitive Dimensions Framework

The Cognitive Dimensions Framework was proposed by Green and Petre [Green 96]. It is a framework to do a broadbrush usability evaluation of information artifacts and has specifically been tested to perform analysis of visual programming environments. One of the advantages of the framework is that it allows a light-weight cognitive evaluation to be performed by other people than cognitive experts.

In [Green 00], the general principles of the approach are presented and demonstrated on a telephone and on a graphical and textual programming environment for the programming language, PROLOG. One of the central things in a cognitive evaluation is to determine whether some tools support the human activities. Depending on the performed activities, the importance of the cognitive dimensions differs.

Six types of activities are identified in [Green 00]:

- **incrementation** adding data which has no influence on the already present information. E.g. adding a task to a task model.

- **transcription** copying data from one medium to another: e.g. copying user roles from a screen model to a template model in SpIeLan (see chapter 7 for the model descriptions).

- **modification** for example, changing the arrangement of user interface components of an abstract presentation model in CUP.

- **exploratory design** designing something without knowing the final result.
The Cognitive Dimensions Framework

**searching** e.g. looking for all user tasks in a task model.

**exploratory understanding** discovering the structure of the user interface in an abstract user interface model as specified in chapter 5.

The framework discerns different four types of components in information artifacts: the notation or the interaction language, editing environments, a medium of interaction and sub-devices. The notation is what the user sees and edits, when what is edited cannot be seen (e.g. a phone number on many telephones) it is called interaction language. The medium of interaction is the hardware used to show the notation, such as display devices, paper or auditory displays. Components that are very important are sub-devices.

Sub-devices are parts of devices or structures. Two types of sub-devices are discerned: **helper devices** offer a new view on the notation. An example of a helper device can be an overview of cross references between models, but also a piece of paper used to make notes during development on a PC is an helper device. **Redefinition devices** are sub-devices that allow the main notation to be changed. A macro-recorder in a word processor is an example of such a sub-device. It is important to note that sub-devices often have a separate set of cognitive dimensions and notations.

Finally, the framework defines a set of cognitive dimensions. When using the dimensions it is important to keep the activities of the user in mind as is evidenced by the contents of table A.1, which was taken from the tutorial on the Cognitive Dimensions Framework [Green 98]. A first cognitive dimension is **viscosity** or the resistance to change. A viscous system needs many user actions to perform one task. **Visibility** is another important property that describes how easily the different components of the information shown in the notation can be discerned. A cognitive dimension that is highly related to visibility is **iuxtaposability**, the ability to place information entities next to each other.

The severeness of constraints on the order of doing things is expressed under the term **premature commitment**. **Hidden dependencies** are links between important entities that are not visible. **Role expressiveness** is determined by how readily the purpose of an entity can be derived from the notation. A notation is **error prone** when it invites the creation of mistakes. A system is **error prone** when it offers little protection against errors. A **secondary notation** offers the capability to make informal annotations to information entities. Indentation in programming languages such as C an Pascal are examples of secondary notations.

The cognitive dimension **abstraction** expresses the availability and types
<table>
<thead>
<tr>
<th></th>
<th>transcription</th>
<th>incrementation</th>
<th>modification</th>
<th>exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td>viscosity</td>
<td>acceptable</td>
<td>acceptable</td>
<td>harmful</td>
<td>harmful</td>
</tr>
<tr>
<td>hidden dependencies</td>
<td>acceptable</td>
<td>acceptable</td>
<td>harmful</td>
<td>acceptable</td>
</tr>
<tr>
<td>premature commit</td>
<td>harmful</td>
<td>harmful</td>
<td>harmful</td>
<td>harmful</td>
</tr>
<tr>
<td>abstraction barrier</td>
<td>harmful</td>
<td>harmful</td>
<td>harmful</td>
<td>harmful</td>
</tr>
<tr>
<td>abstraction hunger</td>
<td>useful</td>
<td>useful (?)</td>
<td>useful</td>
<td>harmful</td>
</tr>
<tr>
<td>secondary notation</td>
<td>useful (?)</td>
<td>-</td>
<td>very useful</td>
<td>v. harmful</td>
</tr>
<tr>
<td>visibility / iuxtempoability</td>
<td>not vital</td>
<td>not vital</td>
<td>important</td>
<td>important</td>
</tr>
</tbody>
</table>

Table A.1: Overview of the significance of a selection of cognitive dimensions for certain tasks as defined in [Green 98]

of abstractions offered to the user. Global find-and-replace functionality is a form of abstraction, just as the ability to make selections of multiple entities and perform changes on this selection. If a user is allowed to define his or her own abstractions an abstraction manager, which is a redefinition sub-device, is needed. A system is abstraction tolerant when it allows the definition of abstraction but does not require it. When the definition of abstractions is required, a notation is abstraction hungry. The abstraction barrier expresses how many abstractions have to be learned before a user can start using the notation.
The Cognitive Dimensions Framework
Appendix B

Example Models of Related Work

B.1 Introduction

This appendix provides some illustrations of three model-based approaches that have had the most impact on the work presented in this dissertation. The first method is Wisdom, extensively discussed by Nunes in his PhD dissertation [Nunes 01]. The second notation for user interface models that is discussed in this appendix is UMLi, introduced by da Silva [da Silva 02]. Finally, UsiXML, proposed by Limbourg [Limbourg 04a], is discussed.

B.2 Wisdom

Wisdom is a user-centered UML-based software engineering methodology for small-team projects. The method starts with the specification of use cases. In order to accurately define these use cases, three stereotypes that can be applied to UML actors are proposed. They can be seen in figure B.1. The stereotypes for the analysis and interaction models are shown in figure B.2. The stereotypes for the analysis model are in fact those that are commonly used in UML robustness diagrams. The interaction model specifies how a user can interact through a user interface («Interaction Space») with objects in the functional core («entity»).

An example of a presentation model is shown in figure B.3. It shows the interaction spaces of a hotel reservation system. Interaction spaces correspond to windows or panels in graphical final user interfaces. The task or dialog
model for this hotel reservation system is shown in figure B.4. It is a UMLified version of the ConcurTaskTrees notation [Paternò 98].
B.2 Wisdom

Figure B.3: Example presentation model in Wisdom notation [Nunes 01]

Figure B.4: Example task model in Wisdom notation [Nunes 01]
B.3 UMLi

UMLi [da Silva 02] is a heavy-weight extension of the UML meta model. The extensions are mainly defined to create a UML presentation model and an activity model that is better able to capture interactive systems than the standard UML activity diagram. An example of a presentation model is shown in figure B.5. 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 (for each there is a label and an inputfield), an area that allows to start two kinds of searches (approximate or exact) and an 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.

The changed activity diagram shown in figure B.6 is started by an interaction of a user. This is shown through the usage of a black square instead of a black dot as a start pseudostate. The specification contains object flows to show the relations between action states and presentation objects through which the actions are performed. UMLi defines a set of new pseudostates. One of these pseudostates is shown in figure B.6: a circle with an horizontal line through it. The action states connected through bidirectional arrows can be performed as long as they are not confirmed. In this case, this means that a user can specify his username and his password and change them until she confirms them through the OK control. The two nested circles indicate the end of the interaction within an action state.
Figure B.5: A presentation model using UMLi [da Silva 02]

Figure B.6: A activity model using UMLi [da Silva 02]
UsiXML is a specification language that reuses semantics of pre-existing models and enhances them in certain areas. Figure B.7 shows the relation between two XML models, the task model and the domain model, which are both based on existing specifications: the ConcurTaskTrees [Paternò 98] and the UML class diagram. In the figure both are shown using their original notation.

Limbourg [Limbourg 04a] proposes a transformation-based development process using the UsiXML language. Figure B.8 shows an example of a mapping rule that can be a part of a transformation between a task model and an abstract user interface model. It is specified in a graphical language, which is proposed to be the graphical language for UsiXML in [Limbourg 04a].
Figure B.8: A mapping rule between that task model and the abstract user interface model [Limbourg 04a]

Figure B.9: An abstract user interface model using the graphical notation proposed by Limbourg [Limbourg 04a]
Figure B.10: An concrete user interface model using the textual notation for UsiXML [Limbourg 04a]

Figure B.9 shows an example of an abstract presentation model using this same graphical notation. It specifies the abstract user interface of an electronic questionnaire. The user of such questionnaire is first asked to specify his name, followed by his address, zip code, sex and age. After this she can answer questions that are shown. Finally, she can send the questionnaire. Figure B.10 shows a concrete user interface description corresponding with the abstract presentation model in figure B.9 using XML, the textual representation of UsiXML. The same user interface is shown in figure B.11 using a Windows look-and-feel.

Some UsiXML tools use alternative graphical notations. The notations that are used in IdealXML [Montero 05] are shown in figure B.12.
Figure B.11: An concrete user interface model using a Windows look-and-feel [Limbourg 04a]
Figure B.12: The graphical notation for a subset of the UsiXML models as used in IdealXML [Montero 05]
Appendix C

Canonical Abstract Prototypes

The Canonical Abstract Prototypes notation (CAP) was introduced by, among others, Larry Constantine in 2000 and got a significant update [Constantine 03b] for his keynote [Constantine 03a] at DSV-IS 2003 [Jorge 03]. CAP was introduced to build a bridge between task models (task trees) and concrete graphical user interface design. The Canonical Abstract Prototypes can be considered as a “standard” toolkit to define abstract graphical layouts.

The toolkit uses abstractions that are based upon what a user can expect from a certain component of the user interface: abstract user interface components. Three basic types of components are discerned: abstract tools (see definition C.1), abstract materials (see definition C.2) and abstract active materials, which is a combination of the two previous abstract user interface components.

Definition C.1 An abstract tool is an abstract user interface component that operates upon material(s) or initiates some action(s).

Definition C.2 An abstract material is an abstract user interface component representing a container, information, or data.

Table C.1 gives an overview of all abstract user interface components that are part of the CAP. Each row shows the icon for the abstract tool, its name and finally some examples. The basic icon for each type of user interface component is shown on the first row below the type of user interface component.

An example of a specification using the CAP is shown in figure C.1. It shows the specification of a film viewer. In the top left, a user can enter a
figure

Figure C.1: CAP example: a film-clip viewer [Constantine 03b]

film title and then start the search using the Find tool. The results are shown in the container Film Clips. For each film clip in the results, the Title, which is editable (editable element) and the Length are shown. The three downpointing arrows indicate that the Title and Length are repeated for each film in the list of films. In the right bottom corner of this collection, there are two tools: duplicate and delete.

The right side of the abstract prototype, shows a Film Clip View that contains a Frame Image, the current frame (Frame) and the title of the film clip. The bottom right corner has four tools: 2 for frame by frame navigation through the film clip (Up 1 Frame and Back 1 Frame), a tool Play and a end tool Stop. These four tools are part of a logical container indicated by a dashed rectangle.

A last feature of the CAP is that it allows free annotations, such as the annotation synchronized to selection, which indicates that the contents of the Frame Image is synchronized to the currently selected film clip in the collection Film Clips.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Abstract tools</strong></td>
<td></td>
</tr>
<tr>
<td>🛠️</td>
<td>action/operation</td>
<td>Print symbol table, Color selected shape</td>
</tr>
<tr>
<td>🌈</td>
<td>start/go/to</td>
<td>Begin consistency check, Confirm purchase</td>
</tr>
<tr>
<td>🌐</td>
<td>stop/end/complete</td>
<td>Finish inspection session, Interrupt test</td>
</tr>
<tr>
<td>👤</td>
<td>select</td>
<td>Group member picker, Object selector</td>
</tr>
<tr>
<td>📑</td>
<td>create</td>
<td>New customer, Blank slide</td>
</tr>
<tr>
<td>🗑️</td>
<td>delete, erase</td>
<td>Break connection line, Clear form</td>
</tr>
<tr>
<td>💼</td>
<td>modify</td>
<td>Change shipping address, Edit client details</td>
</tr>
<tr>
<td>🖇️</td>
<td>move</td>
<td>Put into address list, Move up/down</td>
</tr>
<tr>
<td>📔</td>
<td>duplicate</td>
<td>Copy address, Duplicate slide</td>
</tr>
<tr>
<td>🌃</td>
<td>perform (&amp; return)</td>
<td>Object formatting, Set print layout</td>
</tr>
<tr>
<td>🥁</td>
<td>toggle</td>
<td>Bold on/off, Encrypted mode</td>
</tr>
<tr>
<td>🎨</td>
<td>view</td>
<td>Show file details, Switch to summary</td>
</tr>
<tr>
<td></td>
<td><strong>Abstract materials</strong></td>
<td></td>
</tr>
<tr>
<td>🏷️</td>
<td>container</td>
<td>Configuration holder, Employee history</td>
</tr>
<tr>
<td>🔍</td>
<td>element</td>
<td>Customer ID, Product thumbnail image</td>
</tr>
<tr>
<td>📜</td>
<td>collection</td>
<td>Personal addresses, Electrical Components</td>
</tr>
<tr>
<td>📤</td>
<td>notification</td>
<td>Email delivery failure, Controller status</td>
</tr>
<tr>
<td></td>
<td><strong>Abstract active materials</strong></td>
<td></td>
</tr>
<tr>
<td>🤩</td>
<td>active material</td>
<td>Expandable thumbnail, Resizable chart</td>
</tr>
<tr>
<td>🤓</td>
<td>input/accepter</td>
<td>Accept search terms, User name entry</td>
</tr>
<tr>
<td>📖</td>
<td>editable element</td>
<td>Patient name, Next appointment date</td>
</tr>
<tr>
<td>📜</td>
<td>editable collection</td>
<td>Patient details, Text object properties</td>
</tr>
<tr>
<td>📜️</td>
<td>selectable collection</td>
<td>Performance choices, Font selection</td>
</tr>
<tr>
<td>📜️</td>
<td>selectable action set</td>
<td>Go to page, Zoom scale selection</td>
</tr>
<tr>
<td>📜️</td>
<td>selectable view set</td>
<td>Choose patient document, Set display mode</td>
</tr>
</tbody>
</table>

Table C.1: Canonical abstract user interface components
Appendix D

Unified Modeling Language

D.1 Introduction

The Unified Modeling language is the result of integration of different object oriented design and analysis notations. The effort was started in Rational, which in 1994 employed the creators of both Booch and OMT. They started to work on a Unified Methodology. They, however, saw that a unified modeling language would more easily be embraced. In 1995, Ivar Jacobson, the creator of OOSE, joined their ranks. Together they continued to work on a unified modeling language. From 1996 on, the effort to create the Unified Modeling Language was coordinated through the Object Management Group (OMG). This resulted in the adoption by the OMG of UML 1.1 in late 1997. Several smaller revisions followed. One of those revisions, UML 1.4.2, became ISO 19501:2005. Meanwhile, work started on a major revision which resulted in the adaptation by the OMG of UML 2.0 in late 2005. A summary of this history is shown in figure D.1. [Nunes 01, OMG 06c]

D.2 UML 2.0

UML 2.0 is a specification that can largely split in two parts and a set of related specifications. The two parts of UML 2.0 are the UML Infrastructure, which is an extended version of the Meta-Object Facility [OMG 06a] (MOF) to create extra features considered to be required for UML. The second part is the UML super structure, which is the part of UML that is used by most
Figure D.1: UML history

Figure D.2 shows at which modeling levels, the UML Infrastructure en UML super structure can be placed as specified in the Model-Driven Architecture [Miller 03]. The UML Infrastructure is a meta-meta model (level M3), while the UML super structure is for the most part situated at level M2. Only the package Profiles, which describes the light-weight extension system of the UML super structure, is specified at level M3.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Beacz 1991</td>
<td>Beacz 1993</td>
<td>Rumbauch (CMT) 1993</td>
<td>Jacobson (SD) 1993</td>
<td>Unified Method 0.8</td>
<td>Object Analysis &amp; Design RFP-1 (uml)</td>
<td>UML 1.0 initial submission to OMG (jan)</td>
<td>UML 1.1 final submission to OMG (sep)</td>
</tr>
<tr>
<td>1999</td>
<td>2000</td>
<td>2001</td>
<td>2002</td>
<td>2003</td>
<td>2004</td>
<td>2005</td>
<td>2006</td>
</tr>
<tr>
<td>UML 1.3 (minor technical changes, jan)</td>
<td>UML 1.4 (minor revision, dec)</td>
<td>UML 1.5 (major revision)</td>
<td>UML 1.5 (minor revision)</td>
<td>UML 2.0 (major revision, jul)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure D.2: MDA modeling levels

The general discussion of UML in this appendix is concluded by an overview of the UML diagram types, as discussed in [Wikipedia 06]. This overview is not normative since the UML super structure does not define these diagram types. They, however, correspond to the most commonly used diagram types.
that do not use profiles are other extensions of UML. Figure D.3 provides an overview of these diagrams.

These diagram types can be divided into two categories corresponding to the two major parts of the UML super structure specification: structure diagrams and behavior diagrams. The five structure diagram types listed are:

**class diagram** Diagrams of this type specify relations between classes and interfaces

**composite structure diagram** Composite structure diagrams specify the internal structure of a class such as the relations and constraints between attributes of the class.

**component diagram** Component diagrams provide an overview of relations between different components of the specified system.

**deployment diagram** A deployment diagram shows how a system is deployed to a specific platform (which can consist of both hardware and software).

**object diagram** Diagrams of this type specify relations between class instances or objects.

**package diagram** Package diagrams describe relations between packages, which are used to group classes that logically belong together and thus provide some structure in the large amount of classes and other UML elements that are present in many software projects.

There are seven types of behavior diagrams, four of them belong to a special sub category called interaction diagrams. These four are: sequence diagrams, communication diagrams (called collaboration diagrams in UML 1.x), interaction overview diagrams and timing diagrams. The other three diagram types are the use case diagram, the activity diagram and the state machine diagram. The main difference between the interaction diagrams and the other behavior diagrams is that the interaction diagrams have a greater emphasis on the parts of the system that carry out the actions.
D.3 UML modeling elements

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Artifact</td>
</tr>
<tr>
<td>Artifact</td>
<td>DeploymentSpecification</td>
</tr>
<tr>
<td>Deployment</td>
<td></td>
</tr>
</tbody>
</table>

Table D.1: UML symbols deployment diagram, figures courtesy of [OMG 04]
### Table D.2: UML symbols class and package diagram, figures courtesy of [OMG 04]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td></td>
</tr>
<tr>
<td>PackageSpecification</td>
<td></td>
</tr>
<tr>
<td>Aggregation (AssociationKind)</td>
<td></td>
</tr>
<tr>
<td>Composition (AssociationKind)</td>
<td></td>
</tr>
<tr>
<td>Association</td>
<td></td>
</tr>
<tr>
<td>Generalization</td>
<td></td>
</tr>
<tr>
<td>Dependency</td>
<td></td>
</tr>
<tr>
<td>PackageImport</td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td>Name</td>
</tr>
<tr>
<td>--------</td>
<td>------------------</td>
</tr>
<tr>
<td><img src="image" alt="Action" /></td>
<td>Action</td>
</tr>
<tr>
<td><img src="image" alt="CallBehaviorAction" /></td>
<td>CallBehaviorAction</td>
</tr>
<tr>
<td><img src="image" alt="AcceptEventAction" /></td>
<td>AcceptEventAction</td>
</tr>
<tr>
<td><img src="image" alt="SendSignalAction" /></td>
<td>SendSignalAction</td>
</tr>
<tr>
<td><img src="image" alt="ObjectName" /></td>
<td>ObjectNode</td>
</tr>
<tr>
<td><img src="image" alt="InterruptableRegion" /></td>
<td>InterruptableRegion</td>
</tr>
<tr>
<td><img src="image" alt="ActivityPartition" /></td>
<td>ActivityPartition</td>
</tr>
<tr>
<td><img src="image" alt="ObjectFlow" /></td>
<td>ObjectFlow</td>
</tr>
<tr>
<td><img src="image" alt="ControlFlow" /></td>
<td>ControlFlow</td>
</tr>
<tr>
<td><img src="image" alt="DecisionNode" /></td>
<td>DecisionNode</td>
</tr>
<tr>
<td><img src="image" alt="MergeNode" /></td>
<td>MergeNode</td>
</tr>
<tr>
<td><img src="image" alt="ForkNode" /></td>
<td>ForkNode</td>
</tr>
<tr>
<td><img src="image" alt="JoinNode" /></td>
<td>JoinNode</td>
</tr>
<tr>
<td><img src="image" alt="InitialNode" /></td>
<td>InitialNode</td>
</tr>
<tr>
<td><img src="image" alt="ActivityFinal" /></td>
<td>ActivityFinal</td>
</tr>
<tr>
<td><img src="image" alt="FlowFinal" /></td>
<td>FlowFinal</td>
</tr>
</tbody>
</table>

Table D.3: UML symbols activity diagram, figures courtesy of [OMG 04].
Appendix E

Usability test

This appendix shows selected parts of the usability test discussed in section 8.2.3. All texts and questions are in Dutch. The models are in English. For the sake of brevity, only the models and the questions of the text are shown. The explanations that were given with each of the introductory models, is not reproduced as part of this text.

The test consists of two parts: the first evaluates the comprehension of given models, the second evaluates how easy it is to modify existing models.

The structure of the first part is as follows:

1. An explanation of the notation for the scenario model using the example shown in figure E.1.

2. A series of questions, shown in figure E.3, about the scenario model shown in figure E.2. Except for one open question, all questions are multiple choice. The first part of the questions has objectively verifiable answers, while the second part asks the test users about their opinion over the language.

3. An explanation of the notation for the screen model using the example shown in figure E.4(a).

4. A series of questions, shown in figure E.5, about the scenario model shown in figure E.4(b). The first part of the questions has objectively verifiable answers, while the second part asks the test users about their opinion over the language.
The structure of the second part consists of two modification tasks and a general questionnaire about modification and creation of the models evaluated in the test:

1. A task in which one scene in a scenario should be replaced by two scenes which together offer similar, but enhanced, functionality. The task description is reproduced in figure E.6. The scenario that should be adapted is the same as the one about which the questions were asked in the first part of the test (see figure E.2.

2. A task in which a scene should be modified. The models corresponding to this scene differ from the one in the first part of the test, but not much. Two modifications should be made: adding a piece of information to the screen and removing a piece of information to the screen. The description of the task as it was given to the test users is reproduced in figure E.7.

3. A list of general questions in which the test users were asked about their opining about how easy they think it is to make changes to existing models, create new models and what tool support they deem necessary.
Figure E.1: Example used to explain notation used in scenario model
Figure E.2: Example used for questions about scenario model
1. Hoeveel personen moeten minimaal deelnemen aan de show?
   a. 1
   b. 2
   c. 3

2. Nog dat al deze personen de rol "Buyer" hebben, behalve 1 met de rol "Actor/Guide". Kan de show dan starten?
   a. Ja
   b. Neen, er moet minimaal één persoon met de rol "Actor/Guide" aanwezig zijn.
   c. Neen, er moeten er ten minste twee personen aanwezig zijn.

3. Nemen de personen met de rol "Actor/Guide" aan de show?
   a. Ja, ze zijn bij de beginzet van de show aanwezig.
   b. Ja, maar zij hebben geen invloed op de verloop van de show.

4. Is het mogelijk dat de rol "Actor/Guide" meer dan 1 keer getoond wordt in een show?
   a. Nee
   b. Als alle meerdere karriëren elk handelen.
   c. Nee, maar de meerdere karriëren elkaar.

5. In welke acties kan de waarde van de parameter "new_price" worden gewijzigd?
   a. Welkom, Seller Introduction, Item Presentation, Buyer Question, Seller Answer, End Bidding
   b. Welkom, Seller Introduction, Item Presentation, Bidding or Start Question, Buyer Question, Seller Answer, Bidding or Start Question, Buyer Question, Seller Answer, End Bidding
   c. Welkom, Seller Introduction, Item Presentation, Bidding or Start Question, Bidding or Start Question, End Bidding

6. Welke ongeplande acties zijn gedekt volgens de specifieke?
   a. Welkom, Seller Introduction, Item Presentation, Bidding or Start Question, Buyer Question, Seller Answer, End Bidding
   b. Welkom, Seller Introduction, Item Presentation, Bidding or Start Question, Buyer Question, Seller Answer, Bidding or Start Question, Buyer Question, Seller Answer, End Bidding
   c. Welkom, Seller Introduction, Item Presentation, Bidding or Start Question, Bidding or Start Question, End Bidding

7. Welke acties worden gegeven, als de volgorde van de aanbieding van de item wordt gewijzigd?
   a. Welkom, Seller Introduction, Item Presentation, Bidding or Start Question, Buyer Question, Seller Answer, End Bidding
   b. Welkom, Seller Introduction, Item Presentation, Bidding or Start Question, Bidding or Start Question, End Bidding
   c. Welkom, Seller Introduction, Item Presentation, Bidding or Start Question, Bidding or Start Question, Buyer Question, Seller Answer, End Bidding

Met de volgende vragen willen we nagaan of de Speldata-normatie duidelijk vindt. Bij elke vraag is deke 1 antwoord mogelijk. Ja welke-meteen gaat het voor de volgende vragen: T1 is het minimaal niet aanwezig, 2 is dat aanwezig, 11 is overweldigd, 4 is bijna niet aanwezig, 5 is het minimaal aanwezig.

1. De acties van de "Actor/Guide" bijvoorbeeld is duidelijk.
   a. 1
   b. 2
   c. 3

9. Het gebruik van parameters is duidelijk.
   a. 1
   b. 2
   c. 3

   a. 1
   b. 2
   c. 3

12. De invloed van "Choice" - "End Choice" op een show is duidelijk.
   a. 1
   b. 2
   c. 3

13. De opbouw van een show is acties, objecten en een goede zaak.
   a. 1
   b. 2
   c. 3

14. Het tonen van welke rolnen en actie's kunnen we zien in een show is een goed idee.
   a. 1
   b. 2
   c. 3

15. De symbolen voor het uitspreken op het venster zijn duidelijk.
   a. 1
   b. 2
   c. 3

16. De symbolen voor het uitspreken van de acties en het uitspreken van de acties en het uitspreken van de acties zijn duidelijk.
   a. 1
   b. 2
   c. 3

17. Het aanbieden van de aanbieding van de item wordt duidelijk.
   a. 1
   b. 2
   c. 3

18. Een diagram is een goede manier om de structuur van een show te weergeven.
   a. 1
   b. 2
   c. 3

19. Welk de acties worden gegeven, als de volgorde van de acties aan de dag van de dag wird gewijzigd.
   a. 1
   b. 2
   c. 3

20. Er zijn acties in de diagram om de volledigheid structuur van het bijlage te begrijpen.
   a. 1
   b. 2
   c. 3

21. Hieronder zijn op de onderlagen van deze pagina kunnen vinden eventuele redenen op verklaring wat de regels zijn in de Taalname Diagrammen.
**Vragenreeks “Bidding or StartQuestion”**

In figuur 11 vindt een Screen Model van een zender van de televisie reedsing. Deze vindt vaak goede vragen over het gewenste dagboek. Met deze vragen vullen we magazins zoals de Spelavond in duidelijk. We vullen van verschillende Mori dan de duidelijkheid van dit model, zodat geen fouten maken. Bij dit vraagmodel heeft 1 antwoord mogelijk.

**Screen Model**

Een Screen Model is het vertrek van een volledige TV-programma uit de antwoordvragen van verschillende zenders. Elk van deze zenders wordt in meer detail uitgezet in een Screen Model.

**Voorbeeld: “SellerAnswer”**

In figuur 7 vindt een Screen Model van een Seller and Buyer bij de enige vragen van de vorige vragen, namelijk de “SellerAnswer” zender. Daarna wordt dit Screen Model mogelijk voor meer in detail uitgezet.
22. Wie kan een bedrijfsleven in deze situatie aannemen?

- Salesman
- Een van de rollen: "Buyer", "Bidders", "Auctioneer" en "Seller" is afgevoerd

23. Wie wordt gezien tijdens deze scène?

- Alle personen die deel uitmaken van deze situatie
- Een van de rollen: "Buyer", "Bidders", "Auctioneer", "Seller" is afgevoerd

24. Kan er van gereed worden gemaakt dat alle personen met de rol "Buyer" van afstand gebaad worden op het scherm?

25. Wie kan de films van de "Auctioneer" zien?

- Alle personen behalve de auctioneer
- De auctioneer zelf

26. Wie kan het huidige bed "van op afstand" zien?

- Alle personen behalve de auctioneer
- De auctioneer zelf

27. Wie kan een vraag stellen?

- Alle personen behalve de rol "Seller" of "Auctioneer" behalve
- Salesman

Met de volgende vragen willen we nagaan of de e-standaard op academisch niveau is. Bij elke vraag is slechts 1 antwoord mogelijk. En welke mate gaat u als u met de volgende vragen deel uitmaken

20. Het gebruik van blauw om aan te geven wie wat kan zien, is voluit onnodoig?

21. De aantallen symbolen voor delelen van de schijf waarin informatie over deelnames aan de aangeboden worden, is ondubbelzinnig.

22. Het gebruik van de rol "All" is duidelijk.

23. Het gebruik van de rol "All" is ondubbelzinnig.

24. De afbeelding van de onderdelen van een schijf is ondubbelzinnig.

25. Het plaatsen van de rollen in een eenduidige onderdeel van het schijf, als er andere onderdelen in een duidige regel van een schijf gebeurd kunnen worden, is duidelijk.

26. De inleiding voor live videos of van te voren opgenomen video zijn duidelijk.

27. De aantallen symbolen voor de rollen zijn onduidelijk.

28. Het plaatsen van de rollen in een eenduidige onderdeel van het schijf is onduidelijk.

29. Hieronder kan vẫn voor verschillende andere oplossingen geven die u hebt over de notatie die gebruikt wordt onder e-Stoer Model.
Figure E.6: Task: adaptation of the scenario model shown in Figure E.2.

Heedige situatie: De acteur waarin een "Buyer" zijn bed kan veranderen (actie 2) in "Building or StartQuotation". Dit bed wordt dan onmiddellijk gekoppeld. Actie 3 laat een aankoop van "Buyer" worden gedaan (actie "Building Quotation") wanneer gedeeltelijk of volledig verkoop wordt door de "Bid or StartNextScene" (Figure 3) laat toe dat aan "Buyer" een bed beheer (actie 2) opnemen.

"Confirm and Raise Bid" laat de "Actor" toe om dat activiteit aan te plegen en de hoogte van het volgende bed met te stellen (actie 3).

Geef in het bovengenoemde Scenario Diagram (Figure 1) aan wat u zou veranderen om aan de nieuwe verhardste velden.

Gebruik van de volgende symbolen:
1. Zet een kruis over elke steek of onder de steek van het diagram dat u wilt verwijderen.
2. Plaats een op de plaats waar u de scène "Before StartNext" wilt toevoegen.

Figure 1: Scene "Bid or StartNextScene".

Figure 2: Scene "Bid or StartNextScene".

Figure 3: Scene "Confirm and Raise Bid".

3. Plaats een op de plaats waar u de scène "Raise Bid" wilt toevoegen.
4. Geef hieronder een benaming aangegeven over een kruis.

---

Usability test
Opdracht 2
Er is gebleken dat het net mogelijk dat er videobevelden werden getoond van de personen met de rol "seller" tijdens het beduifen van het starten van een vraag. Het is echter wel mogelijk de

1. In de scène in het Screenshot Diagram (Figuur 5 en Figuur 6)
2. In het Scene Model (Figuur 7 en Figuur 8)
   - Ontdekken van het verreden en door een klus te plaatsen over het te vervullen onderdelen
   - Toevertrouwen door het genuino onderdelen te tonen en op de lege voorstelling
   - Gelezen van het Figuur 8 om meken te realiseren

![Bid or StartNextScene Diagram](image1)

**Figuur 5: Screenshot de overbodige elementen in deze scène**

![Bid or StartNextScene Diagram](image2)

**Figuur 6: Toegewezen elementen toe in deze scène**

![Bid or StartNextScene Diagram](image3)

**Figuur 7: Scene Model: Schot bij de overbodige elementen**
Vragenreeks
De volgende vragen en stellingen zullen niet gebruikt worden om u te evalueren, maar wel om de notatie te evalueren, u kan dus geen fouten maken. Bij elke vraag is slechts 1 antwoord mogelijk.
In welke mate gaat u akkoord met de volgende stellingen? 1 is helemaal niet akkoord, 2 is niet akkoord, 3 is neutraal, 4 is akkoord, 5 is helemaal akkoord.
1. Het aanpassen van de diagrammen vond ik moeilijk.
   1 □ 2 □ 3 □ 4 □ 5 □
2. De opdracht zou gemakkelijker geweest zijn met behulp van een computerprogramma.
   1 □ 2 □ 3 □ 4 □ 5 □
3. Ik kan in een geschikt programma een uitgeschreven scenario omzetten naar een show met behulp van de notatie in deze test.
   1 □ 2 □ 3 □ 4 □ 5 □
4. Ik heb meer uitleg nodig om een uitgeschreven scenario om te zetten naar een show met behulp van de Splelan-notatie.
   1 □ 2 □ 3 □ 4 □ 5 □
5. Ik kan ook met meer uitleg, een uitgeschreven scenario niet omzetten naar een show met behulp van de Splelan-notatie.
   1 □ 2 □ 3 □ 4 □ 5 □
6. Ik begrijp de notatie nu helemaal.
   1 □ 2 □ 3 □ 4 □ 5 □
7. Het gebruik van kleuren zorgt ervoor dat informatie snel kan gevonden worden.
   1 □ 2 □ 3 □ 4 □ 5 □
   1 □ 2 □ 3 □ 4 □ 5 □
Open vragen
9. Welke functies verwacht u van een computerprogramma dat met behulp van deze taal toelaat om een televisieshow te maken? Schets hoe zo'n programma eruit zou kunnen zien.

Figure E.8: Questions about adaptation of models
Appendix \textbf{F}

Scientific Contributions and Publications

This appendix lists the international refereed publications co-authored by Jan Van den Bergh during the period of his doctoral research.

The publications are split in two sections. Section \textbf{F.1} lists the papers whose contribution is discussed is fully or partly discussed in this dissertation. Section \textbf{F.2} lists the papers whose contribution is not discussed or discussed as related work in this dissertation.

\section*{F.1 \textbf{Publications with Contributions to Dissertation}}

The following papers discussed in this section directly contributed to this dissertation. \cite{Van_den_Bergh_04a} introduces the Contextual ConcurTaskTrees, discussed in chapter 3. The UML-profile for the Contextual ConcurTaskTrees is not discussed in this paper.

\cite{Van_den_Bergh_04b} forms the basis for the discussion of the related approaches of model-based user interface design in chapter 2.

\cite{Van_den_Bergh_05a} discusses the context integration in the first version of CUP-profile, discussed in chapter 4. \cite{Van_den_Bergh_05b} introduces the CUP-profile with a focus on the visualization aspect. It uses the Cognitive Dimensions Framework \cite{Green_96} for a broadbrush evaluation of the usability of CUP. \cite{Van_den_Bergh_05c} introduces the first aspects of CUP 2.0. A complete discussion of CUP 2.0, the foundation of chapter 5 is provided in \cite{Van_den_Bergh_06a}, while \cite{Van_den_Bergh_06b}, a position paper, introduces the methodology discussed in section 6.2.2.
[Van den Bergh 06c] introduces the scenario model and the screen model of SpIeLan discussed in chapter 7 and the generation of interactive abstract prototypes discussed in section 8.1.2. The template model is introduced in [Van den Bergh 06d].


F.2 Other Publications


F.2 Other Publications

[Clerckx 06] discusses an integration of environment tasks into the task model introduced in [Clerckx 04b].

[Houben 05] discusses how the activity model of CUP as discussed in chapter 4 can be used to give a high-level description of a road assistance system, a multi-user multi-context context-sensitive interactive application.

[Luyten 06] discusses how the user interface deployment diagram as discussed in chapter 5 can be used to describe how a distributed user interface is spread over multiple devices.

[Coninx 03, Preuveneers 04, Van den Bergh 02, Van den Bergh 03, Van den Bergh 04] do not include contributions of this dissertation.


Scientific Contributions and Publications


Bijlage G

Samenvatting

G.1 Inleiding

Recentelijk is er een evolutie in hoe en waar interactieve digitale toepassingen gebruikt worden. Het gebruik van interactieve digitale toepassingen is op dit ogenblik al ver doorgedrongen binnen kantoor- en thuisomgevingen. Deze toepassingen worden echter ook meer en meer gebruikt in mobiele toestellen en omgevingen. Een indicatie van deze evolutie is het succes van laptops; in 2005 zijn er meer laptops verkocht dan vaste desktop computers. Nog mobielere toestellen zoals smartphones, GSM’s waarop men ook interactieve applicaties zoals routeplanners, agenda’s, spelletjes enz. kan installeren, en PDA’s zijn ook aan een stijle opgang bezig.

Tegelijkertijd is er een evolutie in het aantal van zulke digitale toestellen dat iemand gebruikt. Men gebruikt op dit moment vaak meer dan een interactief digitaal toestel, vb. een laptop of desktop computer en een smartphone. Daar in verschillende omstandigheden hetzij het ene, hetzij het andere toestel meer geschikt is om te gebruiken, zouden gebruikers het interessant vinden dat ze dezelfde (of gelijkaardige) toepassingen op beide toestellen kan gebruiken. Deze toepassingen moeten dan wel optimaal aangepast zijn aan het toestel en aan de omstandigheden waarin ze gebruikt worden.

Dit leidt er toe dat de ontwikkeling van zulke toepassing ontzettend complex wordt. Een gestructureerde aanpak die gebruik maakt van modellen met een hogere abstractie is aangewezen om goed te kunnen omgaan met deze complexiteit. Daarnaast is het belangrijk dat er in zo’n aanpak voldoende aandacht is voor de omstandigheden waarin zulke interactieve digitale toepas-
Samenvatting

Informatie over deze omstandigheden, met inbegrip van de gebruik, het platform (hardware en software), de omgeving en de software- en hardwaretoepassing wordt ook wel *context* genoemd. Deze observaties leidden tot het opstellen van de doelstellingen van dit proefschrift, geformuleerd in hoofdstuk 1. Deze doelstellingen kunnen als volgt samengevat worden: de definitie van gedetailleerde, maar hoog-niveau modellen voor het ontwerp van contextgevoelige gebruikersinterfaces die hetzelfde meta-model gebruiken dat voor het ontwerp van de rest van de contextgevoelige toepassingen wordt gebruikt. Hoofdstuk 1 tevens voorziet in een overzicht van de verdere structuur van het werk.

In hoofdstuk 2, wordt een precieze definitie gegeven van wat context inhoudt in dit werk. Tevens worden in hoofdstuk twee definities gegeven van gerelateerde concepten zoals gebruikscontext (*context of use*) en contextgevoelige gebruikersinterface (*context-sensitive user interface*).

Hoofdstuk 2 introduceert ook een selectie van model elementen die later in dit hoofdstuk gebruikt worden om een vergelijkende studie te doen van enkele relevante modelgebaseerde benaderingen voor het ontwerp van gebruikersinterfaces. Er wordt besproken hoe deze model elementen gebruikt worden in modellen in deze benaderingen en hoe de modellen worden voorgesteld (grafisch, tekstueel of beiden) en de bruikbaarheid van deze voorstellingen. Uit deze studie blijkt dat de ondersteuning van contextgevoeligheid in gerelateerde aanpakken nog niet voldoende is.

Het proefschrift bestaat verder uit drie delen. Het eerste deel hiervan bestaat uit vier hoofdstukken en introduceert modellen en notaties voor contextgevoelige gebruikersinterfaces. Deze modellen kunnen allen worden uitgedrukt met behulp van UML [OMG 04] (zie ook appendix D), uitgebreid met *UML profiles*. Dit zijn lichtgewicht uitbreidingen aan UML die toelaten om de semantiek van bestaande UML klassen te verfijnen of meer specifieke eigenschappen toe te kennen voor een specifiek domein, in dit geval contextgevoelige gebruikersinterfaces, te maken.

De genericiteit van UML is niet echter altijd een voordeel. Vooral wanneer het doelpubliek niet bestaat uit mensen voor wie het ontwerpen van software niet het hoofddoel is van hun activiteiten. Deze doelgroep krijgt vaak een specifieke naam: *end user* of eindgebruiker. Wanneer deze personen het doelpubliek zijn, kan het beter zijn om een domeinspecifieke taal te ontwerpen die niet gebaseerd is op een taal zoals UML. Deel III introduceert een domeinspecifieke taal, SpIELan, die gebruikt kan worden voor het modelleren van *staged

\footnote{De drie benaderingen die het dichtst aanleunen bij dit werk worden verder besproken in appendix B.}
**G.2 Ondersteuning van de Software Ontwerper**

*participatory multimedia events.* Dit is een specifieke soort interactieve digitale televisietoepassingen waarvoor een bepaalde soort context, namelijk de overige gebruikers van de toepassing, zeer belangrijk is.

Tenslotte worden in deel IV een algemene conclusie gegeven en nog enkele paden voor toekomstig werk voorgesteld.

### G.2 Ondersteuning van de Software Ontwerper: Integratie met UML


Na deze analyse, wordt in hoofdstuk 3 de Contextual ConcurTaskTrees voorgesteld. Deze uitbreiding introduceert *contextuele taken* in definitie 3.1 op pagina 41 en breidt de associatie van platformen aan taken uit naar gebruikscontext. Tot slot wordt een UML profiel voorgesteld voor de Contextual ConcurTaskTrees dat afgebeeld is in figuur 3.5. Een kritische blik op de Contextual ConcurTaskTrees en het bijhorende UML-profiel rondt het hoofdstuk af.

In hoofdstuk 4, wordt een eerste versie van een UML-profiel, CUP, voor een aantal modellen voor contextgevoelige modellen geïntroduceerd die zo nauw mogelijk aansluiten bij de vereisten voor de specifieke bruikbaarheidsbeïnvloed en voor de modellen voor gebruikersinterfaces maar toch de syntaxis en semantiek van UML zo veel mogelijk volgen. Het voorgestelde profiel heeft bewust een aantal beperkingen wat betreft expressiviteit: er wordt in de definitie van het profiel voor geconcentreerd op de dynamische integratie van context zoals gedefinieerd in hoofdstuk 2.

Figuur 4.3 op pagina 57 biedt een overzicht van dit UML profiel. Het bevat stereotypes voor de definitie van drie modellen: het activiteitenmodel (op basis van het activiteitendiagram), het abstract presentatiemodel (op basis van het deployment diagram) en het context model (op basis van het klassendiagram). Een vierde model, het applicatiemodel (gebruik makend van het klassendiagram) vereist geen extra stereotypes.
Het profiel werd geanalyseerd door middel van het Cognitive Dimensions Framework van Thomas Green waarover meer informatie te vinden is in appendix A om een ruwe indruk te krijgen van de bruikbaarheid. Contacten met software ontwerpers brachten echter aan het licht dat het gebruik van het deployment model voor de voorstelling van het abstract presentatiemodel voor verwarring kan zorgen. Verder was er ook wat onduidelijkheid in verband met enkele aspecten van het activiteitenmodel, zoals toegelicht in sectie 4.5. Deze commentaren werden verwerkt bij het ontwerp van een nieuwe versie van het UML-profiel CUP.

Hoofdstuk 5 introduceert de tweede versie van het UML-profiel. Deze versie van het UML-profiel biedt ondersteuning voor het modelleren van een ruimer gamma aan contextgevoelige applicaties. Met behulp van dit profiel kunnen ook contextgevoelige gebruikersinterfaces worden gemodelleerd die sequentieel of in parallel gebruikt worden op meerdere platformen en in meerdere gebruikscontexten. Daartoe werden er belangrijke aanpassingen gedaan aan de modellen die reeds gedefinieerd konden worden met behulp van de de eerste versie van CUP en werden een extra model toegevoegd.


Twee modellen uit de eerste versie van CUP, namelijk het applicatiemodel en het contextmodel, werden in CUP 2 samengevoegd in een nieuwe versie van het applicatiemodel omdat uit toegenomen ervaring bleek dat er soms vrij grote overlap was tussen de inhoud van deze modellen. Wanneer men de relevante delen van het UML-profiel uit versie 1 (zie figuur 4.3(b) op pagina 57) en versie 2 (zie figuur 5.3 op pagina 80) vergelijkt, is duidelijk zichtbaar dat de metaklassen waarop twee van de gedefinieerde stereotypes van toepassing waren aangepast werden om krachtigere modellen mogelijk te maken.

Het nieuwe contextmodel, zoals het gedefinieerd is in sectie 5.4 op pagina 81 en volgende, laat de gebruiker toe om de relevante gebruikscontexten te modelleren. Dit brengt de inhoud van het context model meer in overeenstemming met het context model zoals het gebruikt wordt in benaderingen die
het gebruiken van het contextmodel in UsiXML [Limbourg 04b].

Het abstracte gebruikersinterfacemodel wordt in CUP 2.0 gedefinieerd als een extensie van het UML klassendiagram in plaats van het UML deployment diagram in de eerste versie van CUP. Deze verandering maakt dat deze specificatie meer in lijn loopt met het gebruik van de diagrammen voor software-ontwerp waarin de structuur van software vaak wordt uitgedrukt met behulp van het klassendiagram.

De stereotypes die relevant zijn voor dit model worden getoond in figuur 5.10 op pagina 89. Uit deze figuur, het voorbeeld in figuur 5.11 op pagina 5.11 en de bijbehorende uitleg in sectie 5.6 blijkt duidelijk dat het abstracte gebruikersinterfacemodel de volledige structuur van de gebruikersinterface kan modelleren, inclusief de relaties tussen de verschillende logische onderdelen van de gebruikersinterface, in gerelateerd werk ook wel presentation units (presentatie-eenheden) genoemd.

De toegankelijkheidsspecificatie (private, protected, public, en dergelijke) van attributen en operaties wordt in het abstracte gebruikersinterfacemodel gebruikt om de toegankelijkheid van de verschillende componenten van de gebruikersinterface door de gebruiker van het gemodelleerd systeem aan te duiden. Voor de juiste betekenis die deze specificaties krijgen in het gedefinieerde model verwijzen we naar sectie 5.6.

Het laatste model waarvoor een stereotype is gedefinieerd in CUP 2.0, is het gebruikersinterface deployment model, waarin beschreven staat hoe een abstracte gebruikersinterface gerealiseerd wordt op een bepaald platform in een specificeerde gebruikscontext.

Hoofdstuk 6 geeft aan hoe de modellen gedefinieerd in hoofdstuk 3 en hoofdstuk 5 gecombineerd kunnen worden in een modelgebaseerde methode voor het ontwerp van een contextgevoelige gebruikersinterface. Tevens worden deze modellen geanalyseerd aan de hand van de criteria die werden opgesteld in secties 2.2 en 2.4 voor het evalueren van het gerelateerd werk. Tenslotte wordt een besluit gegeven over de bijdragen geleverd in deel II van dit proefschrift.

**G.3 Naar ontwerp door de eindgebruiker**

Deel III introduceert een domeinspecifieke modeleertaal, SpIeLan, die toelaat om een bepaalde soort contextgevoelige applicaties te ontwikkelen: staged participatory multimedia events (SPME). Met deze term worden interactieve televisietoepassingen met een hoge graad aan participatie aangeduid die ontwikkeld en gedirigeerd kunnen worden door eindgebruikers. Het is belangrijk
Samenvatting

dat een gebruiker van zulke toepassingen zich bewust is van een bepaald soort
context, namelijk de andere gelijktijdige gebruikers.

In hoofdstuk 7, worden de drie modellen waaruit SpIeLan bestaat bespro-
ken. Twee van deze modellen zullen het vaakst door de eindgebruiker gebruikt
worden: het scenariomodel en het schermmodel. Het scenariomodel laat toe
aan een gebruiker om de sequentie van scènes op te bouwen waaruit de SPME
zal bestaan en wordt in detail besproken in sectie 7.5. Een voorbeeld van een
scenario is afgebeeld in figuur 7.7 op pagina 125.

Elk van de scènes is gekoppeld aan een schermmodel (of screen model)
waarin de opbouw van het scherm voor alle gebruikers compact wordt voorge-
steld. Alle componenten van de gebruikersinterface kunnen op een abstracte
voorstelling van het scherm worden geplaatst. Omdat de gebruikers van een
SPME verschillende rollen kunnen hebben, worden alle componenten van de
gebruikersinterface ook gekoppeld aan de rollen. Hoe dit gebeurt wordt be-
sproken in sectie 7.6.

Het is echter belangrijk om op te merken dat er voor de componenten van
de gebruikersinterface op een abstracte manier worden voorgesteld door middel
van een aangepaste versie van Canonical Abstract Prototypes [Constantine 03a]
(zie ook appendix C). Deze notatie bleek al een effectief communicatiemiddel
om met eindgebruikers gebruikersinterfaces te bespreken. Voor deze specifieke
toepassing drongen er echter zich wat aanpassingen op. Naast de eerder ver-
melde koppeling van de componenten aan rollen werd het ook nodig bevonden
om specifieke notaties te ontwikkelen voor componenten die informatie over
medegebruikers tonen. Figuur 7.9 op pagina 128 toont een voorbeeld van een
schermmodel. Hierin is een van de ontwikkelde notaties te zien in de rechter-
bovenhoek.

Tenslotte is nog een derde model, het template model, dat ook gekoppeld
is aan een scène. Dit model legt de relatie van de componenten van de gebrui-
kersinterface met de componenten op de servers en van de componenten op
de servers onderling vast. Het is echter niet de bedoeling de de eindgebruiker
die een SPME ontwerpt vaak met dit model zal werken. Beperkte aanpassin-
gen aan dit model kunnen indirect gedaan worden door aanpassingen aan het
schermmodel. Het opbouwen van deze modellen is enkel nodig wanneer er een
nieuw type scène, een template, wordt aangemaakt. Meer details over het tem-
plate model worden gegeven in sectie 7.7 waarin tevens een voorbeeld van zo’n
template model wordt getoond in figuur 7.12 op pagina 135. Twee belangrijke
elementen van dit template model zijn de Interaction en de MultiUserInterac-
ton die een compacte beschrijving toelaten van de interactie van componenten
van de gebruikersinterface met componenten op de servers waarvan de SPME
G.4 Conclusies en Toekomstig Werk

Deel IV geeft het algemeen besluit van dit proefschrift en biedt een overzicht van enkele mogelijkheden voor toekomstig werk. Dit besluit wordt geformuleerd in de vorm van de bijdragen van dit proefschrift aan de onderzoekswereld die samengevat kunnen worden in twee grote bijdragen. Een eerste bijdrage betreft de beschrijving van modellen die toelaten om een ruim gamma aan contextgevoelige gebruikersinterfaces te modelleren. Hiervoor werden enkele UML profielen gedefinieerd die toelaten om deze modellen in reeds bestaande software-ontwerpomgevingen te gebruiken.

Een tweede bijdrage is de domeinspecifieke taal SpIeLan voor het ontwerp van *staged participatory multimedia events*. Specifieke bijdragen, zijn onder andere de compacte voorstelling van de abstracte schermlayout voor meerdere rollen of de Interaction en MultiUserInteraction (zie sectie 7.7.2) en het scenario-model wat een compacte voorstelling geeft van een SPME aan de hand van de scènes waaruit de SPME bestaat en de interactiviteit die er voor de verschillende deelnemers aan de SPME mogelijk is.

Tenslotte biedt sectie 9.2 nog enkele richtingen voor toekomstig werk die vooral handelen over de definitie van modeltransformaties, het uitbreiden van SpIeLan met meer laag-niveau informatie en de definitie van een meta-model dat toelaat om elementen uit de modellen uit deel II direct herbruikbaar te maken, wat het consistent houden van de modellen zou vereenvoudigen voor zowel de gebruiker als de bouwer van de ontwerpomgevingen.

g.4 Conclusies en Toekomstig Werk

In hoofdstuk 8, worden de mogelijkheden van SpIeLan verder verkend. Sectie 8.1 bespreekt de bestaande software-ondersteuning en hoe er abstracte interactieve prototypes gegenereerd kunnen worden op basis van het scenario-model en het schermmodel zodat er al snel een globale indruk kan gegeven worden van het verloop van de gemodelleerde SPME.

In sectie 8.2 wordt SpIeLan verder geanalyseerd en ingekaderd in het geheel van dit proefschrift. De expressiviteit van de modellen van SpIeLan wordt ook besproken aan de hand van de modeelementen die voorgesteld werden in hoofdstuk 2 voor het bespreken van andere modelgebaseerde benaderingen voor het ontwerp van gebruikersinterfaces. Daarnaast wordt het chatvoorbeeld dat gebruikt werd voor de bespreking van SpIeLan in hoofdstuk 7 gemodelleerd met enkele modellen die werden besproken in deel II. Tenslotte wordt een gebruikerstest van SpIeLan besproken en wordt een besluit getrokken voor deel III in secties 8.2.3 en 8.3.
Samenvatting